# VegaNet: A Virtualized Experimentation Platform for Production Networks with Connectivity Consistency

**Mingwei Xu and Qi Li, Tsinghua University**
**Patrick P. C. Lee, The Chinese University of Hong Kong**
**Yanhai Peng and Jianping Wu, Tsinghua University**

**Abstract**

To explore the effectiveness of network protocols and services, it is desirable to deploy and evaluate them under a realistic network setting. However, experimentation in production networks is difficult in practice, since it may involve modifications of underlying routers or other hardware components and disrupt the normal operations of existing protocols. We propose VegaNet, a virtual network architecture that provides an experimental platform atop a physical production network. VegaNet uses a lightweight probing mechanism to provide a consistent connectivity view as in the underlying physical production network so that experimentation can be performed under realistic network conditions. It uses virtualization to host multiple experiments on a single physical machine, while reflecting the current connectivity status to each hosted experiment in an accurate and timely manner. We prototype VegaNet and empirically evaluate its effectiveness atop a real-life production network that is currently deployed nationally.

The networking research community regularly proposes new protocols and services for production networks to improve their overall performance. To validate the effectiveness of new protocols and services, it is important to experiment with them under realistic topologies and data traffic patterns. It is ideal to have such experimentation directly performed atop the production networks where the protocols and services are to be deployed. However, experimentation in production networks is a challenging issue, since it may involve changes in the underlying routers or hardware components and disrupt the normal operations of existing applications.

There has been an extensive body of work on the construction of large-scale experimental testbeds for the evaluation of networking protocols and services. There are two classes of experimental testbeds. The first class is *offline*, in which network performance is provisioned in advance and emulated by the replay of network events collected in advance. Examples include Emulab [1] and DETER [2]. However, the controlled environments of off-line testbeds lack the realism, as the experiment environments are relatively static. Another class of experimental testbeds is *online*, in which experimentation is conducted atop a shared physical network infrastructure. Examples include Planetlab [3] and VINI [4], both of which are developed atop the Internet. Online testbeds provide a more realistic setting for network experiments, as they actually reflect the current network conditions of the Internet.

However, there are still design limitations in online testbeds to achieve full realism. In particular, they do not specifically mirror the connectivity view of the underlying physical network. Normally, link failures in physical networks are directly notified by router hardware, and the detection delay is generally within 100 ms [5]. Suppose that a physical link has failed. While the physical network reacts to the link failure (e.g., by rerouting), the physical link failure and the topological change of the physical network may not be immediately exposed to the experimental testbed layered above. Thus, the physical network and the experimental testbed will have different connectivity views, leading to inaccurate observations of network experiments.

In this article, we propose a virtual network architecture called *VegaNet*, which serves as an experimental testbed atop a production network, with an emphasis on mirroring a consistent connectivity view of the underlying physical production network in a timely and accurate manner. This enables network experiments to be conducted in a more realistic setting. To summarize, this article makes the following contributions:

- We analyze the failure traces we collected from CERNET2 [6], a national production network deployed in China. We study the failure characteristics of a typical production network so as to motivate the design of VegaNet.
- We design and implement VegaNet. We propose a lightweight adaptive probing algorithm to detect the status of physical links. Our algorithm extends the Bidirectional Forwarding Detection (BFD) approach [7], such that the probing frequency is reduced if the current traffic load of the physical network is high. This avoids overloading the physical network under heavy traffic load. Our algorithm has a simple design that is as easily implemented as BFD, while preserving accurate and timely failure inference.
- We build VegaNet based on network virtualization, so that it allows multiple experiments to be simultaneously hosted. We prototype VegaNet using off-the-shelf software and protocols for practical deployment. We conduct an exten-


0890-8044/12/$25.00 © 2012 IEEE

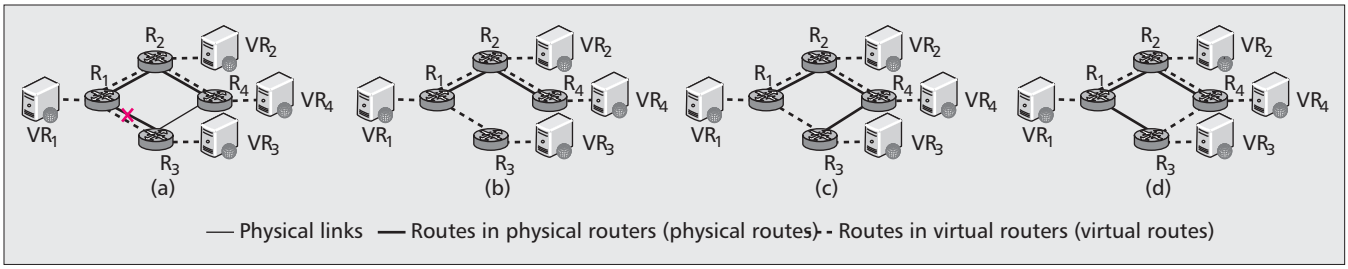


**Figure 1.** *An example: inconsistency between physical and virtual routes occurs at different stages after network failure occurs: a) stage 1: consistency between physical and virtual routes before failure; b) stage 2: inconsistency between physical and virtual routes after failure detection; c) stage 3: inconsistency between physical and virtual routes after rerouting; d) stage 4: inconsistency between physical and virtual routes after failure recovery.*

sive experimental study to validate the practicality of VegaNet atop CERNET2. We deploy VegaNet nodes at 12 sites of the CERNET2 backbone. Our experimental results demonstrate that VegaNet can accurately reflect the connectivity status of CERNET2.

