Virtual Network Migration on Real Infrastructure: A PlanetLab Case Study

Samantha Lo, Mostafa Ammar, Ellen Zegura
School of Computer Science,
Georgia Institute of Technology
Email: {samantha,ammar,ewz}@cc.gatech.edu

Marwan Fayed
Computer Science and Mathematics
University of Stirling
Email: mmf@cs.stir.ac.uk

***Abstract*—Network virtualization enables the deployment of novel network architectures and services on existing Internet infrastructure. In addition, virtual networks (VNs) can share the resources in the physical substrate. To enable efficient resource reallocation and network agility, VNs must sometimes *migrate*, i.e., change their placements on a substrate network. While VN placement, and to a lesser extent migration, has been studied in the past, little attention has been devoted to deploying and evaluating these functions over a real infrastructure. In this paper, we study the VN migration problem based on network virtualization in PlanetLab. We create a tool, PL-VNM, that orchestrates the VN migration on PlanetLab for a given new VN placement. The design and deployment of the tool reveal challenges and constraints. Some are particular to PlanetLab while others apply more generally to any virtualized infrastructure. Most significantly, we find that while in principle one can specify a migration schedule (sequence of migration steps) as an input to our tool, certain PlanetLab features make VN migration scheduling very difficult if not infeasible. Our work leads to recommendations about the features of a general virtualization environment and specific recommendations for PlanetLab that enable VN migration and migration scheduling. We believe that the recommended features make long-term experiments and application deployments on PlanetLab and other realistic virtualized infrastructures possible.**

I. Introduction

The critical importance of the Internet makes direct experimentation hard. The ensuing gap between development and deployment poses problems when evaluating novel network architectures. Network virtualization is a process that enables live experimentation, while providing isolation ([1], [2], [3], [4], [5]). In this context, a virtual network (VN) is built on top of a substrate network and is granted a portion of the substrate physical resources. This decoupling of virtual networks from the physical substrate provides opportunities for flexible VN management when allocating substrate resources [6], [7], [8], [9]).

Figure 1 shows by example multiple VNs that are built on top of the same substrate network. The nodes and links of a VN are mapped to selected substrate nodes and paths, respectively. Resources are mapped according to the requirements of the VN and their availability. These requirements are described by the policy of the application running on this VN [10]. Available resources in the substrate are dependent on mappings and utilization: A substrate node can host multiple VN nodes

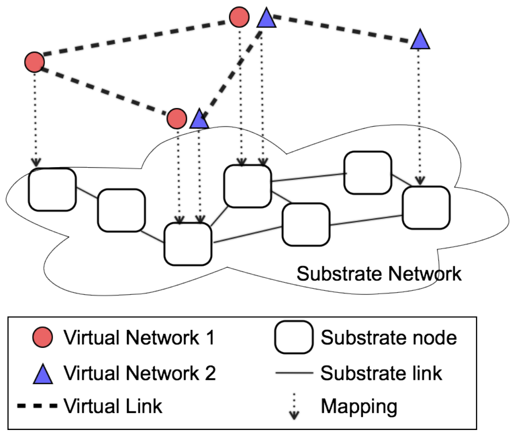


Fig. 1. Virtual networks are mapped to a substrate network.

that share the resources of the substrate node; similarly, each of the substrate links can also host multiple links from different VNs.

Over time, a static mapping of VNs may lead to over- or under-utilization of the substrate resources. As a consequence, long-term benefit can only be maximized by redistributing substrate resources. This means re-mapping virtual components from one set of components of the substrate network to another, when there are changes in the traffic carried on the VN [11], [10], [6], [7], [8], [12], [13].

During the remapping process, we migrate the VN from its *initial placement* to its *final placement*. To enable efficient resource allocation and network agility, a VN must be able to change its placement on the physical network quickly. However, there is a lack of systematic exploration of this VN agility feature. Prior work investigates migration of a single virtual router [14]. In our previous work we use this technique to devise and simulate VN migration schedules [15]. Other work considers live network migration that utilizes Software-Defined Networking techniques [16]. Still, this leaves open questions pertaining to deployment and evaluation of VN placement and migration over real infrastructure.

In this paper, we specifically study the VN migration problem on PlanetLab [17]. PlanetLab is a prominent virtualized infrastructure for deploying multiple VNs that share the substrate network resources distributed across hundreds

ISBN 978-3-901882-58-6 © 2014 IFIP

of physical domains on the Internet. A user requests a slice on PlanetLab and selects a set of PlanetLab nodes to add to the slice. Once the user selects a set of nodes for the experiment, the user can connect these nodes together to form a VN within the slice. However, some of the nodes may fail or become unstable over time [18]. This can have a negative impact, for example, on experimental results, as well as the ability to experiment in the long-term. An effective virtual network migration system could mitigate negative effects on PlanetLab and similar platforms, while facilitating application development.

A VN migration mechanism should minimize any disruption introduced to the VN. It is also desirable that the process of VN migration be automated and relatively fast. We propose a migration mechanism for VN migration on PlanetLab designed to meet these objectives.

Given that most of the users can request only one slice for their experiments, we focus on VN migration within a single PlanetLab slice. We design, implement, and evaluate with live experiments, a controller called *PL-VNM* that orchestrates the VN migration to new VN placements on PlanetLab. In principle, we can schedule the migration steps to minimize the disruption to the VN. In practice, our experiments have revealed that certain PlanetLab features make VN migration scheduling infeasible. Our work leads to specific recommendations for PlanetLab that would enable VN migration scheduling, as well as recommend features for more general virtualization environments. We believe that the recommended features make long-term experiments and application deployments on shared virtualized infrastructures possible.

We first give an overview of the PlanetLab architecture and how VNs are implemented on PlanetLab. Then we present the architecture of our proposed migration controller PL-VMN in Section III. In Section IV, we discuss the challenges of VN migration scheduling to minimize the disruption to the VN in the ideal case. We further experiment with gateway scheduling in PL-VNM as the realistic case in Section V. We then give recommendations in Section VI for enabling VN migration scheduling on both PlanetLab environment and generic virtualization environment. We conclude in Section VII with future work.

