ETSI GS NFV-REL 002 V1.1.1 (2015-09)

Network Functions Virtualisation (NFV);
Reliability;
Report on Scalable Architectures for Reliability Management

Disclaimer

This document has been produced and approved by the Network Functions Virtualisation (NFV) ETSI Industry Specification Group (ISG) and represents the views of those members who participated in this ISG.
It does not necessarily represent the views of the entire ETSI membership.

Reference

DGS/NFV-REL002

Keywords

architecture, NFV, reliability

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Important notice

The present document can be downloaded from:
http://www.etsi.org/standards-search

The present document may be made available in electronic versions and/or in print. The content of any electronic and/or print versions of the present document shall not be modified without the prior written authorization of ETSI. In case of any existing or perceived difference in contents between such versions and/or in print, the only prevailing document is the print of the Portable Document Format (PDF) version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at
http://portal.etsi.org/tb/status/status.asp

If you find errors in the present document, please send your comment to one of the following services:
https://portal.etsi.org/People/CommiteeSupportStaff.aspx

Copyright Notification

No part may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm except as authorized by written permission of ETSI.
The content of the PDF version shall not be modified without the written authorization of ETSI.
The copyright and the foregoing restriction extend to reproduction in all media.

© European Telecommunications Standards Institute 2015.
All rights reserved.

DECT™, **PLUGTESTS**™, **UMTS**™ and the ETSI logo are Trade Marks of ETSI registered for the benefit of its Members.
3GPP™ and **LTE**™ are Trade Marks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners.
GSM® and the GSM logo are Trade Marks registered and owned by the GSM Association.

Contents

Intellectual Property Rights 4
Foreword....4
Modal verbs terminology....4
1 Scope 5
2 References 5
2.1 Normative references 5
2.2 Informative references....6
3 Definitions and abbreviations....7
3.1 Definitions....7
3.2 Abbreviations 7
4 Scalable Architecture and NFV....8
4.1 Introduction 8
4.2 Overview of Current Adoption in Cloud Data Centres 9
4.3 Applicability to NFV....9
5 Scaling State....10
5.1 Context 10
5.2 Categories of Dynamic State 13
5.3 Challenges 14
6 Methods for Achieving High Availability....15
6.1 High Availability Scenarios 15
6.2 Dynamic Scaling with Migration Avoidance 16
6.3 Lightweight Rollback Recovery....20
6.3.1 Overview 20
6.3.2 Checkpointing....21
6.3.3 Checkpointing with Buffering 22
6.3.4 Checkpointing with Replay 23
6.3.5 Summary Trade-offs of Rollback Approaches 24
7 Recommendations 24
7.1 Conclusion....24
7.2 Guidelines for Scalable Architecture Components....24
7.3 Future Work 25
Annex A (informative): Experimental Results....26
A.1 Migration Avoidance Results....26
A.2 Lightweight Rollback Recovery Results....27
A.2.1 Introduction 27
A.2.2 Latency....28
A.2.3 Throughput....29
A.2.4 Replay Time 29
A.2.5 Conclusion....30
Annex B (informative): Authors & contributors....31
History 32

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://ipr.etsi.org).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This Group Specification (GS) has been produced by ETSI Industry Specification Group (ISG) Network Functions Virtualisation (NFV).

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The present document describes a study of how today's Cloud/Data Centre techniques can be adapted to achieve scalability, efficiency, and reliability in NFV environments. These techniques are designed for managing shared processing state with low-latency and high-availability requirements. They are shown to be application-independent that can be applied generally, rather than have each VNF use its own idiosyncratic method for meeting these goals. Although an individual VNF could manage its own scale and replication, the techniques described here require a single coherent manager, such as an orchestrator, to manage the scale and capacity of many disparate VNFs. Today's IT/Cloud Data Centres exhibit very high availability levels by limiting the amount of unique state in a single element and creating a virtual network function from a number of small replicated components whose functional capacity can be scaled in and out by adjusting the running number of components. Reliability and availability for these type of VNFs is provided by a number of small replicated components. When an individual component fails, little state is lost and the overall VNF experiences minimal change in functional capacity. Capacity failures can be recovered by instantiating additional components. The present document considers a variety of use cases, involving differing levels of shared state and different reliability requirements; each case is explored for application-independent ways to manage state, react to failures, and respond to increased load. The intent of the present document is to demonstrate the feasibility of these techniques for achieving high availability for VNFs and provide guidance on Best Practices for scale out system architectures for the management of reliability. As such, the architectures described in the present document are strictly illustrative in nature.

Accordingly, the scope of the present document is stated as follows:

- Provide an overview of how such architectures are currently deployed in Cloud/Data Centres.
- Describe various categories of state and how scaling state can be managed.
- Describe scale-out techniques for instantiating new VNFs in a single location where failures have occurred or unexpected traffic surges have been experienced. Scale-out may be done over multiple servers within a location or in a server in the same rack or cluster within any given location. Scaling out over servers in multiple locations can be investigated in follow-up studies.
- Develop guidelines for monitoring state such that suitable requirements for controlling elements (e.g. orchestrator) can be formalized in follow-up studies.

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at http://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

Not applicable.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] R. Strom and S. Yemini: "Optimistic Recovery in Distributed Systems", ACM Transactions on Computer Systems, 3(3):204-226, August 1985.

[i.2] Sangjin Han, Keon Jang, Dongsu Han and Sylvia Ratnasamy: "A Software NIC to Augment Hardware", in Submission to 25th ACM Symposium on Operating Systems Principles (2015).