## Problem Statement

We start with analyzing the failure traces collected on CERNET2, a national IPv6 backbone that interconnects more than 200 educational institutions in China. It consists of 25 points of presence (PoPs) that span the whole country. Based on our observations, we motivate the problem of maintaining consistent connectivity views between the virtual and physical networks.

We collected six months of failure events on CERNET2 from August 2010 through January 2011. In the CERNET2 backbone, we deployed eight traceroute servers, each of which generates traceroute probes to all 25 PoPs every 50 ms. Within the six-month span, we observe a total of 12,715 link failures. Among all the 62 links that are covered by our measurements, around 80 percent of them experience at least one failure. The distribution is heavy-tailed, and around 20 percent of the links have at least 1000 failures. All the failures we observed are short-term, and will recover to the normal state afterward. About 13 percent of the failures last for less than 500 ms, and over 96 percent of the failures last for less than 5 s. Thus, link failures are not uncommon, and it is crucial to expose such failures for network experiments in virtual networks.

### Problem of Inconsistency

We are interested in building a virtual network for network experiments atop a production network, such that the virtual network serves as an experimental testbed to reflect the realistic behavior of the underlying network, capturing most (if not all) network events occurring in the network. We observe that most failures are short-term, and it is challenging to capture them. Existing virtual networks (e.g., Emulab [1], PlanetLab [8], and VINI [4]) provide experimental platforms that can address connectivity and resource provisioning. However, to the best of our knowledge, none of the existing virtual networks specifically provides solutions to accurately and quickly capturing the current connectivity status of the physical network. Enabling the virtual network to mirror the physical network connectivity is crucial so that network experiments are conducted under the current data forwarding conditions in the underlying physical network.

Unfortunately, mirroring the physical network connectivity in a virtual network is a nontrivial task. In particular, if the physical network experiences a link failure, then we argue that this can lead to *inconsistent views* in both the virtual and physical networks. To motivate, Fig. 1 shows a small-scale virtual network, in which we attach a virtual router $VR_i$ to a physical router $R_i$, where $i = 1, 2, 3, 4$. Ideally, the virtual network should have a consistent view of the connectivity status as in the physical network. Figure 1a illustrates this ideal scenario, in which no link failure occurs. Suppose now that link $R_1$–$R_3$ fails. Then there are three scenarios where inconsistency can occur:

- *After failure detection* (Fig. 1b). Both physical routers, $R_1$ and $R_3$, detect the failure of link $R_1$–$R_3$ and trigger routing reconvergence. However, virtual routers $VR_1$ and $VR_3$ may still treat link $R_1$–$R_3$ is intact and will not immediately recompute new routes.
- *After rerouting* (Fig. 1c). $R_1$ and $R_3$ reroute and form the route $R_1$–$R_2$–$R_4$–$R_3$, with four hops. If $VR_1$ and $VR_3$ do not update the routing status immediately, they will treat the route $VR_1$–$VR_3$ as directly connected with only one hop.
- *After failure recovery* (Fig. 1d). New routes, such as $VR_3$–$VR_4$, may be formed in the virtual network during the failure of $R_1$–$R_3$. If link $R_1$–$R_3$ is recovered, $R_1$ and $R_3$ revert to using the original route $R_1$–$R_3$ as in Fig. 1a, so the physical route between $R_3$ and $R_4$ switches back to $R_3$–$R_1$–$R_2$–$R_4$. However, $VR_3$ and $VR_4$ may not yet capture this change immediately and still believe $VR_3$–$VR_4$ is a one-hop route.

From the above examples, there are at least two inconsistent views between the virtual and physical networks:

- The virtual network cannot quickly capture link failures observed by the physical network.
- The virtual routes and the physical routes have different hop information.

This inconsistency is further complicated by the fact that network failures are prevalent in production networks, and most of the failures are short-term. Thus, the virtual and physical networks can often have inconsistent views.

To maintain the connectivity consistency, a naive approach is to have the physical routers report the up-to-date connectivity status immediately to their attached virtual routers, but this "bottom-up" approach requires re-engineering of the physical routers and is generally infeasible. A more practical approach is to have virtual routers generate frequent probes using *traceroute*. However, traceroute involves a large volume of probes and may overload the physical network. More important, traceroute may be disabled in physical routers, especially in a production network, due to the privacy issue [9]. Therefore, this article aims to address the following question: *How can we develop a virtual network that provides connectivity and resource provisioning for network experiments as in existing platforms [1, 4, 8], while maintaining accurate and timely consistent connectivity views between the virtual and physical networks?*

## Overview of VegaNet

We propose *VegaNet*, a virtual network architecture that provides an experimental platform atop a physical production network. Its main goal is to maintain a consistent connectivity view between the virtual and underlying production networks

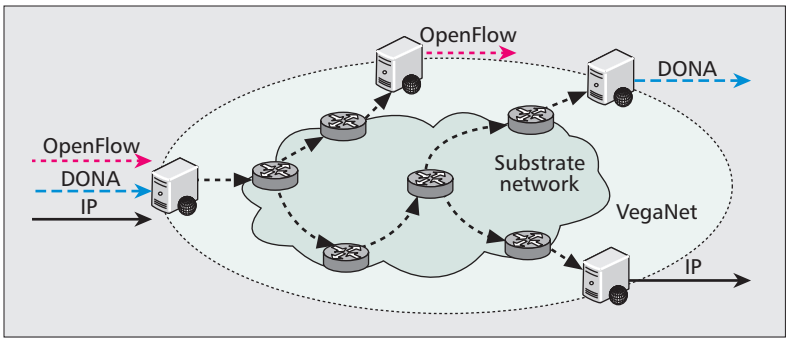


Figure 2. *The architectural view of VegaNet atop a physical network.*

so that network experiments running atop the virtual network are conducted under the realistic data forwarding conditions of the physical network. In this section, we overview the design of VegaNet, and justify the design features achievable by VegaNet.