II. Virtual Networks on PlanetLab

In this section we briefly describe two well known methods for implementing and managing VN resources in PlanetLab. We begin first by discussing the ways in which VNs may be allocated the resources that are granted by PlanetLab.

A. *VN-to-PlanetLab Resource Allocation*

PlanetLab uses a *slice* abstraction to gather a collection of resources across different PlanetLab components. Slices are requested by a user. Once allocated, the user can add nodes to the slice. All nodes are shared by different slices. For each node that is added to the slice, a virtual container is created. This container holds the node within the slice, and isolates the node from other slices. This virtual container can be accessed by the user with an appropriate SSH key.

Figure 2 shows two types of virtual containers. Circles, triangles, and diamonds represent the virtual containers that are allocated to slices 1, 2, and 3, respectively. The shapes colored white are not actively participating in any routing or forwarding. We call them the non-active virtual containers. The colored shapes represent the virtual containers that are allocated to the slices and have virtual routers running on them. We call them the active containers.

Consider a slice that is allocated entirely to a single VN. In this allocation the slice and the VN are identical, and include all resources, active and non-active containers. In this context the migration of a VN in PlanetLab means migrating either the slice itself, or all the active and non-active virtual containers from nodes in the current slice to nodes in another slice. Either type of migration requires changes to the infrastructure on PlanetLab and the way it assigns resources to different slices. This capability is not currently available, nor is it possible, in the PlanetLab environment.

Alternatively, it is possible to partition resources within a single slice. This feature can be used to build multiple VNs *within* a single slice. The VN is first mapped to a subset of virtual containers. These virtual containers are active and hold the VN's virtual routers. Virtual routers are connected virtual links using point-to-point tunnels. In Figure 2, a VN of slice 1 is placed on the three active virtual containers (colored circles) and the virtual links between them. The three non-active virtual containers of slice 1 (white circles) are not part of the VN. The substrate resources available for a VN placement and migration are confined to the resources of a slice. For example, migration of the VN in slice 1 can be done by migrating the active containers to the three non-active containers within slice 1.

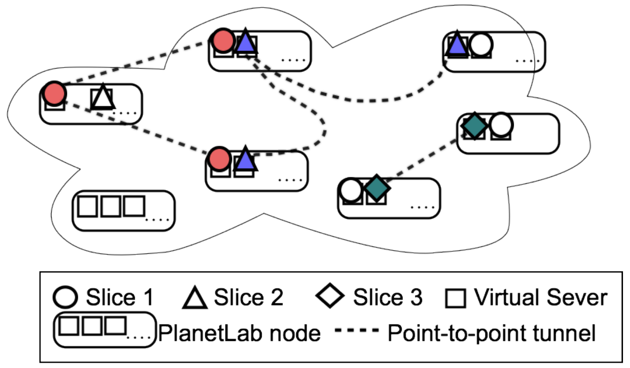


Fig. 2. An example assignment of VN nodes within PlanetLab

B. *Implementation and Management of VNs on PlanetLab*

There are two common methods to create and manage a VN within a slice on PlanetLab. The first implements VNs in user space. This requires the creation of a virtual router in each container using the Click router [19], and connecting the virtual routers together with UDP tunnels [20]. User space implementations increase the latency of forwarding packets due to the time copying packets between the kernel and user space and also the time waiting for Click to run on the CPU.

In our design and evaluations we have selected to implement VNs in kernel space by setting up packet forwarding in kernel space using the Vsys API [21]. With Vsys, a virtual router in a PlanetLab node can install routes to the forwarding table in the kernel space. When a PlanetLab node receives packets for one of the slices it hosts, it does not copy the packets from kernel space to user space before the redirection. Instead, the packets are forwarded directly according to the forwarding table of the node. Thus, the latency of packet forwarding within a node is considerably lower compare to using the user space routers [22].

Vsys currently is fully functional on the PlanetLab Europe nodes. It allows each slice to have access to certain super user `sudo` privileges, that include creating virtual interfaces and tunnels and forwarding table entries for that slice. Each user is assigned a private IP subnet for the slice. The forwarding table in the kernel is shared among other slices. Isolation is provided through Vsys such that actions performed on a PlanetLab node in a slice do not affect other slices sharing the same PlanetLab node. A user can only modify the forwarding table entries that belong to the assigned IP subnet. With this kernel space forwarding table sharing feature, a user space router can install routes to the forwarding table in kernel space directly.

PlanetLab uses ingress filtering to prevent packets from forged IP source addresses. The packets are filtered according to the strict mode of unicast Reverse Path Forwarding (uRPF) [23]. The packets are still forwarded directly in kernel space according to the forwarding table entries. However, packets are filtered based on their source IP addresses. When a PlanetLab node receives a packet from one of its virtual interfaces, the node checks for the best route of the packet's source IP address in the forwarding table. If the virtual interface of that best route does not match with the receiving virtual interface, i.e., the arriving virtual interface is not used to forward packets to that source IP address, the packet will be dropped. If it passes the check, the node looks for the best route of the destination IP address and forwards the packet accordingly. Note that only the best route is considered. Other existing routes in the forwarding table for the same IP address or subnet with lower priority are ignored. This feature prevents asymmetric paths on the VN between two nodes and has negative implications on VN migration, as shown in Section IV.

To setup VNs, we use a Python Vsys API package provided by NEPI [24]. According to the VN topology, we connect the PlanetLab nodes with point-to-point tunnels through the Vsys API. We assign private IP addresses within the assigned subnet of our slice to the virtual interfaces for those tunnels. Then we install the pre-computed routing table entries to the forwarding tables of the PlanetLab nodes through the Vsys API. In our current implementation, we assume that we are using static routing. In the future implementation, we can install a routing daemon to interact with the forwarding table in the kernel space through the Vsys API for updating forwarding table entries. [1]

III. Virtual Network Migration on PlanetLab

We now describe the proposed virtualization architecture that supports VN migration in PlanetLab, the migration process and how to evaluate it.

A. *Migration Architecture and Process Overview*

At the center of our VN migration architecture we have designed and implemented our own migration controller, PL-VNM. Gateways are used in addition to redirect traffic between virtual networks. An example instantiation of our architecture is shown Figure 3. Shown in the upper-right, PL-VNM orchestrates the VN setup and migration remotely. It connects to all the virtual nodes in the VN and the gateways to set up the VN and the virtual links between the nodes.

