On the Resiliency of Virtual Network Functions

Bo Han, Vijay Gopalakrishnan, Gnanavelkandan Kathirvel, and Aman Shaikh

The authors explain the resiliency requirements for virtual network functions in order to provide carrier grade services, summarize the existing solutions in the literature, highlight several research challenges, and present a concrete case study to demonstrate how to decompose a type of virtual router and thus enhance its resiliency.

ABSTRACT

Network functions virtualization is an emerging technology that can significantly improve the flexibility of network service provisioning and offer potential cost savings. However, it is critical that service providers offer high reliability and availability of the network and services when moving from proprietary hardware appliances to virtualized network functions on commodity servers. In a network consisting of physical appliances, providers can deploy redundant hardware and extra capacity to handle failures, although this is quite expensive in practice. Virtualization of network functions lead to more challenges for resiliency, but also bring new opportunities to address these challenges in a more cost-effective manner. In this article, we explain the resiliency requirements for virtual network functions in order to provide carrier grade services, summarize the existing solutions in the literature, highlight several research challenges, and present a concrete case study to demonstrate how to decompose a type of virtual router and thus enhance its resiliency.

INTRODUCTION

Many network and telecommunication service providers have started migrating their infrastructure to take advantage of network functions virtualization (NFV) [1]. The idea behind NFV is to replace dedicated network appliances, such as routers and firewalls, with software that provides that same capability on top of commodity servers. Each individual network function, such as a software-based router, runs in a virtual machine and is called a virtual network function (VNF). Figure 1 illustrates the architectural framework of NFV with three major components: VNFs, NFV infrastructure (NFVI), and an accompanying management and orchestration (MANO) framework (e.g., ECOMP [2] from AT&T and the open source ONAP[1] project). The shown VNFs are virtual network address translation (NAT), firewall (FW), router, deep packet inspection (DPI), and VPN Internet gateway (VIG).

While NFV promises significant flexibility and control to network operators, it also brings more concerns for resiliency. Resiliency has been defined as "*the ability of the network to provide and maintain an acceptable level of service in the face of failures and challenges to normal operation*" [3]. Traditional carrier-grade systems have been engineered to offer higher than 99.999 percent (five 9s) availability (translates to roughly 25.9 s downtime per month). Achieving such availability is extremely difficult for VNFs due to multiple reasons: first, the commodity servers that host VNFs are more prone to errors and failures compared to the dedicated hardware appliances [1, 4]. Next, since the software implementations of these VNFs are at their infancy, they can be rather buggy and are susceptible to failures. To address this issue of bugs, operators and VNF vendors are looking toward continuous integration and continuous deployment (CI/CD) to rapidly roll out changes to VNFs. However, these frequent updates again have the potential to impact service unless handled appropriately. Finally, the availability of a single cloud instance depends on the collective availability of the building, mechanical electrical plumbing (MEP), hardware infrastructure (server, storage, and network), cloud orchestration software (e.g., OpenStack), and so on. Given these constraints, most cloud instances are usually designed to offer 99.9 percent (three 9s) availability (43.2 min downtime per month). Unless addressed carefully, the unavailability of VNFs can result in significant down time for customers and may violate the service level agreements (SLAs) that have been made.

Unfortunately, vendors of VNFs have thus far focused on capabilities and performance to match the physical network functions. As a result, they provide very limited resiliency features. Note that a thorough treatment of resiliency demands that we address all aspects of NFV (VNFs, NFVI, and MANO). The reason is that a service's resiliency depends on all these aspects. On one hand, we should design VNFs by taking advantage of the resiliency offered by NFVI and MANO. On the other hand, VNFs should provide capability that NFVI and MANO can leverage to improve their resiliency. However, given the large scope of the topic, we focus specifically on VNF resiliency in this article and briefly discuss the resiliency of NFVI and MANO.

The resiliency feature of VNFs should be able to gracefully handle both planned maintenance (e.g., upgrading VNF software or reconfiguring VNFs) and unexpected failures [5]. However, due to the heterogeneity of devices, velocity of network evolution, and complexity of management, it is challenging to maintain high availability of services, especially when failures happen [5]. Hence, we need mechanisms built into VNFs that can detect and recover from or even prevent failures. With well designed VNF resiliency, we can maintain the desired service level for customers while reaping the benefits provided by NFV.