VegaNet is a network of nodes that can be deployed on regular PCs or servers. Each VegaNet node can be viewed as a software-based *virtual router*. It is based on software implementation and supports the basic routing functionalities as seen in a physical router. It is directly attached to a production router and seeks to maintain the same connectivity view as the production router. In this setting, *virtual* links between two neighboring VegaNet nodes are used to emulate *physical* links between neighboring production routers to which they are attached. To host multiple network experiments, we divide the resources (e.g., CPU, memory) of a VegaNet node into slices, each of which can be independently owned by a network experiment. VegaNet nodes interconnect with each other through tunneling and exchange control information. Note that VegaNet does not require re-implementation of the production routers and hence it will not interfere in the production network operations.

At a high level, a VegaNet node allows different network experiments to have their own routing protocols, network services, and data/control planes. Figure 2 shows an architectural view of VegaNet when it is deployed in a production network. The packets of different new network protocols, such as OpenFlow [10] and DONA [11], can be generated by external users, and multiplexed into VegaNet nodes and their attached physical routers. Packets with different protocols will be encapsulated with new IP packet headers in VegaNet nodes. The destinations of the new packets are set to their neighbor VegaNet nodes according to the corresponding forwarding information base (FIB). Thus, these packets can be delivered between VegaNet nodes by the production routers. Each VegaNet node works like a physical router as it forwards packets over the production network. When the packets reach the destination VegaNet node, they are demultiplexed into the corresponding applications.

At a low level, a VegaNet node provides routing and forwarding separation for different network experiments. Figure 3 shows the internal implementation of a VegaNet node. We divide the routing operations of a VegaNet into different slices for network experiments. We implement a router manager (RM), which is responsible for managing all slices. Each slice has its own control object (CO), which determines the routing policy specifically within the slice (or experiment). Each CO computes the forwarding entries based on its routing policy. The forwarding entries of all COs will then be aggregated in a FIB (flow 1, Fig. 3). There can be multiple FIBs, each of which corresponds to a network protocol with its own address format. For example, DONA and OpenFlow may have their own FIBs. We implement a forwarding object (FO), which manages all FIBs. To forward a packet, the FO looks up the FIB with regard to the network protocol and forwards each packet to the correct network interface. By separating the routing and forwarding functions, the slices do not need to coordinate with each other on how to interact with the low-level (i.e., hardware) network interfaces for packet forwarding, which is now centrally handled by the FO. In addition, inside the RM, we implement a forwarding detection object (FDO), which is responsible for detecting connectivity changes as indicated in the FIB (flow 2, Fig. 3), and notifying the changes to the COs in different slices so that they can recompute new routes (flow 3, Fig. 3).

VegaNet seeks to achieve the following design features:

- *Flexible and realistic experimentation*: We multiplex traffic with different network protocols into the underlying production network. This multiplexing operation is transparent to the user applications. Aside from the user traffic in VegaNet, the underlying production network also carries the regular traffic within its operation. Such traffic provides a realistic workload pattern for network experiments in VegaNet. The similar design feature has also been addressed in the literature (e.g., VINI [4]).
- *Fair resource allocation*: Resources within a VegaNet node, such as CPU and memory, are exclusively used by specific slices (or COs) [12]. Also, based on the existing performance isolation mechanism for virtualization, we can provide fair resource allocation for different slices.
- *Consistent connectivity views*: In each VegaNet node, we unify all forwarding operations of different slices in the FO, which can then easily capture the connectivity status in the physical network by monitoring whether packets can be successfully forwarded. Thus, each experiment hosted in a VegaNet node can obtain a consistent connectivity view with the physical network.

## Achieving Connectivity Consistency

To achieve connectivity consistency in an accurate and timely manner, we leverage *active probing*, in which each VegaNet node sends probes to its neighbors, and determines immediately if there is any connectivity failure or the failure is recovered. Here, we propose a lightweight adaptive probing algorithm that generates probes based on the current traffic

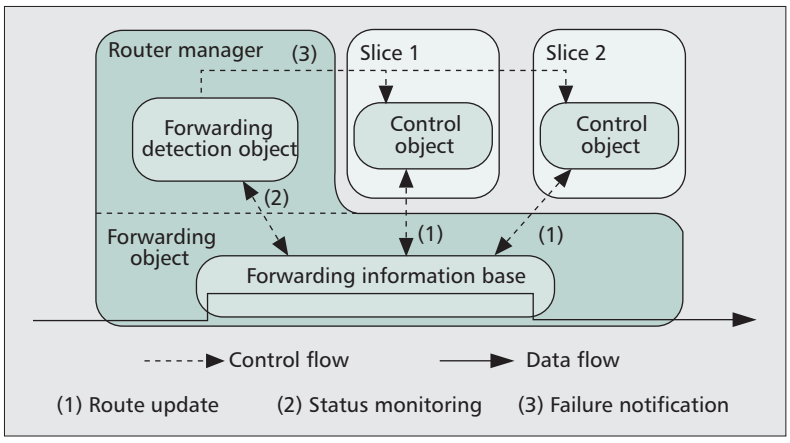


Figure 3. *The internal implementation of each VegaNet node.*

conditions. The probing algorithm is implemented in the FDO of each VegaNet node.