We focus on connections between VN virtual routers and gateways hosted by PlanetLab nodes in the same slice. While it is possible to connect end hosts that are not PlanetLab nodes through UDP tunnels, in our experiments, we choose to maintain all the virtual nodes including the end hosts on PlanetLab. Figure 3 shows the initial and final placements of the VN and they are mapped to different sets of PlanetLab nodes. Only the VN is migrated to its final placement in our experiment. The end hosts and gateways maintain their placements.

In our VN migrations, we focus on network elements between gateways. Since migration is done in the IP layer, one of the roles of the gateways is to hide the IP route changes from the end host applications. If end hosts were directly connected to the VN, then when the VN migrates to its final placement, the interfaces of the end hosts would also have to adopt new IP addresses. Such changes of IP addresses adversely affect the connectivity of the applications.

Besides hiding IP address changes from end hosts, the gateways also perform traffic redirection during VN migration. The changes of routes in the forwarding tables of the gateways determine which placement of the VN the packets should be forwarded to. For example, in Figure 3, the forwarding table of gateway g_i starts with the routes to destination e_j and e_k with the next hop to the initial placement of the VN. During the VN migration, these routes are changed to reflect the next hop as the final placement of the VN.

[1]Prior work investigated the interaction between dynamic routing in the native and overlay layers [25]. With dynamic routing and a high node degree VN topology, the migration process can be improved through rerouting. The VN migration process is similar except for handling the HELLO messages among the virtual routers. For example, in OSPF, if the time of the migration is longer than the HELLO message timer, the effects of the migration will be recognized as a link failure or a network instability event. Currently the recommendation for the time between sending HELLO messages is 10 seconds which is longer than the time for migration, which takes less than 2 seconds. So, the migration cannot be noticeable by the virtual routers. In case it is noticeable, it will be recognized as a link failure and the traffic will be rerouted. As a result, we believe our work still applies to the virtual networks with dynamic routing.

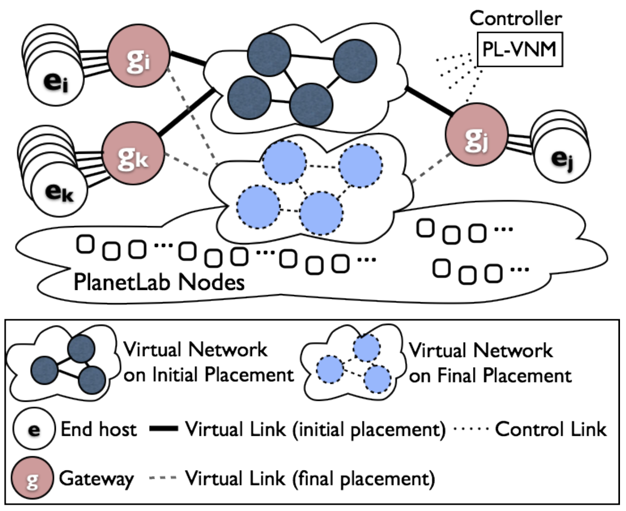

Controller
PL-VNM
g_i
e_i
g_k
e_k
g_j
e_j
PlanetLab Nodes
Virtual Network on Initial Placement
Virtual Network on Final Placement
e End host
Virtual Link (initial placement)
Control Link
g Gateway
Virtual Link (final placement)

Fig. 3. An example VN migration setup in our PlanetLab experiment

The process of VN migration proceeds as follows:

1) Launch new virtual routers on the final placement nodes.
2) Clone the routing states from the active to the final virtual routers.
3) Redirect traffic at the gateways to the final VN placement.

During the migration, PL-VNM installs the required packages at the final placement nodes. Then PL-VNM clones the routing states [2] of the virtual routers on the VN to the virtual routers at the final placements. After cloning, PL-VNM issues traffic redirection commands to all the gateways g_i, g_j, and g_k to change the forwarding table entries with next hop to the final placement of the VN nodes.

B. Migration Evaluation Metrics and Migration Scheduling

We evaluate migration performance by measuring packet loss and the time to complete the migration. In our architecture packet loss can occur at the gateways when the old route to the VN's initial placement is replaced by a new route to the final placement. Because the forwarding and redirecting of traffic are done at the routing layer, the packets are only forwarded to the best route from the forwarding tables of the gateways. Furthermore, because of the aforementioned uRPF check, packets that are buffered on the initial placement of the VN can be dropped or lost during the traffic redirection.

In prior work [15], [16], it has been established that migration scheduling, i.e., the sequence and timings of gateway redirection requests can have a significant effect on migration performance. Also, there is often a tradeoff between migration duration and VN disruption as measured by packet loss. For example, in our system if all gateways are asked to redirect their traffic at the same time, this will cause the migration to complete quickly. This, however, is likely to cause the most packet loss to applications.

Implementing a migration schedule, however, requires the network control (to deploy the required traffic redirection) at small time granularity. As it turns out this is very challenging in the PlanetLab environment. In our work we explore two possible control strategies and evaluate them experimentally as is described in Section V.

C. The PL-VNM Controller

Our controller, PL-VNM, performs both VN installation and migration. During a VN installation or migration, PL-VNM connects to all the virtual routers and gateways through SSH connections to initiate and synchronize the VN installation or migration procedures. We implement PL-VNM with the Python Vsys API package. The package has to be installed on all nodes in the slice. The user also has to request to add the Vsys tags to the slice.

Figure 4 shows its detailed architecture. PL-VNM stores the VN topologies, substrate network topology, and the mapping of the VNs with SQLite database [26]. The *VN orchestrator* installs the VNs according to the mappings with the Python Vsys API to the PlanetLab nodes. The VN orchestrator installs the virtual links between the virtual routers and installs routes to the forwarding tables of the virtual nodes through the API.

When a request for VN migration is initiated with initial and final VN placements, PL-VNM schedules and automates the migration. The *migration routing state translator* identifies the routing states that are affected at the gateways and translates these routes according to the final placement of the VN. With the routing state changes from the migration routing state translator, the *migration orchestrator* first computes a *migration schedule*, and orchestrates the migration according to the schedule. The migration orchestrator installs new translated routes to the VN's final placement PlanetLab nodes. After that, according to the migration schedule, it sends redirection commands to the gateways to redirect traffic to the VN's final placement. These commands change the routes in the forwarding tables of the gateways. Once that is finished, the routing state updates are sent back to PL-VNM with the final VN mapping.