[i.3] E.N. Elnozahy, Lorenzo Alvisi, Yi-Min Wang, David Johnson: "A Survey of Rollback-Recovery Protocols in Message-Passing Systems", ACM Computing Surveys, Vol. 34, Issue 3, September 2002, pages 375-408.

[i.4] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson and A. Warfield: "Remus: High Availability via Asynchronous Virtual Machine Replication". In Proceedings USENIX NSDI, 2008.

[i.5] Kemari Project.

NOTE: Available at http://www.osrg.net/kemari/.

[i.6] J. Sherry, P. Gao, S. Basu, A. Panda, A. Krishnamurthy, C. Macciocco, M. Manesh, J. Martins, S. Ratnasamy, L. Rizzo and S. Shenker: "Rollback Recovery for Middleboxes", Proceedings of the ACM, SIGCOMM, 2015.

[i.7] ETSI NFV Reliability Working Group Work Item DGS/NFV-REL004 (V0.0.5), June 2015: "Report on active Monitoring and Failure Detection in NFV Environments".

[i.8] OPNFV Wiki: "Project: Fault Management (Doctor)".

NOTE: Available at https://wiki.opnfv.org/doctor.

[i.9] E. Kohler et al.: "Click Modular Router", ACM Transactions on Computer Systems, August 2000.

[i.10] "Riverbed Completes Acquisition of Mazu Networks".

NOTE: Available at: http://www.riverbed.com/about/news-articles/press-releases/riverbed-completes-acquisition-of-mazu-networks.html.

[i.11] Digital Corpora: "2009-M57-Patents packet trace".

[i.12] S. Rajagopalan et al.: "Pico Replication: A High Availability Framework for Middleboxes", Proceedings of ACM SoCC, 2013.

[i.13] Remus PV domU Requirements.

NOTE: Available at http://wiki.xen.org/wiki/Remus_PV_domU_requirements.

[i.14] B. Cully et al.: "Remus: High Availability via Asynchronous Virtual Machine Replication", Proceedings USENIX NSDI, 2008.

[i.15] Lee D. and Brownlee N.: "Passive Measurement of One-way and Two-way Flow Lifetimes", ACM SIGCOMM Computer Communications Review 37, 3 (November 2007).

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

affinity: for the purposes of the present document, property whereby a flow is always directed to the VNF instance that maintains the state needed to process that flow

checkpoint: snapshot consisting of all state belonging to a VNF; required to make an identical "copy" of the running VNF on another system

NOTE: One way to generate a checkpoint is by using memory snapshotting built in to the hypervisor.

core: independent processing unit within a CPU which executes program instructions

correct recovery: A system recovers correctly if its internal state after a failure is consistent with the observable behaviour of the system before the failure.

NOTE: See [i.1] for further details.

flow: sequence of packets that share the same 5-tuple: source port and IP address, destination port and IP address, and protocol

non-determinism: A program is non-deterministic if two executions of the same code over the same inputs may generate different outputs.

NOTE: Programs which when given the same input are always guaranteed to produce the same output are called deterministic.

stable storage: memory, SSD, or disk storage whose failure conditions are independent of the failure condition of the VNF; stable storage should provide the guarantee that even if the VNF fails, the stable storage will remain available

state: contents of all memory required to execute the VNF, e.g. counters, timers, tables, protocol state machines

thread: concurrent unit of execution, e.g. p-threads or process.h threads

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

CDF Cumulative Distribution Function
CPU Central Processing Unit
DDoS Distributed Denial of Service
DHCP Dynamic Host Configuration Protocol
DPDK Data Plane Development Kit
DPI Deep Packet Inspection
FTMB Fault Tolerant MiddleBox
Gbps Giga bits per second
HA High Availability
IDS Intrusion Detection System
IP Internet Protocol
Kpps Kilo packets per second
Mpps Mega packets per second
NAT Network Address Translation
NFV Network Function Virtualisation
NFVI Network Function Virtualisation Infrastructure
NIC Network Interface Controller
NUMA Non Uniform Memory Access
QoS Quality of Service
TCP Transmission Control Protocol
VF Virtual Function
VM Virtual Machine

VNF	Virtualised Network Function
VPN	Virtual Private Network
WAN	Wide Area Network

4 Scalable Architecture and NFV

4.1 Introduction

Traditional reliability management in telecommunications networks typically depends on a variety of redundancy schemes. For example, spare resources may be designated in some form of standby mode; these resources are activated in the event of network failures such that service outages are minimized. Alternately, over-provisioning of resources may also be considered (active-active mode) such that if one resource fails, the remaining resources can still process traffic loads.

The advent of Network Functions Virtualisation (NFV) ushered in an environment where the focus of telecommunications network operations shifted from specialized and sophisticated hardware with potentially proprietary software functions residing on them towards commoditized and commercially available servers and standardized software that can be loaded up on them on an as needed basis. In such an environment, Service Providers can enable dynamic loading of Virtual Network Functions (VNF) to readily available servers as and when needed - this is referred to as "scaling out" (see note). Traffic loads can vary with bursts and spikes of traffic due to external events; alternately network resource failures may reduce the available resources to process existing load adequately. The management of high availability then becomes equivalent to managing dynamic traffic loads on the network by scaling out VNFs where needed and when necessary. This is the current method of managing high availability in Cloud/Data Centres. The goal of the present document is to describe how such scalable architecture methods can be adapted for use in NFV-based Service Provider networks in order to achieve high availability for telecommunications services.