### Session Establishment

In VegaNet, each pair of *neighboring* VegaNet nodes will have a *session*, which is used for negotiating control parameters and monitoring connectivity changes in the physical network between the nodes. A session can be viewed as a *virtual link* between the neighboring VegaNet nodes. Here, we maintain sessions by extending the state machine design in BFD [7] in a virtualized network. Although we leverage the basic procedure of BFD for probing, BFD keeps generating probes based on the negotiated probing intervals, without taking into account the current traffic conditions in the data plane. If the data plane is occupied with application traffic, the probes may further overwhelm the network capacity. We adapt the BFD design to account for the current network traffic conditions, as detailed later.

Each BFD session has three states:

- DOWN: the session is torn down
- INIT: the session is to be initiated
- UP: the session is established

During session establishment, the pair of neighboring VegaNet nodes (call them $VR_1$ and $VR_2$) will negotiate two sets of parameters: the *hop count* and the *probing parameters*. For the hop count, its main goal is to maintain the connectivity consistency between $VR_1$ and $VR_2$. The hop count is defined as the number of physical links between two nodes. For example, if the physical routers $R_1$ and $R_2$ are directly connected, the hop count is given by 3, which includes links $VR_1$–$R_1$, $R_1$–$R_2$, and $R_2$–$VR_2$. We then set the time to live (TTL) field in the IP header of the probing packets to be equal to the hop count (e.g., 3 if $R_1$ and $R_2$ are directly connected). The intuition is that if the link between $R_1$ and $R_2$ fails and packets are rerouted, the VegaNet nodes will not receive probing packets from each other, as each probing packet can traverse at most three hops. Thus, the VegaNet nodes can infer that rerouting occurs and can align its connectivity status with that in the physical routers.

For the probing parameters, they are mainly used for specifying the time for detecting a network failure. For each VegaNet node $VR_i$, the probing parameters include three constants [7]: the Desired Minimal TX Interval ($TX_i$), Required Minimal RX Interval ($RX_i$), and Detect Multiple ($DM_i$). $TX_i$ and $RX_i$ are the minimum sending and receiving intervals supported by the VegaNet node $VR_i$, respectively, while $DM_i$ denotes the probing timeout period represented as a multiple of the probing interval. Some typical values of such parameters are $TX_i$ = 50 ms, $RX_i$ = 50 ms, and $DM_i$ = 3 (e.g., [13]). This implies that $VR_i$ sends probes every 50 ms and expects to receive probes from each of its neighbors every 50 ms. If $VR_i$ does not receive probes from a neighbor for $RX_i \times DM_i$ = 150 ms, it may declare that the link to the neighbor is failed. The probing parameters are included in the control packets during the session establishment process so that they can be agreed upon by both of the neighboring VegaNet nodes. It is important to note that each VegaNet node can customize its own set of probing parameters, for example, according to its available link bandwidth.

### Lightweight Failure Identification

In VegaNet, each VegaNet node sends probes to each of its neighbors to determine if the virtual link between the node itself and the neighbor has failed, according to the current traffic conditions. VegaNet consists of two types of packet flows:

- The *control packet flow*: session establishment and probing
- The *data packet flow*: application traffic generated by the user applications

Both the control and data flows are interleaved and forwarded by the FO in each VegaNet node. The intuition here is that if a VegaNet node (say $VR_1$) can always send data traffic to its neighbor (say $VR_2$), we can infer that the virtual link $VR_1$–$VR_2$ is good and has no failure, without the need to generate additional probes. On the other hand, suppose the neighboring VegaNet nodes have no data traffic in between, because either the user applications do not generate any data traffic or the virtual link has failed. In this case, they generate probes based on the negotiated probing parameters. If no probes are received between the nodes, we can infer that the virtual link has failed.

**Input:** $n$ neighboring VegaNet nodes and probing parameters: $\{(VR_1, TX_1, RX_1), \ldots, (VR_n, TX_n, RX_n)\}$;
**Output:** set of failed links: FailedLinkSet;
1: **for** $VR_i$ in $VR_{[1..n]}$ **do**
2: set timer $\rho_i$'s value = $DM_x \times \max(RX_x, TX_i)$;
3: randomly select $r \in [\beta_{min}, \beta_{max}]$;
4: set timer $\tau_i$'s value = $r \times \max(TX_x, RX_i)$;
5: **end for**
6: **for** $VR_i$ in $VR_{[1..n]}$ **do**
7: **if** (exists traffic sent to $VR_i$ before timer $\tau_i$ expires) **then**
8: reset timer $\tau_i$ before it expires;
9: **else**
10: send a probe with the negotiated hop count;
11: **end if**
12: **if** (no traffic from $VR_i$ when timer $\rho_i$ expires) **then**
13: Add failed link $VR_x$ – $VR_i$ to FailedLinkSet;
14: **end if**
15: **end for**
16: return FailedLinkSet;