In our prototype, we have two choices of implementation for controlling traffic redirection at the gateways. The first choice is to execute the commands according to the schedule produced by the algorithm at PL-VNM through SSH sessions to the gateways. With this technique, called *remote scheduling*, there is a lag between the time the control command is issued by PL-VNM and the execution time of the command at the gateway. This lag can be considerable and unpredictable and makes it difficult, if not impossible, to control the actual migration timing.

By contrast we schedule at the gateway nodes with `at` job scheduling utility [27]. With this technique, called *gateway scheduling*, the gateways should be synchronized through NTP [28]. PL-VNM computes the migration schedule and uses `at` to schedule the commands at different gateways. In this

[2]The *routing states* of the PlanetLab virtual routers are defined as the entries in the forwarding tables of the virtual routers on PlanetLab. We clone the routing states instead of copying them to the final placement nodes directly because the virtual interfaces and IP addresses of these virtual interfaces are assigned differently on each virtual node.

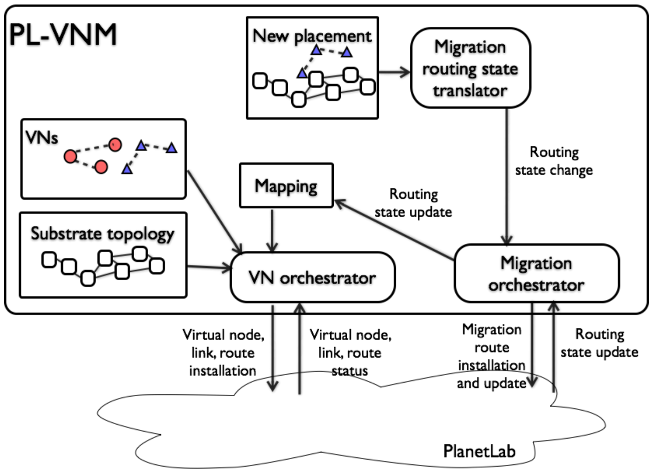


Fig. 4. The detailed architecture of PL-VNM

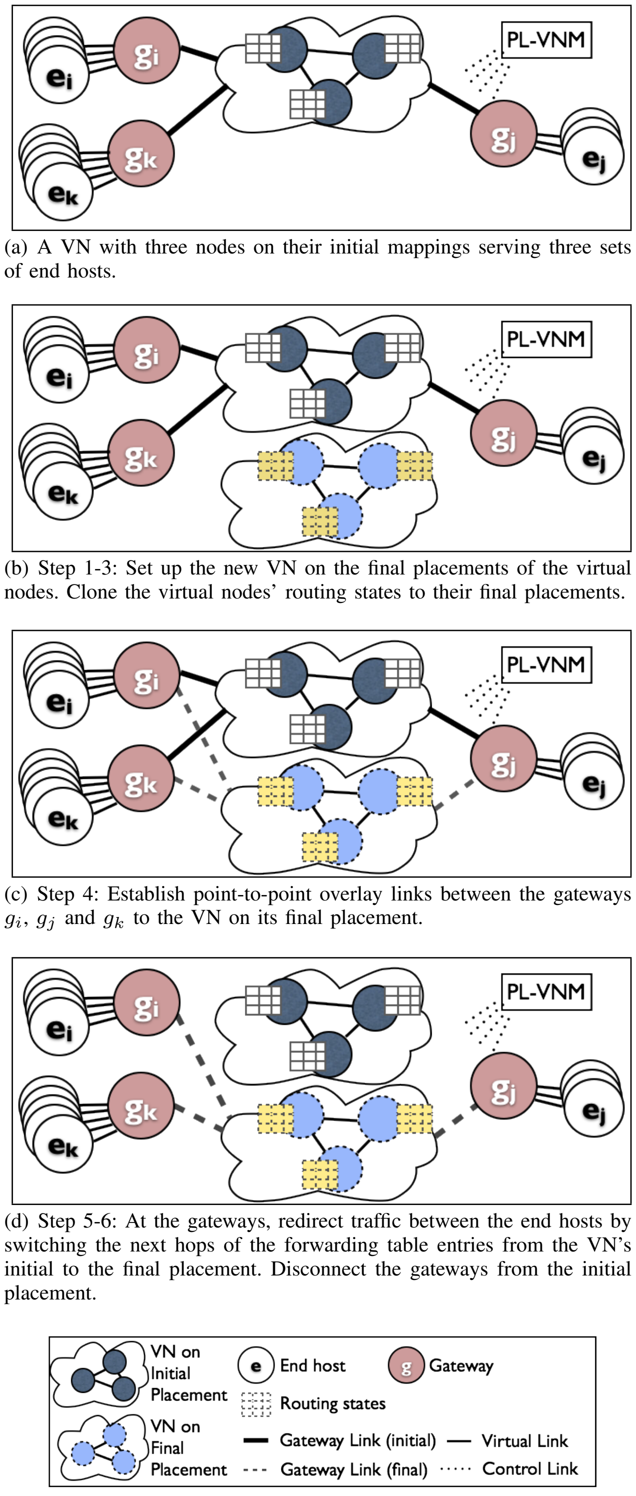
case the latency between PL-VNM and the gateways is not an issue. However, OS process timing resolution and NTP accuracy can, again, make it hard to control migration timing. We explore the relative performance of these two approaches in our experiments in Section V.