The authors are with AT&T Labs.

[1] https://www.onap.org/ (Accessed on April 14, 2017).

Digital Object Identifier: 10.1109/MCOM.2017.1601201

For the rest of this article, we first describe the resiliency requirements for VNFs. Next, we review existing solutions that help us meet these requirements. We then highlight some research challenges and future directions. Finally, we provide a concrete case study of EdgePlex [6], a network VNF, by showing how resiliency is achieved by its architecture and design.

Resiliency Requirements for VNFs

This section illustrates the resiliency requirements for VNFs, including failure management, state management, and the awareness of redundancy and correlated failures. We note that although we discuss them separately, in reality they are interconnected and may depend on the infrastructure and MANO resiliency. However, they should avoid being tied too tightly with the infrastructure, which may limit their capabilities.

Failure Management

Similar to other systems, *VNF failure management usually has four parts: design, detection, recovery, and prevention*.

Ideally we want to design VNFs in a way that minimizes the impact of failures. For example, using software quality assurance measures and safe programming languages can prevent many failures. Despite this, in practice, it is difficult to avoid unforeseen failures, and thus VNF design should allow detecting them quickly and recovering from them with minimal impact on the service.

Failure detection covers both deviations from the normal operational behavior of a VNF (not including performance degradation) and errors, for example, by checking the invariants of a building block or an algorithm. The NFV infrastructure/MANO should be able to detect *quickly* both failures that originate from VNFs (e.g., memory crashes) and those associated with the network infrastructure (e.g., link or switch failures).

The goal of automated recovery is to mask a failure to customers. There are five properties desired by any recovery architecture. The first and most fundamental one is correctness. The internal and external state of a recovered VNF instance should be consistent with that before the failure. Second, it should minimize the overhead of failure-free operations, especially the packet-processing latency. Third, the recovery should be fast enough to avoid degrading the end-to-end service, for example, not triggering the TCP timeout (although it may not always be easy to achieve in reality). Fourth, the recovery scheme should be general (e.g.,, not tailored for different VNFs) and should minimize the modifications to VNFs. Finally, it should not significantly increase the operational and management cost.

Failure prevention can take place during runtime by learning from past failures and devising appropriate proactive failure control to prevent them from occurring again. For instance, runtime experience of a VNF may show that CPU utilization above a threshold results in a failure with a high certainty. Given this, once the CPU utilization has reached the threshold, proactive failure control could trigger mechanisms to decrease CPU utilization of the VNF (e.g., by distributing load to other VNF instances).

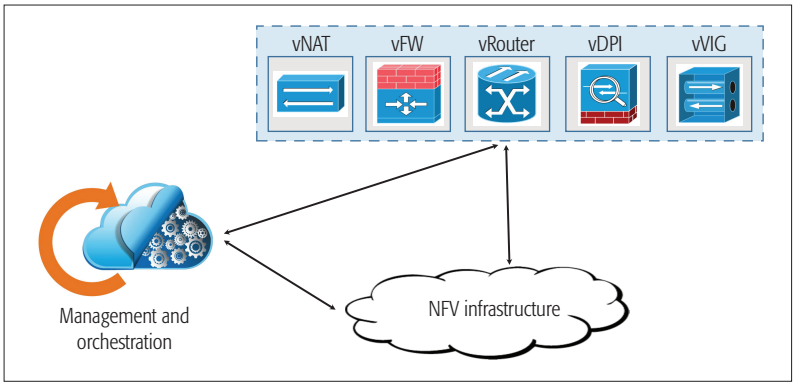

Figure 1. An abstract view of the NFV architectural framework (the concrete view is available in our previous work [1]). In the top dashed box we show a set of VNFs hosted by the NFV infrastructure. Both the infrastructure and VNFs report their status to the management and orchestration, which communicates with VNFs for tasks such as software upgrade and configuration management, and interacts with the infrastructure to spin up, migrate, and shut down VNFs.