NOTE: It is also possible to reduce the number of existing VNFs if specific traffic types have lower than expected loads; this process is known as "scaling in".

The use of scalable architecture involves the following:

- Distributed functionality with sufficient hardware (servers and storage) resources deployed in multiple locations in a Service Provider's region.
- Duplicated functionality within locations and in multiple locations such that failure in one location does not impact processing of services.
- Load balancing such that any given network location does not experience heavier loads than others.
- Managing network scale and monitoring network state such that the ability of available resources to process current loads is constantly determined. In the event of failures, additional VNFs can be dynamically "scaled-out" to appropriate locations/servers such that high availability is maintained.

The following assumptions are stated for the development of the present document:

- Required hardware (servers and storage) is pre-provisioned in sufficient quantities in all Service Provider locations such that scaling-out new VNFs is always possible at any given location when necessary.
- Required hardware is distributed strategically over multiple locations throughout the Service Provider's network.
- The relationship between the type of service and the corresponding VNFs necessary to process the service type is expected to be known.

4.2 Overview of Current Adoption in Cloud Data Centres

Typical services offered by Cloud providers include web based services and cloud computing. Scalable architectures for managing availability in response to load demands have been successfully implemented by Cloud Service providers. A high level overview of the techniques for achieving high availability is as follows:

- Sizing Functional Components: Cloud providers now craft smaller components in terms of functionality and then deploy very large numbers of such components in Data Centres. Sizing such components is thus important - how much functional software can be loaded onto commercial hardware products. Each hardware resource therefore handles fewer functions than the traditional hardware resources. If one or more such components fail, the impact on service delivery is not expected to be very significant.
- Distributed Functionality: Data Centres are located in multiple regions by the Cloud Service Provider. Failure in one Data Centre does not impact the performance of other Centres. Functionality is duplicated simply by deploying large numbers of functional components. The distributed nature of Cloud Data Centres thus permits storage of critical information (service and customer information) within one location and in multiple locations insulated from each other. Failure in one location thus permits the relevant information to be brought online through alternate Centres.
- Load Balancing: Incoming load can be processed though a load balancer which distributes load by some designated mechanism such that no Data Centre system experiences overload conditions. Given multiple locations and multiple storage of critical information, load balancing provides a method to ensure availability of resources even under failure conditions.
- Dynamic Scalability: Again, given the small size of functional components, it is fairly straightforward to scale-out (or scale-in) necessary resources in the event of failure or bursty load conditions.
- Managing Scale and State: Methods for keeping track of the state of a Cloud Service provider's resources is critical. These methods enable the provider to determine whether currently deployed resources are sufficient to ensure high availability or not. If additional resources are deemed necessary then they can be brought online dynamically.

4.3 Applicability to NFV

The main motivating factor for Service Providers for adopting NFV is the promise of converting highly specialized communication centre locations (e.g. Central Offices, Points of Presence) in today's networks into flexible Cloud-based Data Centres built with commercial hardware products that:

1) Continue the current function of communication centres, namely; connect residential and business customers to their networks.

2) Expand into new business opportunities by opening up their network infrastructures to third party services. An example of such a service is hosting services currently offered by Cloud Data Centres.

Embracing an NFV-based design for communication centres allows Service Providers to enable such flexibility. This also incentivizes Service Providers to explore Cloud/Data Centre methodologies for providing high availability to their customers.

Today's communication centres provide a wide range of network functions such as:

- Virtual Private Network (VPN) support
- Firewalls
- IPTV/Multicast
- Dynamic Host Configuration Protocol (DHCP)
- Quality of Service (QoS)
- Network Address Translation (NAT)
- Wide Area Network (WAN) Support

- Deep Packet Inspection (DPI)
- Content Caching
- Traffic Scrubbing for Distributed Denial of Service (DDoS) Prevention

These functions are well suited for an NFV environment. They can be supplemented with additional functions for delivery of Cloud services such as hosting. In principle, all such functions can be managed for high availability via Cloud-based Scalable Architecture techniques.

Traditional reliability management in telecommunications networks typically depends on a variety of redundancy schemes whereby spare resources are designated in some form of active-active mode or active-standby mode such that incoming traffic continues to be properly processed. The goal is to minimize service outages.

With the advent of NFV, alternate methods of reliability management can be considered due to the following:

- Commercial Hardware - Hardware resources are no longer expected to be specialized. Rather than have sophisticated and possibly proprietary hardware, NFV is expected to usher in an era of easily available and commoditized Commercial Off-the-Shelf products.
- Standardized Virtual Network Functions (VNF) - Software resources that form the heart of any network's operations are expected to become readily available from multiple sources. They are also expected to be deployed in multiple commercial hardware with relative ease.

In such an environment, it can be convenient to "scale-out" network resources - rapidly instantiate large numbers of readily available and standardized VNFs onto pre-configured commercial hardware/servers. This results in large numbers of server/VNF combinations each performing a relatively small set of network functions. This scenario is expected to handle varying traffic loads:

- Normal Loads - Typically expected traffic loads based on time-of-day and day-of-week.
- Traffic Bursts - Such situations can arise due to outside events or from network failures. Outside events (e.g. natural disasters, local events of extreme interest) can create large bursts of traffic above and beyond average values. Network failures reduce the available resources needed to process service traffic loads thereby creating higher load volumes for remaining resources.

Scaling out resources with NFV can be managed dynamically such that all types of network loads can be satisfactorily processed. This type of dynamic scale-out process in response to traffic load demands results in high availability of network resources for service delivery.