D. Our Process for VN Migration in PlanetLab

The migration process is described by the steps below, and accompanied by matching depictions of each step in Figure 5:

1) Add the PlanetLab nodes for the new node placements to the slice such that the virtual containers for the slice are created on those nodes. This step can be done through the PlanetLab API or the PlanetLab website.
2) Set up the new VN on the final placements of the virtual nodes:
   a) Set up virtual interfaces on each new virtual router according to the configuration of the old virtual routers.
   b) Connect the virtual routers at their final placements according to the VN topology with point-to-point tunnels through the Vsys API.
3) Clone the routing states of the old virtual routers to the new virtual routers of the VN. The routing states cannot be directly copied from the old virtual router because the virtual interfaces of these states are different.
4) Establish point-to-point overlay links between the gateways to the VN on its final placement.
5) PL-VNM issues commands to the gateways according to the schedule to redirect traffic to the new VN. These redirection commands change the forwarding table entries on the gateways between the VN and the end hosts. This step requires synchronization among all the gateways.
6) Disconnect the old VN from the gateways and remove the routing states on the old virtual routers.

Step 1-4 and 6 can be done in the background. Step 5 requires accurate redirection timing in order to implement a given migration schedule.



(a) A VN with three nodes on their initial mappings serving three sets of end hosts.

(b) Step 1-3: Set up the new VN on the final placements of the virtual nodes. Clone the virtual nodes' routing states to their final placements.

(c) Step 4: Establish point-to-point overlay links between the gateways g_i, g_j and g_k to the VN on its final placement.

(d) Step 5-6: At the gateways, redirect traffic between the end hosts by switching the next hops of the forwarding table entries from the VN's initial to the final placement. Disconnect the gateways from the initial placement.

Fig. 5. Migrating a VN with three virtual nodes serving three sets of end hosts e_i, e_j and e_k.

IV. VN Migration Scheduling Challenges in PlanetLab

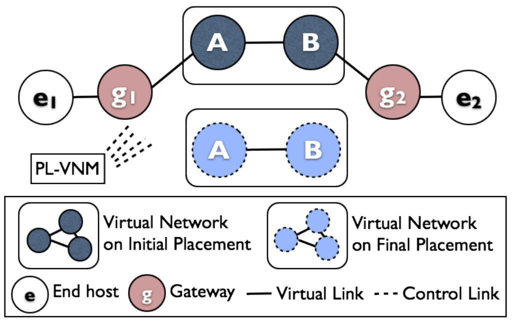

A
B
e1
g1
g2
e2
PL-VNM
Virtual Network on Initial Placement
Virtual Network on Final Placement
End host
Gateway
Virtual Link
Control Link

Fig. 6. The topology of the 2-node VN on PlanetLab

We illustrate PlanetLab VN migration scheduling challenges here through a 2-node VN example as shown in Figure 6. We consider $f_{1,2}$ a traffic flow from end host e_1 to e_2. $f_{1,2}$ has a virtual path of $(e_1, g_1, A, B, g_2, e_2)$. Nodes A and B move from an initial placement to a final placement through redirection of traffic at both g_1 and g_2. PL-VNM issues traffic redirection commands to g_1 and g_2. We assume these redirection commands are executed (*not* issued by PL-VNM) at times t_1 and t_2, respectively.

We define l_i and l_f as the one-way path latencies of the virtual paths of the flow $f_{1,2}$ from e_1 to e_2 at the initial placement and final placement, respectively. Assuming that the capacity of the virtual paths (g_1, A, B, g_2) of the initial and final placements are the same, we consider three cases:

1) When $l_i = l_f$: After g_1 switches from the initial mapping path to final mapping path at t_1, the last packet traveled on the initial mapping path takes l_i to arrive to g_2. At the same time, the next packet (after the redirection) takes the final mapping path and arrives to g_2 at $t_1 + l_i$. Since $l_i = l_f$, g_2 should switch to the final path at time $t_1 + l_i$ to prevent packet loss.
2) When $l_f > l_i$: Similar to the previous case except that the packets on the final mapping path take a longer time to arrive to g_2. As a result, the time for g_2 to switch to the final path should fulfill $t_1 + l_i < t_2 < t_1 + l_f$ or $l_i < t_2 - t_1 < l_f$.
3) For the case of $l_i > l_f$, packet loss is unavoidable. Once g_1 has issued the redirection command at t_1, the packets after t_1 switch to the final mapping path with latency $l_f < l_i$. The first packet on the final path reaches g_2 earlier than the last packet on the initial path. No matter when g_2 switches the path, packet loss would be observed. To minimize packet loss, g_2 should switch the path between t_1 and $t_1 + l_i$, i.e., $t_1 \leq t_2 \leq t_1 + l_i$.

Note that the above analysis is for one direction of traffic flow. Avoiding packet loss in the opposite direction of traffic (from e_1 to e_2 in the above case) will most likely require different timing considerations for the gateway redirection. Because PlanetLab requires symmetric paths, it is not possible to manage the two flow directions independently.

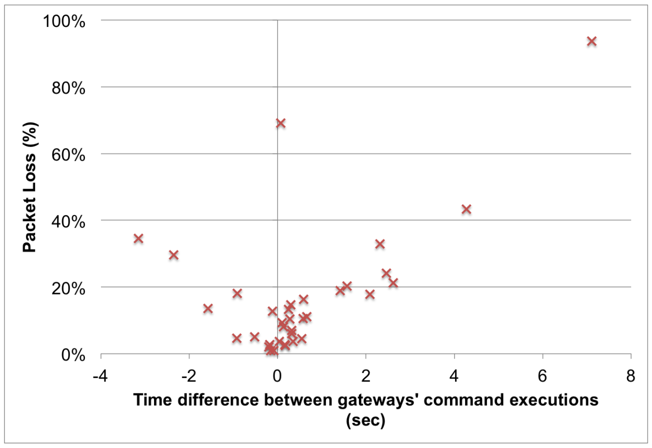


Fig. 7. Percentage of packet loss vs. redirection command execute time at g_2 minus redirection command execute time at g_1, i.e., $(t_2 - t_1)$

Experimental results: We now show results from an experiment on the topology in Figure 6. We migrate the VN from its initial placement to its final placement 35 times. We use iperf to send a UDP flow with 1.5 Mbps rate from e_1 to e_2 for 10 seconds. PL-VNM issues the redirection commands to the two gateways at the same time. Under ideal conditions this will cause the simultaneous redirection at both gateways. However, as noted earlier, various factors can cause the actual execution times to be quite variable. In fact, we observe values for $t_2 - t_1$ ranging from -7.1 to +3.2 seconds. This confirms our earlier assessment of the difficulty of controlling migration timing in PlanetLab.