State Management

Since most VNFs are stateful, to guarantee service continuity for either unexpected failures or planned maintenance, VNFs need to manage their state intelligently and efficiently or expose their state to the MANO. The level of difficulty to ensure service continuity depends on whether a VNF is stateless or not. For stateless VNFs, such as DNS, it is relatively easy to prevent service discontinuity, as there is usually no state information to preserve both during maintenance and for failure recovery. We note that in order to guarantee service continuity for stateless VNF, we still need to be able to detect a failure and recover from it rapidly.

Most VNFs have networking service related state, including entries in the routing information base (RIB) and forwarding information base (FIB), the mapping between IP addresses in an NAT, the mapping between incoming flows and next-hop instances for a load balancer, flow information in a firewall, and so on. Stateful VNFs should maintain the state internally or externally or use a combination of both. During the normal operational phase, these VNFs need to synchronize the state with either backups or the management systems. They should preserve the state when a failure occurs in order to shield consumers from failures and provide the correctness of rollback recovery. For example, after spinning up a new VNF instance upon a failure, this instance should restore all of the state needed and handle existing traffic with limited or no disruption. For planned maintenance, when upgrading the software of a VNF, we should migrate the state to a backup instance during traffic redirection. It may take time and is sometimes difficult to achieve flawless migration in practice, as the state is usually tightly coupled with data processing [7].

Awareness of Infrastructure Resiliency

In order to gracefully handle unexpected failures, the design of VNFs should be aware of various redundancy schemes provided at different levels. There are multiple dimensions when providing redundancy, active-active vs. active-standby (VNF level), local vs. geographic (infrastructure level), and so on. For mission-critical VNFs and VNFs serving high-value customers, active-active

Although redundancy is required mainly for unexpected failures, it may also help planned maintenance. For example, during the upgrade of VNF software, we can first conduct it on the standby instance, then switch the roles between active and standby, and finally upgrade the software of the old active instance.

mode should be considered such that when one instance fails, the other one can immediately take over the customer traffic. To optimize resource utilization for active-standby, spare resources could be shared among standby instances that are activated only to handle failures, which is hard to achieve for physical network functions.

The risk of failure caused by an OpenStack instance may be mitigated by deploying another OpenStack instance in the same cloud data center, and the risk of failure caused by an entire cloud may be mitigated by deploying a third OpenStack instance at a remote cloud. VNFs should also be conscious of the latency of infrastructure-level redundancy schemes. Local redundancy is required to ensure low latency when switching over to a redundant instance that should have *already* been instantiated on site (e.g., in the secondary OpenStack instance of the same cloud). Geographic redundancy is required to prevent the impact of an entire cloud failure by placing redundant VNF instances in a selected remote cloud. VNFs with geographic redundancy should be able to tolerate a slightly longer failover time.

The design and implementation of a VNF should be cloud-aware to achieve five 9s availability. Typical cloud (e.g., OpenStack) supports application programming interfaces (APIs) and resiliency building blocks to overcome single points of failure such as server failure, full rack failure, single cloud instance failure, or complete data center failure. VNFs can overcome single server failures by leveraging OpenStack's anti-affinity rule and placing VMs on multiple servers. OpenStack's availability zone can also be used to place VMs on different racks to overcome rack failures. VNFs deployed on two or more instances of cloud data center are not likely going to be impacted by a single cloud instance failure. The true five 9s availability can only be achieved by placing VMs on multiple geo-locations (minimum of three data centers), which requires an overarching orchestration framework [2].

Although redundancy is required mainly for unexpected failures, it may also help planned maintenance. For example, during the upgrade of VNF software, we can first conduct it on the standby instance, then switch the roles between active and standby, and finally upgrade the software of the old active instance.

Awareness of Correlated Failures

As the resiliency demands of VNFs, NFVI, and MANO are interrelated, VNF designers should take the correlated failures of NFVI and MANO into consideration. Although VNFs can leverage redundancy to enhance their resiliency, there may be hidden and deep dependencies among these seemingly independent and redundant components/instances/systems, including both hardware and software dependencies. For example, two VNFs in a redundancy group may be running on the same server due to an improper placement policy of OpenStack. As a result, a failure of the server will undermine a VNF's redundancy efforts. Different types of VNFs may also share the same software components/libraries. Thus, a bug/defect in this common part will lead to concurrent failures of these VNFs.