The present document provides an overview of some of these techniques to ensure high availability of these functions under conditions of network failures as well as unexpected surges in telecommunications traffic.

5 Scaling State

5.1 Context

This clause presents a high level overview of the context underlying the solution methods that are presented in clause 6. The focus here is on managing high availability of VNF services within a single location; this location may be a cluster deployed within a Service Provider's Central Office, a regional Data Centre, or even a set of racks in a general-purpose cloud. A Service Provider's network will span multiple such locations. The assumption is that there is a network-wide control architecture that is responsible for determining what subset of traffic is processed by which VNFs in each location. For example, the controlling mechanism might determine that Data Centre D1 will provide firewall, WAN optimization and Intrusion Detection services for traffic from customers C1, . . . , Ck. A discussion of this network-wide control architecture is beyond the scope of the present document.

It is critical to note that the focus of the present document is only on meeting the dictates of the network-wide controlling mechanism within a single location, in the face of failure and traffic fluctuations. Some high level descriptions of the architecture utilized for this study are as follows:

- Infrastructure View: It is understood that multiple architectures are possible for the solution infrastructure. The clause 6 solution techniques are based on a high level architecture that comprises a set of general-purpose servers interconnected with commodity switches within each location. The techniques for managing scale are presented in the context of a single rack-scale deployment (i.e. with servers interconnected by a single switch); the same techniques can be applied in multi-rack deployments as well. As shown in figure 1, a subset of the switch ports are "external" facing, while the remaining ports interconnect commodity servers on which VNF services are run. This architecture provides flexibility to balance computing resources and switching capacity based on operator needs. A traffic flow enters and exits this system on the external ports: an incoming flow may be directly switched between the input and output ports using only the hardware switch, or it may be "steered" through one or more VNFs running on one or more servers.

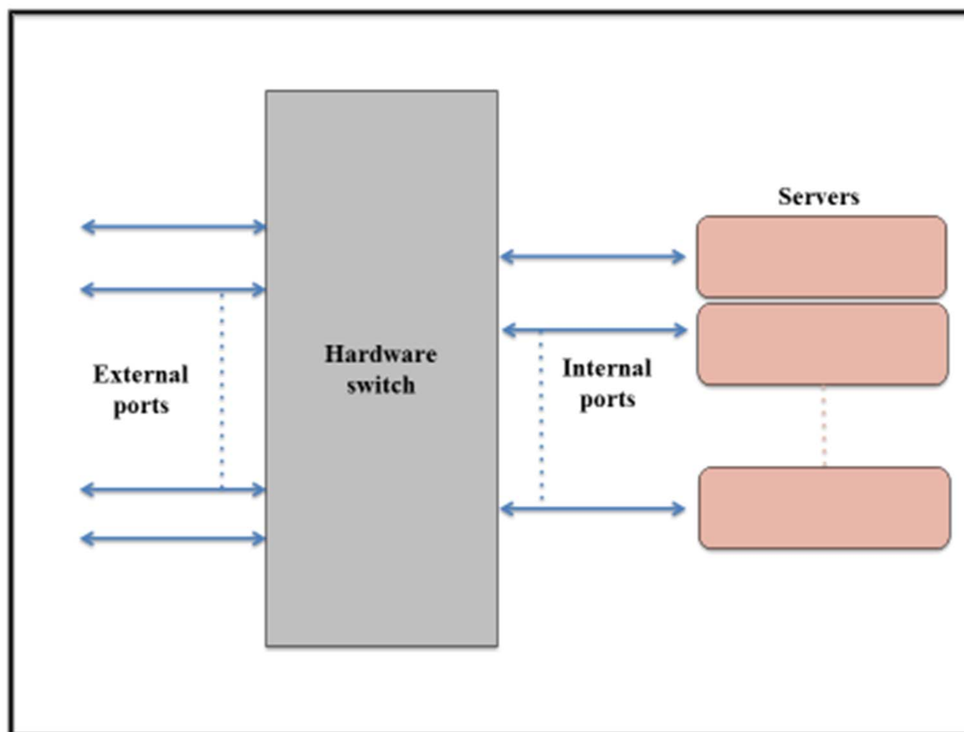

Figure 1: Hardware Infrastructure

- System View: The overall system architecture (within a single location) is illustrated in figure 2. This architecture comprises three components:
  - Logically centralized controlling mechanism (such as an orchestrator) that maintains a system-wide view.
  - Virtual Network Functions (VNFs) implemented as software applications
  - Software switching layer that underlies the VNFs - VNFs implement specific traffic processing services - e.g. firewalling, Intrusion Detection System (IDS), WAN optimization - while the software switching layer is responsible for correctly "steering" traffic between VNFs.