Figure 7 shows a scatter plot of the packet loss versus the time between the redirection commands executed on each gateway (i.e., $t_2 - t_1$). Each point on the figure represents one measurement result. The path latencies in the initial and final placements are 240 ms and 160 ms, respectively. We measure the percentage of packet loss within 10 seconds of the traffic redirection. This falls under case 3 in the discussion above. Thus we expect packet loss to be minimized when $t_2 - t_1$ is less than 240 ms. Indeed we do observe packet loss at a minimum (with some experimental variation, at and around this interval). The minimum time difference of $t_2 - t_1 = 89$ ms gives a minimum packet loss of 0.009%. However, there are also other cases with roughly a time difference of 75 ms with 79% packet loss.

V. Gateway Scheduling in PL-VNM

In this section, we evaluate whether gateway scheduling (where redirection tasks are scheduled in advance at the gateways themselves) can provide better timing control for VN migration in PlanetLab. We do this using our PL-VNM prototype to control a 3-node VN migration experiment (Figure 8). The VN has three virtual routers that are connected to three end hosts through three gateways. In our experiment, we migrate the VN between its initial and final placements back and forth 110 times. We also use iperf to measure the packet loss for 60 seconds between all pairs of end hosts, i.e., total of 6 UDP flows with 1.5Mbps rate among 3 end hosts.

A. Remote Scheduling Baseline

We first run baseline experiments using remote scheduling. During the migration, PL-VNM first issues commands to clone the VN on the VN's final placements. Then PL-VNM issues redirection commands to the gateways for switching the flow to the final VN placement at the same time through remote scheduling (SSH). We record the time of command execution at the gateways and measure the packet loss for the flows.

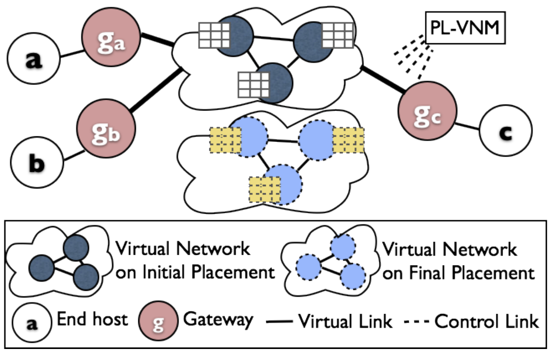

a
g_a
g_b
b
g_c
c
PL-VNM
Virtual Network on Initial Placement
Virtual Network on Final Placement
End host
Gateway
Virtual Link
Control Link

Fig. 8. The topology of the 3-node VN in our PlanetLab experiment

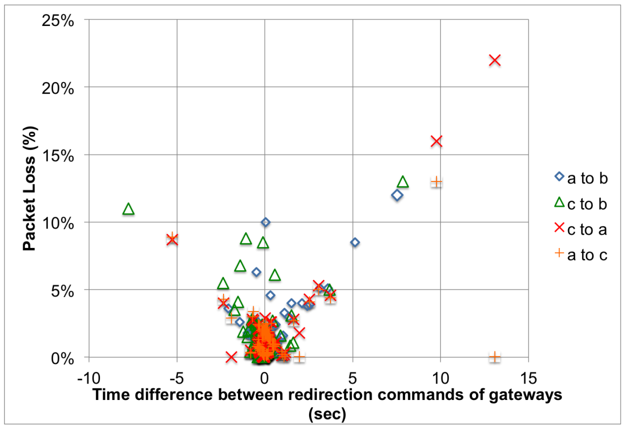


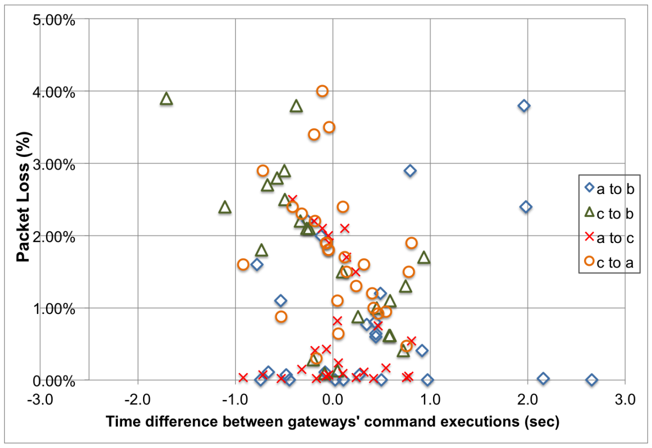
Fig. 9. Packet loss vs time difference between the execution time of the related gateways' redirection commands using PL-VNM remote scheduling on the 3-node VN with only 4 flows

Figure 9 shows the result of this experiments with remote scheduling. The percentage of packet loss on the y-axis is based on the total measurement time of 60 seconds with iperf. The x-axis shows the time difference between the traffic redirection command issued times to the gateways for the corresponding flows. Even though the redirection commands are issued at the same time, we find the time differences ranging from -7.8 to +13.1 seconds and again the time differences illustrate the serious timing problems of the remote scheduling approach. Flows from node b are not shown in the figure because we notice that flows from node b to any other nodes experience high packet loss. The loss rate on these two flows are too high even there is no migration.

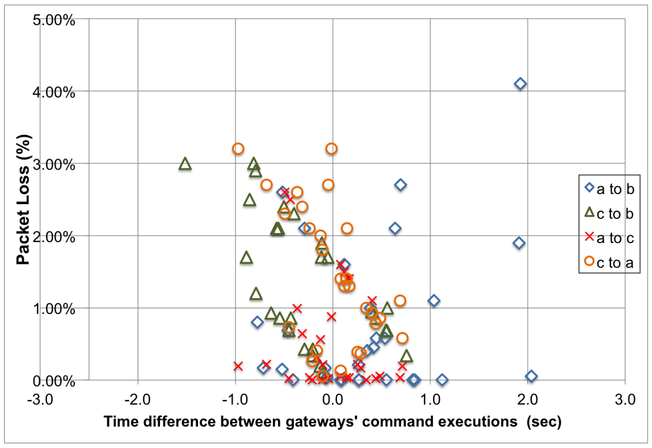
We observe that when there is almost zero time difference between the redirection command execution time at the two related gateways, the packet loss is close to zero. This is because the difference in latencies between the initial and final placement of the paths is also small (tens of milliseconds). We still observe cases that time difference is close to zero while 10% packet loss is observed. This effect can be caused by a background event happening in the substrate network.