Correlated failures have been extensively investigated in transport networks and cloud services, for example, discovering automatically shared risk link groups [8] and determining the inter-dependencies of cloud infrastructure that are hidden due to proprietary business relationships [9]. VNFs could leverage existing technologies/solutions developed for transport networks, core networks, and cloud services to handle these correlated failures. For example, they should consider catastrophic events and inevitable accidents (e.g., earthquakes and tsunamis), which may all overwhelm redundancy schemes with large-scale correlated failures caused by these disasters. The deployment of VNF instances and the design of service chaining should minimize the impact of correlated failures by exploring the dependency information from MANO and considering the topology of the infrastructure.

Existing Solutions

In this section, we review existing solutions that we can leverage to improve the resiliency of VNFs, including state management, VNF migration, and rollback recovery. We also discuss an advanced approach based on VNF decomposition.

State Management and Synchronization

In general, there are three types of state for most stateful VNFs, control state (e.g., routing entries and firewall rules), per-flow state (e.g., TCP status and the number of packets per flow), and aggregate state (e.g., state machine of an intrusion detection system). A VNF could choose to separate its state information from the corresponding VNF instance and store it in a central location (e.g., a database). A challenge here is the synchronization of the state between the running instances and the centralized state manager.

VNFs usually have two broad classes of state: internal and external. Internal state, similar to application logic, is only used and stored by a given VNF instance. It is unique to a running VNF instance. External state can be viewed as a large distributed data structure that is shared and managed across all replicas. Based on this classification, VNFs can build a split/merge-aware state management framework to facilitate the synchronization among replicas [10], or a control plane architecture that can coordinate quick and fine-grained reallocation of flows across VNF instances during failures while maintaining the consistent VNF state [11], or a centralized data store layer to decouple the state of VNFs from the packet processing component [7].

Make-before-Break VNF Migration

When either the underlying cloud infrastructure or the orchestrator detects a situation that may lead to a failure for a running VNF instance, it may initiate a remediation procedure, for example, by proactively migrating the VNF instance to another server of the same OpenStack instance. We note that if a VNF is completely down, it is too late for migration. Migration is helpful mainly for failure prevention. An efficient VNF migration scheme should minimize the traffic disruption on the data plane.

A possible solution is the so-called *make-before-break* migration that makes the destination VNF instance run at the same time as the original instance for a short period of time. It then shuts down the source when the destination instance can handle both the control plane and data plane correctly and independently. Such techniques have been used in

the seamless migration of virtual routers [12]. For example, after migrating the control plane to a new server, a virtual router can clone the original data plane on it by repopulating the FIB and then have two data planes running on both servers.

Rollback Recovery

Rollback is an operation to restore a VNF to its pre-failure state in order to recover from a failure. Checkpointing is the most widely used mechanism for rollback recovery, which periodically takes snapshots of a running VNF instance. Checkpointing does not require any modifications to a VNF. Backup instances can be activated immediately by restoring from the most recent snapshot. However, checkpointing alone cannot guarantee the correctness of recovery simply because the state changes between the most recent snapshot and the failure will be lost.

Checkpointing with buffering [13] ensures the correct recovery by holding outgoing packets in a buffer until a checkpoint has been generated. However, when there is no failure, the maximum delay added to a packet could be as high as the checkpoint interval, which is on the order of tens of milliseconds. Checkpointing with replay [4] first loads the most recent snapshot and then restores the internal state changes since the last snapshot by re-processing all duplicated packets stored in the input logger. The logs constitute a record of every non-deterministic event since the last snapshot. Although checkpointing with replay can reduce the latency of failure-free operations, it requires code modifications to a VNF.

Live VM Migration with Pass-through Devices

There are several challenges for migrating VMs with pass-through single-root input/output virtualization (SR-IOV) devices. One reason is that it is difficult to migrate the internal hardware state of the network interface cards (NICs), as it is directly managed by the virtual function (VF) driver, and the hypervisor cannot simply re-program it as it does with the software state. Moreover, the hypervisor cannot easily track the dirty memory inside the VM, which is modified by the pass-through devices when receiving packets. Thus, the received data during live migration may be lost.