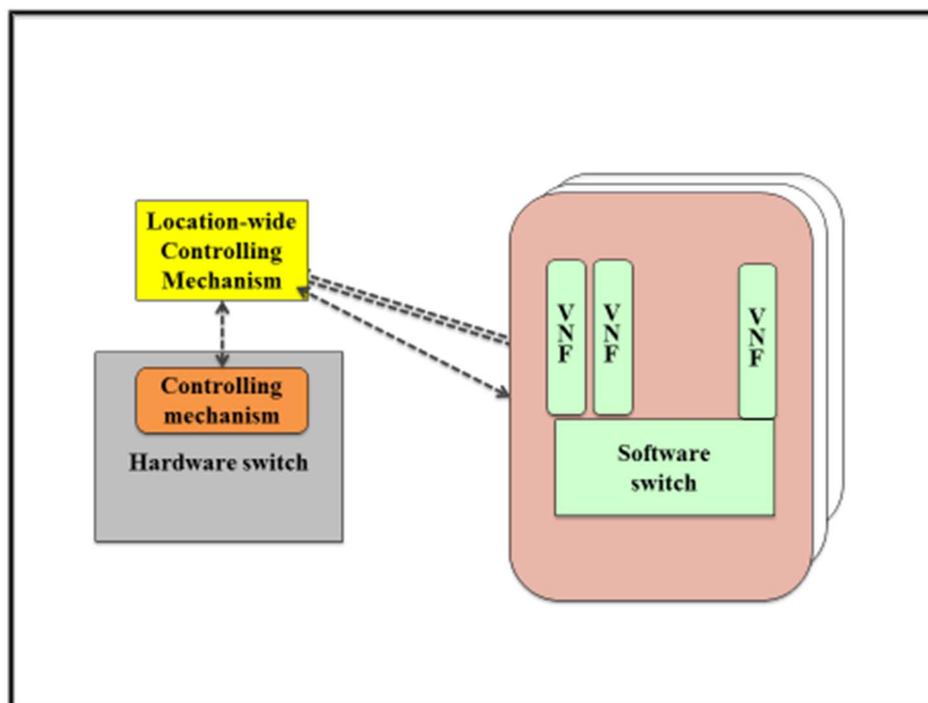

Figure 2: System View

- VNF Implementation View: VNFs are implemented as multi-threaded applications that run in parallel on a multicore CPU (see figure 3). It is assumed that 'multi-queue' Network Interface Controllers (NIC) are deployed offering multiple transmit and receive queues that are partitioned across threads. Each thread reads from its own receive queue(s) and writes to its own transmit queue(s). The NIC partitions packets across threads using the classification capabilities offered by modern NIC hardware - e.g. hashing a packet's 5-tuple including source and destination port and address to a queue; hence, all packets from a flow are processed by the same thread and each packet is processed entirely by one thread. The above are standard approaches to parallelizing traffic processing in multicore packet-processing systems.

NOTE: It is possible to implement VNFs as single-threaded applications. In such cases, they are equivalent to Per-flow State (see clause 5.2) and hence, recovery mechanisms for such applications fall into the "straightforward" type (see clause 6.1).

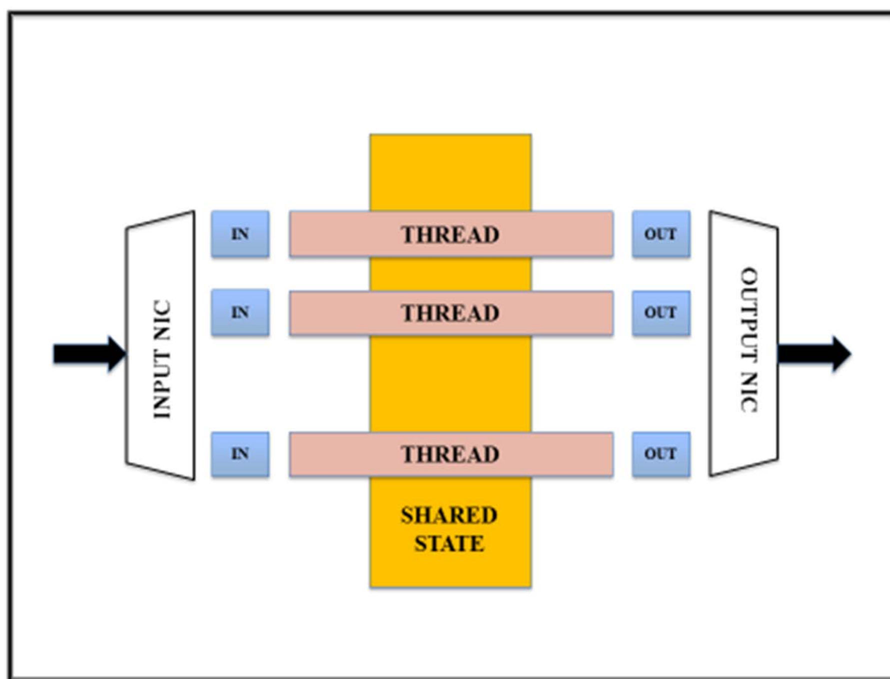

Figure 3: Multi-Threaded VNF

- Virtualisation View: VNF code is assumed to be running in a virtualised mode. The virtualisation need not be a VM per se; containers could be used or some other form of compartmentalization that provides isolation and supports low overhead snapshots of its content.

5.2 Categories of Dynamic State

Those VNFs dealing with stateful applications - e.g. Network Address Translators (NATs), WAN Optimizers, and Intrusion Prevention Systems all maintain dynamic state about flows, users, and network conditions. As discussed in clause 5.3, correctly managing this state is the key challenge in achieving high availability. While many forms of state are possible for general applications, the focus here is on three forms of state that are most common to traffic processing applications:

- Control State: State that is created (i.e. written) by a single control thread and consumed (i.e. read) by all other threads. The canonical example of such state would be data structures that store forwarding entries or access control lists. Note that the reading and writing thread(s) may run on different cores within a single server, or on different servers.

- Per-flow State: State that is created and consumed when processing packets that belong to a single flow. State maintained for byte stream reconstruction, connection tracking, or counters that track the number of bytes or packets per flow are examples of per-flow state.