**Algorithm 1.** *Adaptive probing algorithm in $VR_x$.*

Algorithm 1 shows the pseudo-code of our adaptive probing algorithm, which is called by each VegaNet node (say VRx) to initiate the sending of probing packets. It extends BFD [7] to account for the data plane condition. First, $VR_x$ initializes the packet receiving timer (denoted $\rho_i$) and the packet sending timer (denoted $\tau_i$) associated with its neighboring VegaNet node $VR_i$ (steps 1 to 5). For $\rho_i$, it is set to $DM_x \times \max(RX_x, TX_i)$. For $\tau_i$, it is set to $r \times \max(TX_x, RX_i)$, where $r$ is uniformly selected at random between two constants $\beta_{min}$ and $\beta_{max}$ such that $0 \le \beta_{min} < \beta_{max} < 1$. Here, we choose $r$ at random so as to avoid self-synchronization of probing among the neighboring VegaNet nodes [7]. Also, we generally set $\beta_{max}$ less than 1 to ensure that the probes reach the other side before the detection timeout (e.g., they may be delayed due to congestion). Here, in our current implementation, we set $\beta_{min}$ = 0.75 and $\beta_{max}$ = 0.9.

For each neighbor $VR_i$ in $VR_{[1..n]}$ of $VR_x$, if there is traffic sent from $VR_x$ to $VR_i$, $VR_x$ can reset the timer $\tau_i$ instead of generating additional probes (steps 7–8). Also, if a probe is to be generated, the TTL of the probe is set to the negotiated hop count to ensure that the probe reaches the neighbor only if the underlying link works (i.e., there is neither failure nor rerouting) (steps 9–10). In the meantime, $VR_x$ checks if it receives any traffic from $VR_i$ before the packet receiving timer $\rho_i$ expires. If not, it means that the virtual link $VR_x$–$VR_i$ has failed (steps 12–14).

Algorithm 1 follows the simplicity of design as in BFD [7] and does not add complicated logic, making it easily implementable. We show that the simple design sufficiently achieves our goal of maintaining the connectivity consistency between the virtual and underlying production networks.

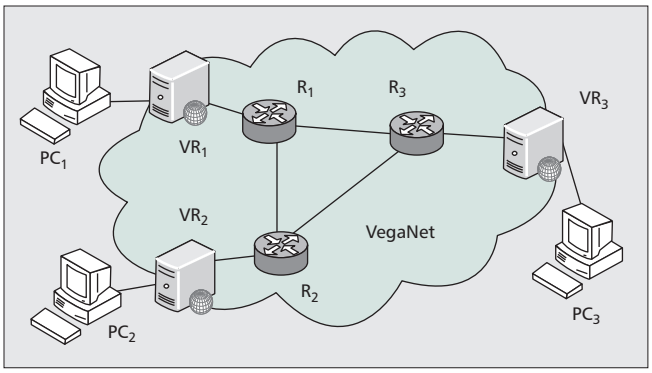


**Figure 4.** *Testbed topology.*

## Evaluation

We implement a proof-of-concept prototype for VegaNet. VegaNet is built on off-the-shelf software and protocols. Each VegaNet node is built on the virtualization platform Xen [12]. We conduct microbenchmark and macrobenchmark experiments for our VegaNet prototype. We deploy VegaNet in a small-scale testbed (Fig. 4) and CERNET2.

**Summary of results:** We show that VegaNet can immediately detect link failures in the underlying physical network (experiments 1–2) while generating fewer probing packets than BFD [7] (experiment 3). Also, VegaNet captures well the behavior of Open Shortest Path First (OSPF) convergence occurring among the physical routers (experiment 4). Finally, we validate that VegaNet behaves correctly when it is actually deployed in the production network CERNET2.

### Microbenchmarks in Testbed Deployment

In microbenchmark experiments, we aim to study different performance aspects of VegaNet subject to the characteristics of real-life failure traces. To achieve this, we replay the failure events we collect from CERNET2 on a small-scale VegaNet testbed as shown in Fig. 4. Our VegaNet testbed is a full-mesh topology composed of three software-based physical routers that form a physical network. Each physical router is attached to a VegaNet node. Each of the physical routers and VegaNet nodes is deployed on an Intel Xeon Quad Core machine that runs Linux. We create a single guest domain, DomU, in each VegaNet node, and form a virtual network atop the testbed. We install the open source router software Quagga [14] in each physical router and the DomU domain in each VegaNet node. Both the physical and virtual networks run the OSPF protocol.

To simplify our study, we replay all failure events only on the physical link $R_1$–$R_2$. To regenerate a link failure, we disable the link $R_1$–$R_2$ for a duration recorded in the corresponding failure event, and re-enable the link.

We argue that the performance of VegaNet is generally independent of network size and the number of links being simultaneously monitored. Note that VegaNet nodes detect the connectivity status by investigating the data and control traffic between each neighboring node pair *independently*. The effectiveness of our link failure detection between one neighboring node pair is mainly related to the communication performance of that pair, instead of being influenced by the failures occurring in other node pairs.

*Experiment 1 (Failure Detection Delay in VegaNet)* — In this experiment, we evaluate the connectivity consistency between VegaNet and its underlying physical network. We measure the *failure detection delay* in VegaNet, defined as the duration starting from when a failure occurs in the physical link R1-R2 until the failure is captured in Dom0. We configure the following probing parameters: TX = RX = 50 ms and DM = 3. Figure 5a shows the failure detection delay in VegaNet vs. the failure events. We observe that the failure detection delay is in the range of 90–135 ms. Note that it is smaller than the maximal detection time DX × max(TX, RX) = 150 ms, as a failure occurs during the interval of waiting for the next probe after the last one. The results indicate that VegaNet detects failures correctly with regard to the configured probing parameters.