Existing approaches for the live migration of VMs with pass-through devices can be divided into the following three categories [14]. The first solution uses NIC bonding (e.g., Linux Ethernet bonding driver) to bond a primary VF with a secondary virtual network interface that can support live migration. It leverages the secondary interface to forward traffic during the live migration. The second scheme implements an extra layer between the guest operating system (OS) kernel and the VF driver to emulate the hardware state, which cannot be migrated. It tracks the dirty memory by explicitly writing dummy data into the page that has the received data. The third approach enhances the hypervisor by leveraging the physical function (PF) driver of SR-IOV devices to inspect the VF state and track the memory of received packets.

VNF Decomposition

Another solution is the decomposition of a VNF. A straightforward dimension to decompose a VNF is by splitting its control plane and data plane. A VNF can also be decomposed into fine-grained software modules performing different packet processing functions, such as header lookup, flow reconstruction, and decompression [15] After decomposing a VNF into multiple components, the next step is to distribute them either within a cloud or even in more than one site such that failures in a single site do not impact the data packet processing of an end-to-end service. For example, when the server hosting the control plane fails, the data plane on another server can still forward data traffic for customers before the control plane recovers from the failure. In a later section, we illustrate such a solution in detail.

Research Challenges

In this section, we highlight several research challenges and point out a few future directions for VNF resiliency.

Offering resiliency is challenging for layer 3 VNFs due to several key differences between them and application/service VNFs that run at layer 4 or above. First, network VNFs usually interact directly and tightly with the physical network. Some VNFs may need to rely on the network fabric for a subset of their functions. For example, virtual routers may need to leverage the equal-cost multi-path (ECMP) forwarding function of a hardware switch for load balancing among multiple forwarders. Second, various technologies used for application VNFs may not be applicable to network VNFs, such as Domain Name Service (DNS)-based load balancing and anycast routing based failover. For instance, it may not be feasible to use a layer 4 load balancer for network VNFs, as the load balancer itself requires layer 3 support provided by network VNFs. Third, different from application VNFs, which can route packets among their instances using layer 3 devices, network VNFs may need to use various tunneling technologies to forward packets to the next hop.

There are other challenges for VNF resiliency in general. First of all, it is difficult to satisfy most of the requirements summarized earlier, thus requiring further investigation from the research community. In addition, it would be desirable to have a unified API from different VNF vendors for more effective management (similar to OpenFlow for switches), especially to export and import VNF state. There is a trade-off between resiliency, performance, and resource utilization, as offering high availability and reliability may affect the system performance (e.g., lowering throughput or increasing latency) and the overall resource utilization (e.g., due to redundancy). So far, we have discussed the resiliency of individual VNFs. Another challenge is related to improving the resiliency of multiple VNFs when they are chained together to offer networking services (e.g., the fault-tolerant placement of VNFs in a service chain). Finally, instead of migrating VMs among servers, which could be limited by whether a VM uses SR-IOV and other factors, a more efficient and lightweight solution would be to migrate VNFs among VMs.

Case Study: EdgePlex

As a case study, in this section we demonstrate how to enhance the resiliency by decomposing the virtual provider edge (PE) routers. The main functionality of a PE router is to connect customers to a service provider's network. Following the SDN and NFV principles, we have proposed a new architecture, called EdgePlex [6], to improve the reliability and flexibility of PE routers.

There are several challenges for migrating VMs with pass-through SR-IOV devices. One reason is that it is difficult to migrate the internal hardware state of the NICs, as it is directly managed by the VF (Virtual Function) driver and the hypervisor cannot simply re-program it as it does with the software state.