B. VN Migration Scheduling through Gateway Scheduling

We now consider migration control via gateway scheduling. We run a set of experiments on the same topology as before. However, we schedule the traffic redirection commands ahead at the gateways with `at` utility at the same time. All the PlanetLab nodes are synchronized with NTP so that the error of the time is limited to 100 ms.



(a) Switching from the initial to the final placement



(b) Switching from the final back to the initial placement

Fig. 10. Packet loss vs time difference between the execution time of the related gateways' redirection commands using gateway scheduling on the 3-node VN with only 4 flows

Figure 10(a) shows the results of the experiments that migrating the VN from its initial to its final placement. Figure

10(b) shows the results of the experiments that migrating the VN from its final to its initial placement. First, the range of time difference is within -1.7 to +2.7 seconds given that the commands on different gateways are scheduled to be executed at the same time. The command execution timings behave much better than in the remote scheduling case. This also causes packet loss to be much less with this control paradigm.

The time variation is caused by the accuracy of NTP synchronization and the load on gateway CPUs. The resources including CPU time and memory of the gateways are shared among different PlanetLab slices. If the gateways are executing some other jobs for other slices, the traffic redirection commands will be prioritized to be executed after the other jobs. Thus, the execution time will be delayed.

Even though timing control is significantly better with gateway scheduling, we still believe it is not adequate to provide the fine-grained control required to minimize loss caused by VN migration.

VI. Recommendations for Platforms to Support VN Migration

When we started this work, our goal was to complete our prototype with a migration scheduling heuristic and to demonstrate how this heuristic can be used to minimize disruption in VN migration on PlanetLab. It is relatively straightforward to take the analysis of the scenarios in Section IV and produce a scheduling heuristic that under ideal conditions will provide the least packet loss when migrating a VN. We show a sketch of such a heuristic in the Appendix. However, on shared infrastructure, the difficulty in controlling redirection timing due to network latency and resource availability makes this an uninteresting exercise.

Instead we conclude by giving some recommendations about substrate features that are required to enable VN migration. Some of these are feasible to deploy on PlanetLab while others point to design elements in a future infrastructure such as GENI [29].

A. Providing Improved Timing Control in Substrate Nodes

The performance and time to execute the migration command is constrained by available synchronization and scheduling mechanisms. We recommend the following features at the substrate network to provide improved timing control. These features also apply to a general substrate infrastructure that supports VN migration.

Reduce clock granularity to facilitate synchronization: VN migration scheduling as described in Section IV depends on the path latencies between end hosts. The path latencies typically encountered are in the few hundred milliseconds range. By contrast, scheduling with the `at` utility is limited to a granularity of seconds. Our evaluations suggest that a job scheduler with a granularity of 1 ms at the gateways is appropriate. Quartz, for example, is a commercial-grade java-based job scheduler that shows job scheduling with a granularity of 1 ms is feasible [30].

Enable prioritization of migration commands at gateways: We observe that the time to complete migration commands is characterized by delayed starts in addition to slow execution. Time averages measured over 110 executions are summarized in Table I. Lag time is the time between the scheduled start and actual start times, and averages at least 0.5 seconds at each of the gateways. Furthermore, we see in Table I that execution time is highly dependent on load, with values ranging from a few milliseconds at gateway g_c, to a many hundred milliseconds at remaining gateways. Prioritization of migration commands would reduce these timing effects within a slice. The ideal is for prioritization commands to be respected at the substrate layer, in addition.

TABLE I
THE AVERAGE LAG TIME BETWEEN COMMAND ISSUE TIME AND THE TIME THE GATEWAYS START EXECUTING THE COMMANDS AND THE AVERAGE REDIRECTION COMMAND EXECUTION TIME AT THE GATEWAYS

	g_a	g_b	g_c
Average lag time (sec)	0.536	0.869	0.564
Average execution time (sec)	0.813	0.702	0.066

B. An Asymmetric Routing Policy to Reduce Packet Loss

Currently, PlanetLab supports only symmetric routing, a policy that is enforced by way of the strict mode in uRPF [23]. This policy insists that changes in forwarding behavior must be synchronized across gateways with matching segments along forward and reverse paths, as described in Section II-B. At any time, if the gateways are not synchronized with symmetric paths, i.e. one gateway is using the initial placement to forward packets while the other one is using the final placement, the traffic flowed through these gateways will be dropped. During migration this causes long contiguous steams of lost packets.

To prevent packet loss during the traffic redirections we recommend support at the gateways for feasible mode (see [23]) in uRPF in addition to strict mode. In feasible mode, asymmetric routing is allowed. As long as the forwarding tables maintain simultaneous routes to the initial and final placements, packets will pass the checks and will be forwarded.

VII. Concluding Remarks and Future Work

We believe that our work makes long-term experiments and application deployments on shared virtualized infrastructures possible. We propose a VN migration mechanism for PlanetLab. We design and implement a controller called PL-VNM to orchestrate the VN migration. We evaluate our work with experiments. Although we only perform small scale experiments, the challenges also apply to public networks that support virtualization through the routing layer. Ideally, a migration schedule would minimize the disruption to the VN. However, certain PlanetLab features make migration scheduling infeasible. We further recommend features for general virtualization environments to enable VN migration scheduling.

We note an alternative approach may exist by way of collaboration between the VN migration controller and the end systems. This may be particularly advantageous in cases where it is possible to recover gracefully from the packet loss that occurs within the short disruption period that follows a redirection. Applications that deliver data with an on-off behavior are appropriate targets, with dynamic adaptive streaming over HTTP (DASH) as just one example [31]. In this context a VN controller could trigger migrations between the transmission of video segments. Additionally, the controller could notify the end systems of impending migrations and the need to buffer packets locally until the migration is complete. We leave this investigation for future work.

Acknowledgments

This work was supported by the NSF Grant 1017237 and 1319490. We thank Alina Quereilhac for her help on setting up Vsys and her work on the Python Vsys package.