- Aggregate State: State that is created and consumed when processing packets belong to an aggregate of flows. Examples of flow aggregates include all flows to/from an enterprise, all flows from a source prefix, or all flows to a destination address. Common forms of aggregate state include counters that track the number of connections initiated from (say) an IP prefix range, IDS state machines, rate limiters, packet caches for WAN optimizers, etc.

The three types of state described above can be shared across multiple threads; this is referred to as shared state. High Availability (HA) techniques for shared state face additional challenges (clause 5.3) since they have to consider the effects of coordination across multiple threads. For the model proposed in the present document for how VNFs partition traffic across threads, all packets from a flow are processed by a single thread, so per-flow state is local to a single thread and is not shared state. Control state is shared state, but this is a relatively easy case since there is only a single thread that writes to the state; all other threads only read the shared state. Aggregate state may be shared and, in this case, each thread can both read and write the shared state. While it is preferable that aggregate state be contained within a single server, this may not always be possible. In particular, if the total throughput demand of a flow aggregate cannot be handled by a single server, then any state associated with that aggregate has to be spread across multiple servers; multiple servers may be needed in any case for redundancy purposes. Hence it is necessary to further distinguish between aggregate state that is shared across multiple threads on a single server (aggregate, single-server) and aggregate state that is shared by threads on different servers (aggregate, multi-server).

5.3 Challenges

There are two aspects to achieving high availability for VNF services: scaling out/in the number of VNF instances in response to variations in traffic load, and recovering from the failure of a VNF instance.

- Challenges for Dynamic Scaling: Under overload condition, new VNFs are instantiated when the existing VNFs are unable to cope with the load. For the purposes of the present document, existing VNFs and existing traffic are referred to as "old"; new VNFs and newly arriving traffic are referred to as "new". VNFs exhibit two characteristics that make dynamic scaling of VNFs challenging: statefulness and low packet processing latencies. Statefulness makes dynamic scaling challenging because it requires load balancing techniques that split traffic across new and old instances in a manner that maintains affinity between packets and their associated (control, per-flow or aggregate) state while also maintaining a good load distribution among the replicas. Such load balancing techniques have also to be fast to avoid any noticeable disruption to applications. Finally, they shall be compatible with the resource and feature limitations of commodity hardware switches (e.g. limited flow table sizes, features, and rule update times). A description of how these challenges constrain the design space is provided in clause 6.2.

- Challenges for Fault-Tolerance: Akin to the above discussion, VNFs exhibit three characteristics that, in combination, make recovery from failure challenging: statefulness, very frequent non-determinism, and low packet-processing latencies. As mentioned earlier, many VNFs are stateful. With no mechanism to restore lost state, backup VNFs may be unable to correctly process packets after failure, leading to service disruption. Thus, failover solutions shall correctly restore state such that future packets are processed as if this state was never lost (see clause 6). This could be achieved in many ways. For example, an 'active:active' operation could be deployed, in which a 'master' and a 'replica' execute on all inputs but only the master's output is released to users. One problem with this approach is that it is inefficient, requiring 1:1 redundancy for every VNF. More egregiously, this approach fails when system execution is non-deterministic, because the master and replica might diverge in their internal state and produce an incorrect recovery. Similarly, such non-determinism prevents replicated state machine techniques from providing recovery in this context.

Non-determinism is a common problem in parallel programs when threads 'race' to access shared state: the order in which these accesses occur depends on hard-to-control effects (such as the scheduling order of threads, their rate of progress, etc.) and are thus hard to predict. Unfortunately, as mentioned earlier, shared state is common in many VNFs, and shared state such as counters, caches or address pools may be accessed on a per-packet or per-flow basis leading to frequent non-determinism. In addition, non-determinism can also arise because of access to hardware devices, including clocks and random number generators, whose return values cannot be predicted. Any failure recovery technique shall cope with all of these sources of non-determinism. As described in the following clauses, the common approach to accommodating non-determinism is to intercept and/or record the outcome of all potentially non-deterministic operations. However, such interception slows down normal operation and is thus at odds with the other two characteristics of traffic processing applications, namely very frequent accesses to shared state and low packet processing latencies. Specifically, a piece of shared state may be accessed 100 k -1 M times per second (the rate of packet arrivals), and the latency through the VNF should be in 10 - 100 s of microseconds. Hence, mechanisms for fault-tolerance shall support high access rates and introduce extra latencies of a similar magnitude.

6 Methods for Achieving High Availability

6.1 High Availability Scenarios

Two situations where Scalable Architecture methods can be considered for reliability management are:

1) Unexpectedly large surges in incoming traffic - this situation can be mitigated by dynamically scaling out new VNFs to handle additional traffic loads.

2) Network failures - failures in element hardware, virtualisation layer, and VNFs that require fast recovery.

Each situation may have traffic flows undergoing different types of state; such state needs to be replicated for successful instantiation of new VNFs. As described in clause 5.2, there are various types of state that need to be considered:

1) Control State

2) Per-flow State

3) Aggregate State:

   a) Single-server Case

   b) Multiple-server case

Combinations of these situations and the type of state can be stated as straightforward, non-trivial but common, uncommon and difficult.

There are three straightforward cases:

1) Dynamic scaling of control state: control thread pushes updates to all other threads or servers or to a shared repository. The rate of updates can be tuned based on the VNF's consistency requirements for that state. If updates are to be atomic, then standard two-phase commit protocols can be used to push out updates although this will necessarily constrain the frequency of updates.

2) Recovery of control state: the control thread can checkpoint state before pushing it out to the other threads. If a thread other than the control thread fails, it can simply restore it from the control thread's copy.