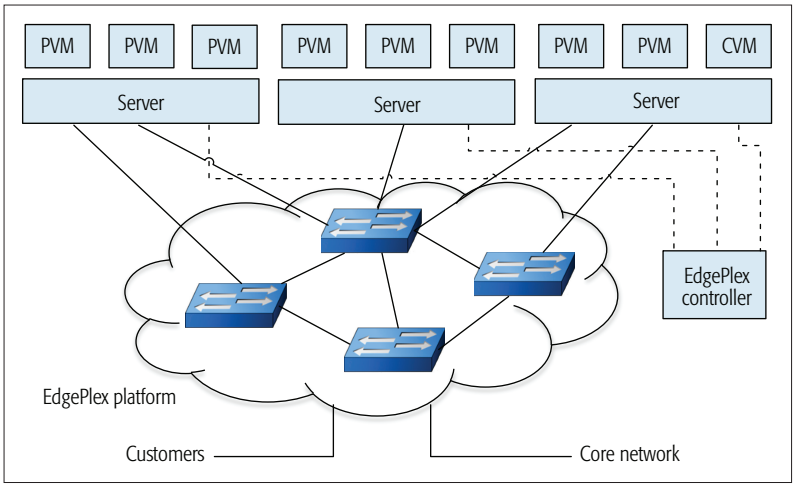

Figure 2. Architecture of the EdgePlex platform.

EdgePlex -- Architecture and Design

We present the EdgePlex architecture in Fig. 2, which is built on top of commodity servers and switches. It also leverages the recent advances in virtualization.

A key novel aspect of EdgePlex is that we use a sandboxed environment (e.g., VMs) to isolate customers. It offers the flexibility to independently move per-customer VMs within or across the platform for either planned maintenance or failure recovery. We assign each customer a VM, called a PortVM (or PVM), which is similar to a physical port on traditional physical routers. EdgePlex stores customer-specific configuration, such as routing and access control, and control and data plane state (e.g., routing and forwarding tables) in PVMs. As a result, we terminate a customer's routing session (e.g., BGP) on its PVM. This design provides network operators with the ability to migrate a customer with limited impact to others.

To connect to the core network, a simple solution is to allow each PVM to directly communicate with the core network, behaving as a virtual router with only one configured customer. However, this approach has scalability limitations, as each PVM needs to have a different multiprotocol label switching (MPLS) label and at least one Border Gateway Protocol (BGP) session to the core network. In order to address this issue, we introduce another VM, called a ControlVM (or CVM), which represents the PE router (from the control plane perspective) to the core network. The CVM speaks BGP with the core routers and relays the control plane information to and from the PVMs.

We have also designed a controller for EdgePlex to manage its components. It instantiates the per-customer PVMs and configures their routing protocols and connectivity in the hypervisors and switches. It also monitors the platform and takes action when needed (e.g., reacts to failures).

Resiliency of EdgePlex

We use existing VM redundancy technologies (i.e., Micro-Checkpointing in KVM) to protect the CVM from unexpected failures, as it is on the control plane, and its failure will make the entire router not available. Micro-Checkpointing takes periodic snapshots of the running VM and stores them on a separate machine. Upon a failure, the backup VM will quickly detect a loss of network connectivity and immediately load and activate the most recent snapshot. By using Micro-Checkpointing, EdgePlex can recover from a CVM failure without affecting its running protocols, because the failover to the backup CVM is usually faster than the timeout of control plane protocols. The backup CVM is activated only when the primary CVM fails, and thus there is no IP address conflict. To leverage the cloud resiliency feature, the backup CVM should be placed, for example, on a different availability zone of the same OpenStack instance.

Due to the number of PVMs and their limited failure impact, we can choose not to apply Micro-Checkpointing to them. If a PVM fails, it will affect only the customer configured on it, and we can re-instantiate a PVM to restore connectivity or redirect traffic to other PVMs. For the planned maintenance of PVMs, we can utilize a make-before-break live migration to reduce the connectivity interruption by modifying the source code of KVM (including libvirt and QEMU). As mentioned earlier, the key idea is to shut down the original source PVM *after* the destination PVM has been running. Because of the identical IP addresses on both PVMs, we use an OpenFlow switch to forward packets to only one of them by configuring the rules accordingly.

For failures of other components, a server failure is isolated to only customer PVMs on that server, and we can re-instantiate these PVMs rapidly. We are also investigating lightweight protection mechanisms for PVMs. The impact of a switch failure depends on how the servers are interconnected using the switches, which can be handled by redundant connections among them.