References

[1] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris, "Resilient Overlay Networks," in *Proc. 18th ACM SOSP*, (Canada), pp. 131–145, Oct. 2001.

[2] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in *Proc. ACM SIGCOMM*, (Pittsburgh, PA), 2002.

[3] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach, "Secure routing for structured peer-to-peer overlay networks," in *Proc. 5th USENIX OSDI*, (Boston, MA), Dec. 2002.

[4] A. D. Keromytis, V. Misra, and D. Rubenstein, "SOS: Secure overlay services," in *Proc. ACM SIGCOMM*, (Pittsburgh, PA), pp. 61–72, 2002.

[5] L. Subramanian, I. Stoica, H. Balakrishnan, and R. H. Katz, "Overqos: offering internet qos using overlays," *SIGCOMM Comput. Commun. Rev.*, vol. 33, pp. 11–16, January 2003.

[6] Y. Zhu and M. Ammar, "Algorithms for Assigning Substrate Network Resources to Virtual Network Components," in *Proc. IEEE INFOCOM*, (Barcelona, Spain), Mar. 2006.

[7] J. Lu and J. Turner, "Efficient mapping of virtual networks onto a shared substrate," *Tech Report WUCSE2006*, vol. 35, no. 2006-35, pp. 1–11, 2006.

[8] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: Substrate support for path splitting and migration," *ACM SIGCOMM Computer Communications Review*, Apr. 2008.

[9] M. Demirci and M. Ammar, "Fair allocation of substrate resources among multiple overlay networks," in *Proc. of MASCOTS*, vol. 0, (Los Alamitos, CA, USA), pp. 121–130, IEEE Computer Society, 2010.

[10] J. Fan and M. H. Ammar, "Dynamic topology configuration in service overlay networks: A study of reconfiguration policies," in *Proc. IEEE INFOCOM*, 2006.

[11] I. Fajjari, N. Aitsaadi, G. Pujolle, and H. Zimmermann, "Vnr algorithm: A greedy approach for virtual networks reconfigurations," in *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, pp. 1–6, 2011.

[12] N. Farooq Butt, M. Chowdhury, and R. Boutaba, "Topology-awareness and reoptimization mechanism for virtual network embedding," in *Proc. IFIP*, NETWORKING'10, 2010.

[13] V. Eramo, E. Miucci, A. Cianfrani, and M. Listanti, "Evaluation of power saving in an mpls/ip network hosting a virtual router layer of a single service provider," in *Proc. ICTC 2013*, (Jeju Island, South Korea), Oct. 2013.

[14] Y. Wang, E. Keller, B. Biskeborn, J. van der Merwe, and J. Rexford, "Virtual routers on the move: Live router migration as a network-management primitive," in *Proc. ACM SIGCOMM*, (Seattle, WA), 2008.

[15] S. Lo, M. Ammar, and E. Zegura, "Design and Analysis of Schedules for Virtual Network Migration," in *Proceedings of the 11th International IFIP TC 6 Conference on Networking*, IFIP Networking 2013, 2013.

[16] E. Keller, S. Ghorbani, M. Caesar, and J. Rexford, "Live migration of an entire network (and its hosts)," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, (New York, NY, USA), pp. 109–114, ACM, 2012.

[17] "PlanetLab." http://www.planet-lab.org/, 2010.

[18] D. Oppenheimer, B. Chun, D. Patterson, A. C. Snoeren, and A. Vahdat, "Service placement in a shared wide-area platform," in *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference*, ATEC '06, (Berkeley, CA, USA), pp. 26–26, USENIX Association, 2006.

[19] "The click modular router project." http://www.read.cs.ucla.edu/click/.

[20] A. Bavier, M. Huang, and L. Peterson, "An overlay data plane for planetlab," in *Telecommunications, 2005. advanced industrial conference on telecommunications/service assurance with partial and intermittent resources conference/e-learning on telecommunications workshop. aict/sapir/elete 2005. proceedings*, pp. 8–14, 2005.

[21] S. Bhatia, G. Di Stasi, T. Haddow, A. Bavier, S. Muir, and L. Peterson, "Vsys: A programmable sudo," in *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'11, (Berkeley, CA, USA), pp. 22–22, USENIX Association, 2011.

[22] Y. Liao, D. Yin, and L. Gao, "Pdp: parallelizing data plane in virtual network substrate," in *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, pp. 9–18, ACM, 2009.

[23] F. Baker and P. Savola, *Ingress Filtering for Multihomed Networks*. Internet Engineering Task Force, 2004. RFC 3704.

[24] "NEPI: Network Experiment Programming Interface." http://nepi.inria.fr/, 2010.

[25] S. Seetharaman and M. Ammar, "On the interaction between dynamic routing in native and overlay layers," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pp. 1–12, April 2006.

[26] "SQLite." http://www.sqlite.org/. URL retrieved November 2013.

[27] "at - execute commands at a later time." http://pubs.opengroup.org/onlinepubs/9699919799/utilities/at.html. URL retrieved November 2013.

[28] "NTP: The Network Time Protocol." http://www.ntp.org.

[29] "GENI: Global Environment for Network Innovations." http://www.geni.net/.

[30] "Quartz: an open source job scheduling service." http://http://quartz-scheduler.org/. URL retrieved November 2013.

[31] T. Stockhammer, "Dynamic adaptive streaming over http–: standards and design principles," in *Proceedings of the second annual ACM conference on Multimedia systems*, pp. 133–144, ACM, 2011.

Appendix

The following is a sketch of a heuristic migration scheduling algorithm:

1) Identify all the directional flows among all the gateways and end hosts.
2) Estimate or measure the flow size of each pair of bidirectional flows $f_{i,j}$ and $f_{j,i}$ between end hosts e_i and e_j.
3) Sort the flows in descending order of flow size or priority.
4) For each flow in the descending order:
    a) Check whether the command execution time t_i and t_j of gateways g_i and g_j, respectively, are set, where g_i and g_j are the gateways that connect e_i and e_j to the VN, respectively.
    b) If not,
        i) Measure the path latency and set the path latency as the time difference between t_j and t_i according to the cases in Section IV.
        ii) Schedule the execution time to execute the redirection commands for the flow according t_i and t_j.