*Experiment 2 (Failure Duration Observed in VegaNet)* — We now compare the deviation of the failure duration observed in VegaNet. Figure 5b shows the failure duration observed in Dom0 of a VegaNet node vs. the actual duration of corresponding physical failures, which last for less than 5 s, accounting for 96 percent of the failures we collected. In general, there is negligible deviation between the failure durations in VegaNet and the physical network. The communication overhead between the physical and virtual networks is fairly insignificant with respect to failure duration. We also evaluate the differences of the failure durations observed in Dom0 and DomU. We observe that the detection delay between Dom0 and DomU is minimal, and the ratio is bounded with 0.05 percent.

*Experiment 3 (Failure Detection under Different Background Traffic)* — We now evaluate how our adaptive probing algorithm fully leverages user traffic to reduce probing overhead for failure detection. We re-conduct experiments 1–4 with user traffic generated between $VR_1$ and $VR_2$, and make similar observations we present above. In this experiment, we only compare the communication overhead introduced by VegaNet with and without background traffic (i.e., user traffic).

Figure 5c shows the cumulative number of probing packets generated by VegaNet with and without background traffic. Note that without background traffic, our probing approach is in essence the BFD approach [7], which generates probing packets at a constant rate. In the experiment, we shut down link $R_1$–$R_2$ at time 20 s and re-enable it at time 40 s, and the whole experiment period lasts for 60 s. First, we launch one DomU in $VR_1$ and $VR_2$ and generate FTP traffic between them in the time range from 10 to 60 s. We observe that the number of probing packets is significantly reduced when background traffic is present. The number of overall probing packets is reduced by 45 percent. Note that VegaNet will still generate probing packets in the presence of background traffic, as FTP application does not generate traffic in every 50 ms probing interval. In particular, VegaNet still generates probing traffic in two directions during 40 to 45 s although the link failure recovers, because TCP will not send FTP traffic once the failure recovers.

Moreover, we launch two DomUs in $VR_1$ and $VR_2$ simultaneously, and generate UDP traffic and FTP traffic between these two DomUs, respectively. As shown in Fig. 5c, VegaNet does not generate probing packets in one direction, but it still generates probing packets in the other direction between 40 s and 45 s since FTP and UDP both do not have traffic in this direction. Overall, we validate that VegaNet will not generate any probing traffic if there is user traffic in each probing interval. Actually, if we directly leverage BFD [7] to detect network events in this experiment, it will generate and inject about 1300 probing packets into the physical network (Fig. 5c). The results demonstrate that VegaNet effectively saves the communication overheads with regard to different background traffic.

*Experiment 4 (OSPF Convergence Performance Comparison between Virtual Routers and Physical Routers)* — We now evaluate if VegaNet maintains consistency of routing convergence as in the underlying physical network. Here, we study OSPF per-