Performance Evaluation

We have implemented a prototype of EdgePlex, using a combination of custom and open source software. We now evaluate the resiliency features of EdgePlex using this prototype implementation in a testbed, as shown in Fig. 3. It has three sites, locations A, B, and C, with each site hosting an EdgePlex PE. We use the Ryu controller to configure the switches through OpenFlow. We run two groups of experiments. For the checkpointing experiments, the CVM is running on the left server of Location A and periodically sends checkpoints to the right server. For the migration experiments, we migrate the PVM of Client1 from the left server to the right one of location A. We refer interested readers to the EdgePlex publication [6] for more experimental results.

The checkpoint frequency is a key parameter of Micro-Checkpointing. On one hand, if we create checkpoints frequently, the communication overhead may be high, as Micro-Checkpointing needs to periodically transfer dirty memory to the backup VM. On the other hand, if we do not generate enough checkpoints, we may lose the changed network state (e.g., route updates) between backups. We have measured the number of transferred bytes for various checkpoint frequencies under the condition that the CVM receives 200 BGP updates every second. We summarize the results in Table 1. As we can see, the total number of bytes transferred per second decreases as we increase the checkpointing interval. Since we use this approach for only the CVM, the amount of transferred data is still manageable even in the worst case (74.7 MB for the 100 ms interval).

We have evaluated the impact of PVM live migration on the data plane using the make-before-break approach mentioned above. For the existing KVM live migration, the connectivity interruption mainly comes from the modification of switch rules, which can be done only after the migration completes. With the make-before-break PVM migration, we can modify these rules when both the source and the destination VMs are running and thus reduce the interruption to customer traffic. Suppose p is the last packet seen on the left server before the migration and q is the first packet on the right server after the migration. Their timestamps on the receiver side (i.e., either the video server or client) are T_p and T_q, respectively. We estimate the duration of migration to be $T_M = T_q - T_p$. The make-before-break approach can reduce T_M from longer than a second to be less than 100 ms. During the PVM migration, both the video traffic for Location B and the background traffic between Locations A and C were not affected.

Conclusion

When adopting NFV and moving network functions from dedicated appliances to error-prone commodity hardware, network operators should guarantee that the reliability and availability of the offered network services are not affected. In this article, we have summarized the resiliency requirements for VNFs, such as the service continuity and automated failure recovery, and reviewed several existing solutions in this area. As a case study, we have also demonstrated how to improve the resiliency feature of virtual PE routers through VNF decomposition, which offers network operators significantly more flexibility to manage their services. In our future work, we plan to apply the summarized principles to other types of VNFs, such as virtual load balancer and firewall, and further investigate the resiliency of NFVI and MANO (e.g., rerouting traffic around failures).

Acknowledgment

We thank the anonymous reviewers for their insightful comments. We thank Jennifer Yates and Oliver Spatscheck for discussions and suggestions.

References

[1] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network Function Virtualization: Challenges and Opportunities for Innovations," *IEEE Commun. Mag.*, vol. 53, no. 2, Feb. 2015, pp. 90–97.

[2] AT&T Inc., "ECOMP (Enhanced Control, Orchestration, Management & Policy) Architecture," white paper, Mar. 2016.

[3] J. P. Sterbenz *et al.*, "Resilience and Survivability in Communication Networks: Strategies, Principles, and Survey of Disciplines," *Computer Networks*, vol. 54, June 2010, pp. 1245–65.

[4] J. Sherry *et al.*, "Rollback-Recovery for Middleboxes," *SIGCOMM*, 2015.

[5] R. Govindan *et al.*, "Evolve or Die: High-Availability Design Principles Drawn from Google's Network Infrastructure," *SIGCOMM*, 2016.

[6] A. Chiu *et al.*, "EdgePlex: Decomposing the Provider Edge for Flexibilty and Reliability," *SOSR*, 2015.

[7] M. Kablan *et al.*, "Stateless Network Functions: Breaking the Tight Coupling of State and Processing," *NSDI*, 2017.

[8] P. Sebos *et al.*, "Auto Discovery of Shared Risk Link Group," *OFC*, 2001.

[9] E. Zhai *et al.*, "Heading Off Correlated Failures through Independence-as-a-Service," *OSDI*, 2014.

[10] S. Rajagopalan *et al.*, "Split/Merge: System Support for Elastic Execution in Virtual Middleboxes," *NSDI*, 2013.