3) Recovery of per-flow state: the techniques needed here are a strict subset of the ones needed for recovery of aggregate state; this involves simple black-box checkpoint and replay techniques. This is because per-flow state is local to a single thread and is not shared state. This case will be discussed as part of the case involving recovery of aggregate state.

There are three non-trivial but common cases:

1) Dynamic scaling of per-flow state.

2) Dynamic scaling of single-server aggregate state.

3) Recovery of single-server aggregate state.

Two techniques - Migration Avoidance and Lightweight Rollback Recovery - are presented below for addressing these three non-trivial but common cases.

Finally, there are two difficult cases involving multi server aggregate states; multi servers may be necessary if the total throughput demand of a flow aggregate cannot be handled by a single server:

1) Dynamic scaling of cross-server aggregate state - this is difficult because reads and writes now have to be synchronized across servers. While algorithms exist for this (e.g. Paxos), they are very slow.

2) Recovery of cross-server aggregate state - this is difficult for the same reason as above.

These two cases are for further study.

6.2 Dynamic Scaling with Migration Avoidance

Solutions for dynamically scaling a service are important for efficient use of infrastructure. In particular, a process is required whereby existing VNFs that are overloaded get supplemented with newly instantiated VNFs (scaling out) and the surging traffic load is now split over the increased number of VNFs. This clause describes a Migration Avoidance method [i.2] for scaling out the number of VNF instances; methods for contraction when underutilized (scaling in) are not presented, but are similar in spirit. The key idea behind migration avoidance is to have all processing switches (hardware and software) act in concert in order to meet the affinity requirement - ensure that all flows are directed to the VNF instance that maintains the state needed to process that flow. It is best applied to two of the three non-trivial but common cases:

1) Dynamic scaling of per-flow state

2) Dynamic scaling of single server aggregate state

Details on the underlying architecture as well as the individual processing steps for migration avoidance are provided below.

Under overload conditions, new VNF instance(s) are instantiated and the incoming traffic is split between the original VNF and the new VNF. The techniques described in this clause build on a few system components that are assumed to exist but that are not described here in detail; these are [i.2]:

1) Overload detector: this component detects when a VNF instance is overloaded and then notifies the system-wide controller; the specifics of how such overloads are detected is orthogonal to the techniques for reacting to overloads that are described below.

2) Placement: this component determines where - i.e. at which server core(s) - the new VNF instance(s) should be placed; the specific logic used to determine placement is orthogonal to the techniques described below.

3) Creation of new VNF instances: this component instantiates new VNF instances (whether processes, containers or VMs) at specified cores (as computed by the placement component).

Once overload is detected and new VNF instances are installed at the appropriate cores, the remaining component needed for dynamic scaling is to configure the network to direct traffic to the appropriate VNF instances. As shall be seen, some of the situations discussed below involve migrating dynamic VNF state from the original to the new VNF instances. The remainder of this clause focuses on techniques for this component.

Most VNFs are stateful (i.e. processing a particular packet depends on state established by the arrival of previous packets in that flow or aggregate of flows) and therefore they require affinity, where traffic for a given flow (or aggregates of flows) has to reach the instance that holds that flow's (or aggregate's) state (for example, if there are multiple instances of a NAT VNF, then a packet p should be sent to the NAT instance that maintains the address mappings for p's flow). Note that the techniques described in this clause only require that the flow ID can be computed based on information available at the switch, such as the packet's header fields. When a VNF instance is replicated, its input traffic will be split in a manner that preserves the VNF's affinity requirement.

A high level overview of the architecture for this technique is depicted in figure 4 below. Note that this is an illustrative example - other architectures are possible. Key components in this architecture are:

1) Servers (S1 and S2 in figure 4) over which VNFs are instantiated - new VNFs are instantiated by replicating existing ones over these servers. In the figure, VNF A on server S1 is replicated with a new instance A' which is instantiated on server S2.

2) Software Switches are run on each server; the VNFs are run over these switches.

3) Hardware Switch connects the servers together and it directs flows to the servers based on which VNF the flows need to be processed by.

4) External Controller coordinates and manages the entire process.

The techniques described in the remainder of this clause are based on the following assumptions:

1) Each flow can be mapped into a flow ID (e.g. via a hash function applied to relevant header fields) in a specified interval R.

2) Each VNF instance is associated with one or more sub-intervals (or range) within *R*. A range is described as a subset of the original interval *R*; note that ranges are non-overlapping.

3) The hardware switch can compute the flow ID and determine which range within R it belongs to using a manageable set of rules.

4) The software switch on each server tracks the active flows that a VNF instance on that server handles. The notation $F_{old}(A)$ is used to refer to the set of flows handled by the VNF instance *A* prior to the instantiation of *A'*.

A VNF instance *A* will have a range filter installed in the server's software switch or in the hardware switch. When replicating a VNF instance *A* with *A'*, the range previously associated with *A* shall now be partitioned between *A* and *A'*. For example, if the original range for *A* is [X, Y], then the partitioning into two new ranges is done without overlap, such as:

- Range for *A* prior to split: [X, Y]
- Range for *A* after split: [X, M) - this notation indicates that time instance M is not part of the range for *A*
- Range for *A'* after split: [M, Y] - this notation indicates that time instance M is part of the range for *A'*

The goal is to partition the range in a manner that ensures good balance of load across *A* and *A'* while ensuring that affinity for the dynamic state associated with flows in $F_{old}(A)$ is maintained.