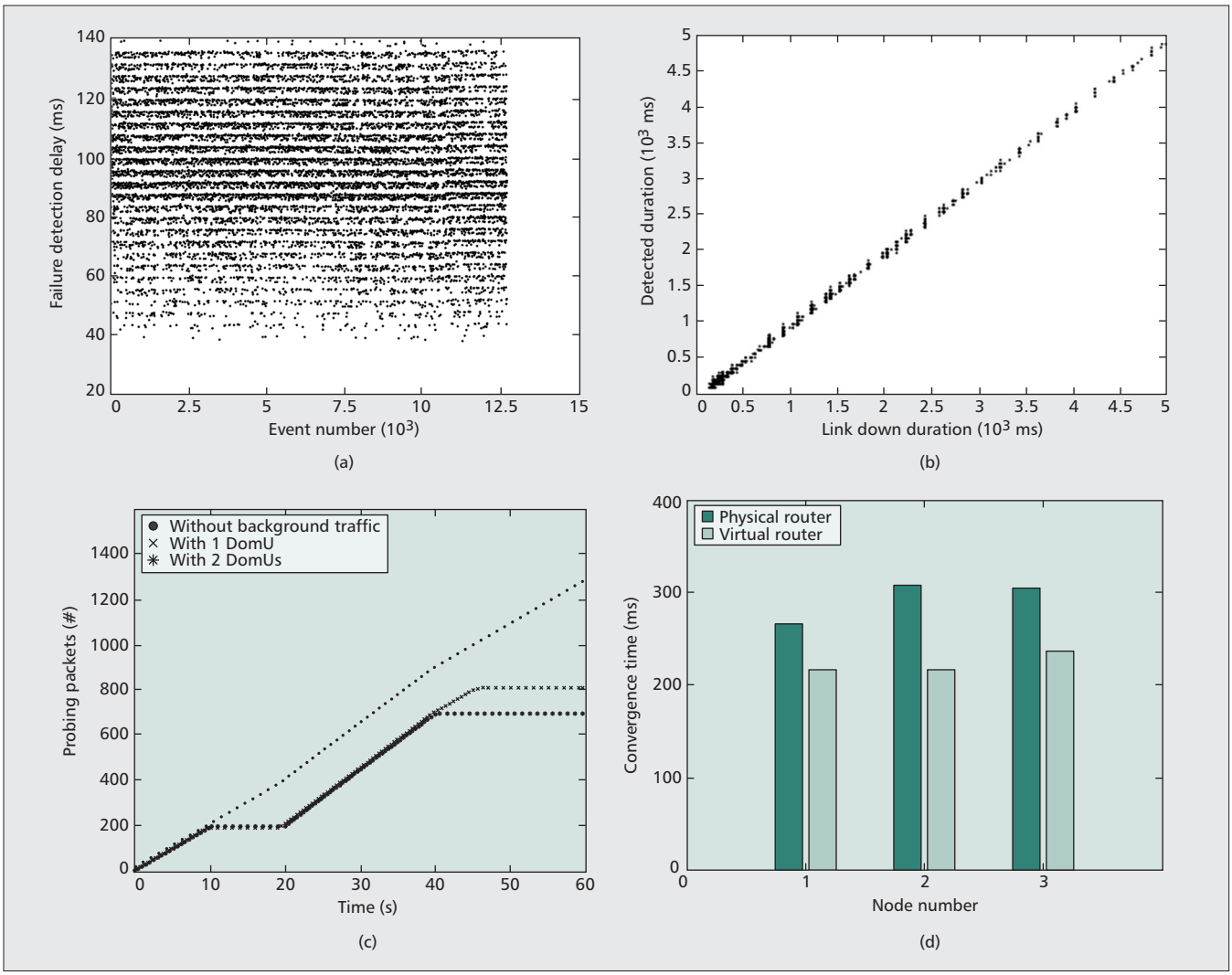


**Figure 5.** *Failure detection performance in testbed: a) (experiment 1) failure detection delay; b) (experiment 2) failure duration; c) (experiment 3) detection overhead; d) (experiment 4) OSPF convergence.*

formance by investigating the log files in physical and virtual routers. In this experiment, we compare the performance with BFD running in physical routers with the same probing parameters setting as in VegaNet (i.e., RX = TX = 50 ms and DM = 3). We shut down link R1–R2 several times and compute the average convergence performance. Each shutdown period lasts about 120 s. We then compare different completion times of OSPF of each router. We also generate FTP traffic as background traffic as in experiment 5.

Figure 5d shows OSPF convergence performance of different routers. We find that the convergence performance in different routers is similar. The virtual routers only have marginal convergence time delay compared to the physical routers by 13.46 percent, because the probing packets in the virtual routers need to traverse more nodes. Overall, each VegaNet node has a routing view consistent with its attached physical router.

## Macrobenchmarks in CERNET2 Deployment

To evaluate VegaNet in a real production network, we deploy our VegaNet prototype atop CERNET2. We deploy 12 VegaNet nodes by attaching them to the backbone routers at 12 sites in 10 cities in China. Note that we only configure and evaluate VegaNet between two neighboring nodes, which can be treated as a full deployment experiment on a partial CERNET2 network.

In our deployment, we again configure the probing parameters TX = RX = 50 ms and DM = 3 for each VegaNet node. We use traceroute data as the baseline. In order to obtain more accurate time records of network events in CERNET2, we reduce the traceroute probing period to 20 ms. Note that here traceroute will capture failures between the VegaNet nodes, but not in the whole network. Thus, within the three-week span, we only observe a total of nine link failures.

*Experiment 5 (Failure Detection and Deviation in CERNET2)* — In this experiment, we measure the failure detection delay in VegaNet. Figure 6 shows that *failure detection delay* in VegaNet versus the failure events by traceroute in CERNET2. We observe that the failure detection delay is in the range of 115–190 ms, which is larger than that in our testbed (Fig. 4) because of the larger link transmission delay in CERNET2. The five failure detection delays shown in Fig. 6 are smaller than the maximal detection time DX × max(TX, RX) = 150 ms. In addition, we observe that the later four failure delays are larger than the maximal detection time. The reason is that traceroute may send out probing packets before failures occur and then detects the failures earlier than they occur. The results indicate that VegaNet correctly detects *all* link failures in real deployment.

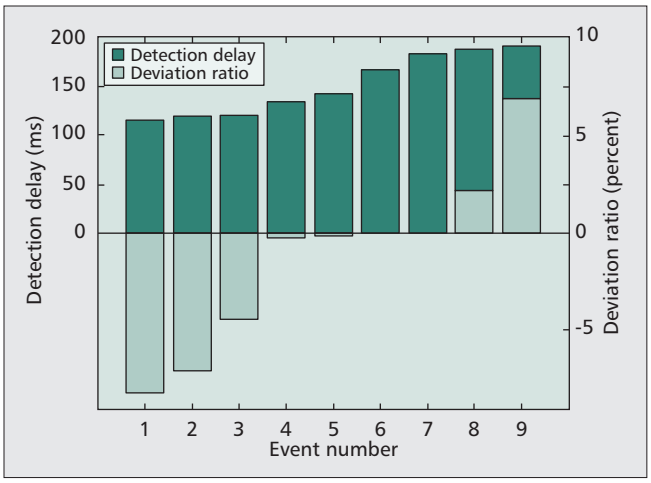


Figure 6. *Experiment 5: failure detection performance in CERNET2.*

We also compare the deviation of the failure duration observed in VegaNet when it is deployed in CERNET2. Figure 6 shows the failure duration observed in VegaNet nodes vs. the detected duration of the corresponding physical failure by traceroute. In general, there is negligible deviation between the failure durations in VegaNet and the detected traceroute results if failure durations are larger (events 4–7 in Fig. 6). However, since some failures we captured in CERNET2 last for a very short duration (i.e., less than 100 ms), the difference ratio of these failures is fairly large. Overall, all failure deviations are bounded within 8.0 percent.