[11] A. Gember-Jacobson *et al.*, "OpenNF: Enabling Innovation in Network Function Control," *SIGCOMM*, 2014.

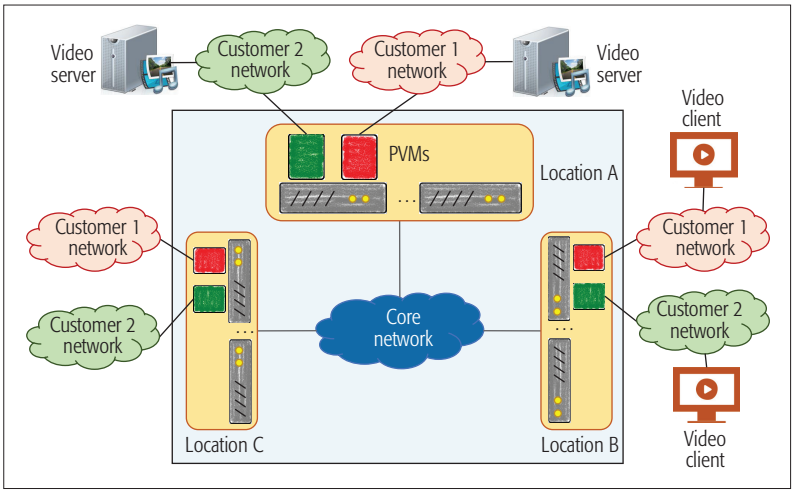

Figure 3. Testbed setup of a prototype implementation of the EdgePlex platform. The two customers are colored in green and red.

Checkpoint interval (ms)	100	200	500	1000
Data transferred (MB)	74.7	39.6	33.2	36.6

Table 1. The number of bytes transferred when checkpointing the CVM for different intervals.

[12] Y. Wang *et al.*, "Virtual Routers on the Move: Live Router Migration as a Network-Management Primitive," *SIGCOMM*, 2008.

[13] B. Cully *et al.*, "Remus: High Availability via Asynchronous Virtual Machine Replication," *NSDI*, 2008.

[14] X. Xu and B. Davda, "SRVM: Hypervisor Support for Live Migration with Passthrough SR-IOV Network Devices," *iVEE*, 2016.

[15] A. Bremler-Barr, Y. Harchol, and D. Hay, "OpenBox: A Software-Defined Framework for Developing, Deploying, and Managing Network Functions," *SIGCOMM*, 2016.

Biographies

Bo Han (bohan@research.att.com) is a senior inventive scientist at AT&T Labs – Research. He received his Bachelor's degree in computer science and technology from Tsinghua University in 2000, his M.Phil. degree in computer science from City University of Hong Kong in 2006, and his Ph.D. degree in computer science from the University of Maryland in 2012. His research interests are in the areas of network functions virtualization, software defined networking, mobile computing, and wireless networking, with a focus on developing simple but efficient and elegant solutions for real-world networking and systems problems.

Vijay Gopalakrishnan [M] is a director in the Network Research Department of AT&T Labs – Research, leading a team focused on systems challenges in the architecture, protocols, and management of networks. His research interests lie broadly in the area of networked systems, where he has worked on topics of network management, content delivery, and the mobile web. Prior to joining AT&T, he got his Master's and Ph.D. in computer science from the University of Maryland, College Park in 2003 and 2006, respectively. Vijay is a member of ACM.

Gnanavelkandan Kathirvel is a lead system architect for AT&T and on the Board of Directors for OpenStack. He leads technology efforts around NFV and AT&T's Integrated Cloud Platform. He is currently responsible for shepherding SDN and NFV projects on to AT&T's Integrated Cloud platform and a big open source advocate. Previously, he led the architecture work to support cloud convergence, building external cloud and a content delivery network for AT&T.

Aman Shaikh is a principal inventive scientist at AT&T Labs – Research. He obtained his Ph.D. and M.S. in computer engineering from the University of California, Santa Cruz in 2003 and 2000, respectively. His current research interests include service quality management, SDN, and NFV. Several tools that have emerged from his research are being used extensively by AT&T operations teams. He has also published results of his research in prestigious conferences and journals.