**Summary:** We show that VegaNet can effectively maintain connectivity consistency with underlying production networks, while introducing low overheads into the underlying networks. We have more experiment results to demonstrate the effectiveness and efficiency of VegaNet. For example, we measure and compare packet drop rate in physical networks and VegaNet under different network events. We observe similar results in these two networks. Also, since VegaNet detects network events between two neighboring virtual routers, multiple link failures will not impact failure detection performance. We confirm this by implementing VegaNet in the NS2 simulator and evaluate the case of multiple failure by simulations under real Internet service provider topologies and traffic matrices of Abilene and GEANT. We find that failure detection performance under multiple failures is similar to that under a single failure.

## Related Work

Several studies propose experimental network platforms [1, 4, 8]. Some of them have been deployed atop the Internet and made available to the public, such as PlanetLab [8] and Emulab [1]. These platforms provide a realistic working environment for experimenting with network behaviors. On the other hand, to our knowledge, VegaNet is the first testbed platform designed for production networks, the specialized deployment environments of which may lead to unique characteristics as opposed to the Internet.

Software defined networking (e.g., OpenFlow [10]) uses network virtualization in its designs, which enables concurrent experiments of different network services and protocols in production networks [10]. Different from these SDN designs, VegaNet aims to provide experiment platforms in current production networks with the existing network architecture.

## Conclusions

We propose VegaNet, a virtual network architecture that enables network experiments to be built atop real production networks. VegaNet accurately and timely captures the real-time connectivity status of the physical network, thereby allowing hosted experiments to be conducted in a consistent connectivity view. We propose a lightweight adaptive probing approach to ensure connectivity consistency between real production networks and VegaNet. We prototype VegaNet and extensively evaluate the correctness of VegaNet based on a real-life production network, CERNET2.

### Acknowledgments

This work is supported by the National Natural Science Foundation of China under Grant No. 61073166; by the National Basic Research Program of China (973 Program) under Grant No. 2012CB315806; and by the National High-Tech Research and Development Program of China (863 Program) under Grant No. 2011AA01A101. The work of Patrick P. C. Lee is supported in part by grant GRF CUHK413711 from the Research Grant Council of Hong Kong.

### References

[1] B. White *et al.*, "An Integrated Experimental Environment for Distributed Systems and Networks," *Proc.OSDI*, 2002, pp. 255–70.
[2] Deterlab testbed, http://www.isi.edu/deter/.
[3] Planetlab, http://www.planet-lab.org/.
[4] A. Bavier *et al.*, "In Vini Veritas: Realistic and Controlled Network Experimentation," *Proc. SIGCOMM*, 2006, pp. 3–14.
[5] A. Markopoulou *et al.*, "Characterization of Failures in an Operational IP Backbone Network," *IEEE/ACM Trans. Net.*, vol. 16, 2008, pp. 749–62.
[6] China Education and Research Network 2 (CERNET2), http://www.edu.cn.
[7] D. Katz and D. Ward, "Bidirectional Forwarding Detection (BFD)," RFC 5880, July 2010.
[8] L. Peterson *et al.*, "Experiences Building Planetlab," *Proc. OSDI*, 2006.
[9] L. Subramanian, V. R N. Padmanabhan, and R. H. Katz, "Geographic Properties of Internet Routing," *Proc. USENIX ATC*, 2002.
[10] R. Sherwood *et al.*, "Can the Production Network be the Testbed?," *Proc. OSDI*, 2010.
[11] T. Koponen *et al.*, "A Data-Oriented (and Beyond) Network Architecture," *Proc. SIGCOMM*, 2007, pp. 181–92.
[12] P. Barham *et al.*, "Xen and the Art of Virtualization," *Proc. SOSP*, 2003, pp. 164–77.
[13] Cisco, Bidirectional Forwarding Detection, http://www.cisco.com/en/US/docs/ios/12_0s/feature/guide/fs bfd.html.
[14] Quagga, http://www.quagga.net/.

### Biographies

MINGWEI XU (xmw@cernet.edu.cn) received B.Sc. and Ph.D. degrees from Tsinghua University. He is a full professor in the Department of Computer Science at Tsinghua University. His research interests include computer network architecture, high-speed router architecture, and network virtualization.

QI LI (liqi@csnet1.cs.tsinghua.edu.cn) received B.Sc. and Ph.D. degrees from Tsinghua University. His research interests include network architecture and protocol design, system and network security.

PATRICK P. C. LEE (pclee@cse.cuhk.edu.hk) received his B.E. degree (first class honors) in information engineering from the Chinese University of Hong Kong in 2001, his M.Phil. degree in computer science and engineering from the Chinese University of Hong Kong in 2003, and his Ph.D. degree in computer science from Columbia University in 2008. He is now an assistant professor in the Department of Computer Science and Engineering at the Chinese University of Hong Kong. His research interests are in network robustness and security.

YANHAI PENG (pengyanhai@gmail.com) received B.Sc. and M.Sc. degrees from Shandong University and Tsinghua University, respectively. His research interests include network measurement and network virtualization.

JIANPING WU (jianping@cernet.edu.cn) received Master's and Ph.D. degrees in computer science from Tsinghua University. He is now a full professor with the Department of Computer Science, Tsinghua University. He has published more than 200 technical papers in academic journals and proceedings of international conferences in the research areas of network architecture, high-performance routing and switching, protocol testing, and formal methods.