Some perspective can be gleaned from describing two opposite and extreme positions as follows:

1) Never Migrate: In this approach, dynamic state is never migrated across VNF instances and instead, affinity is achieved by inserting special 'exception' rules for flows in $F_{old}(A)$ in the hardware or software switch. For example, an exception rule can be inserted for every previously active flow from $F_{old}(A)$ that falls in the range now associated with the new VNF instance *A'*. In practice, the number of exceptions here can be very large. It may be possible to reduce the range (or sub-interval) for the new VNF instance *A'*, but this results in an uneven traffic split. This problem may arise when the filters are installed on a hardware switch, and *A* and *A'* reside on different servers.

2) Always Migrate: In this approach, dynamic state for flows from $F_{old}(A)$ that fall in the range associated with *A'* is migrated from *A* to *A'*. The filter on the switch is then updated with two replacement filters; the original range is split into two new non-overlapping ranges as described above. This method is highly scalable in its use of switch rule table space and achieves a good balance of load between *A* and *A'*. However it does rely on support for state migration which may not exist in legacy applications. In addition, it can be complex and expensive to implement.

The disadvantages of these two extremes are dependence on state migration techniques to move the relevant dynamic state from one instance to another or the need for large rule sets in switches. One way to avoid these disadvantages is the technique of Migration Avoidance stated as follows. The key idea in migration avoidance is to have the hardware and software switch act in concert to maintain affinity. Migration avoidance does not require state migration (state migration can be implemented as an optimization to migration avoidance but is not strictly required), achieves good load balance, and is designed to minimize the number of flow table entries used on the hardware switch to pass traffic to VNF instances. The approach is described as follows:

1) Upon splitting, the original range filter on the hardware switch is initially unchanged, and the new filters (two new ranges plus exceptions) are installed in the *software* switch of the server that hosts A.

2) As flows in $F_{old}(A)$ gradually terminate, the corresponding exception rules can be removed.

3) When the number of exceptions drops below some threshold, the new ranges and remaining exceptions are pushed to the hardware switch, replacing the original filter rule. The purpose of this threshold is to exit migration avoidance once the number of exception rules remaining can be accommodated by the hardware switch. Once migration avoidance terminates, flows processed by the new VNF instance *A*' are no longer detoured through the server that hosts *A* and are instead directly forwarded from the hardware switch to the server hosting *A'*.

By temporarily leveraging the capabilities of the software switch, migration avoidance achieves load distribution without the complexity of state migration and with efficient use of switch resources. The trade-off is the additional latency to new flows being punted between servers (but this overhead is small and for a short period of time) and some additional switch bandwidth (again, for a short duration) - this level of efficient performance is quantified for these overheads in clause A.1.

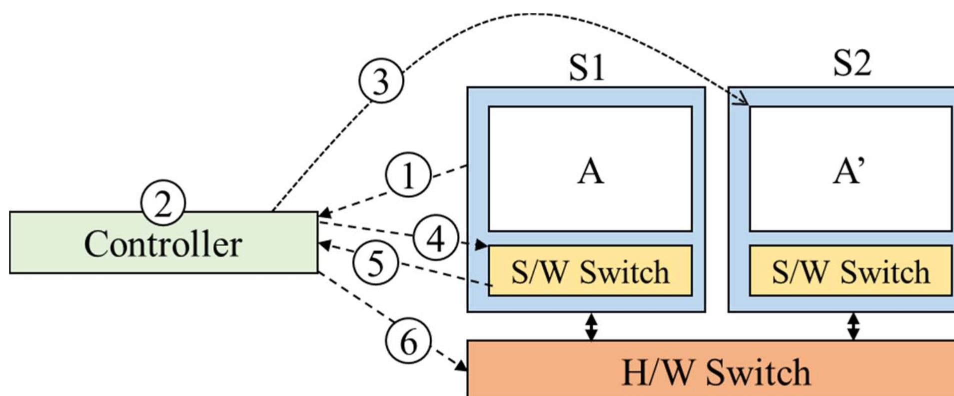

Figure 4: High Level Migration Avoidance Architecture

A step-by-step process for Migration Avoidance is as follows:

1) The controller is notified that VNF *A* is overloaded. There are many ways in which overload could be detected and communicated (e.g. this could be done by the VNF itself, or by external monitors); this process does not mandate any particular method for overload detection and communication.

2) The controller determines where a new VNF instance should be placed. In the example in figure 4, the controller chooses server S2.

3) The controller instantiates the new VNF instance *A'* at server S2.

4) As introduced above, [X,Y] denotes the range for *A* prior to the replication and [X,M) and [M,Y] the ranges for *A* and *A'* after the replication; similarly, $F_{old}(A)$ denotes the flows that were active at *A* prior to the replication. The controller's next step is to configure the software switch at S1 to send packets from *new* flows that fall in the range [M,Y] to the server S2 while continuing to send packets from flows in $F_{old}(A)$ to VNF instance *A* at S1.

5) As flows in $F_{old}(A)$ gradually terminate, their corresponding rules are removed from the software switch at S1. Once the number of active flows in $F_{old}(A)$ that belong to the range [M,Y] drops below a configured threshold value *T*, the controller is notified.

6) The controller now configures the hardware switch to achieve the following:

   a) Packets from active flows in $F_{old}(A)$ that fall in the range [M,Y] are sent to S1 (as per the above, there will be fewer than *T* of these).

   b) Packets from flows that fall in the range [X,M) are sent to S1.

   c) Packets from flows not in $F_{old}(A)$ that fall in the range [M,Y] are sent to S2.

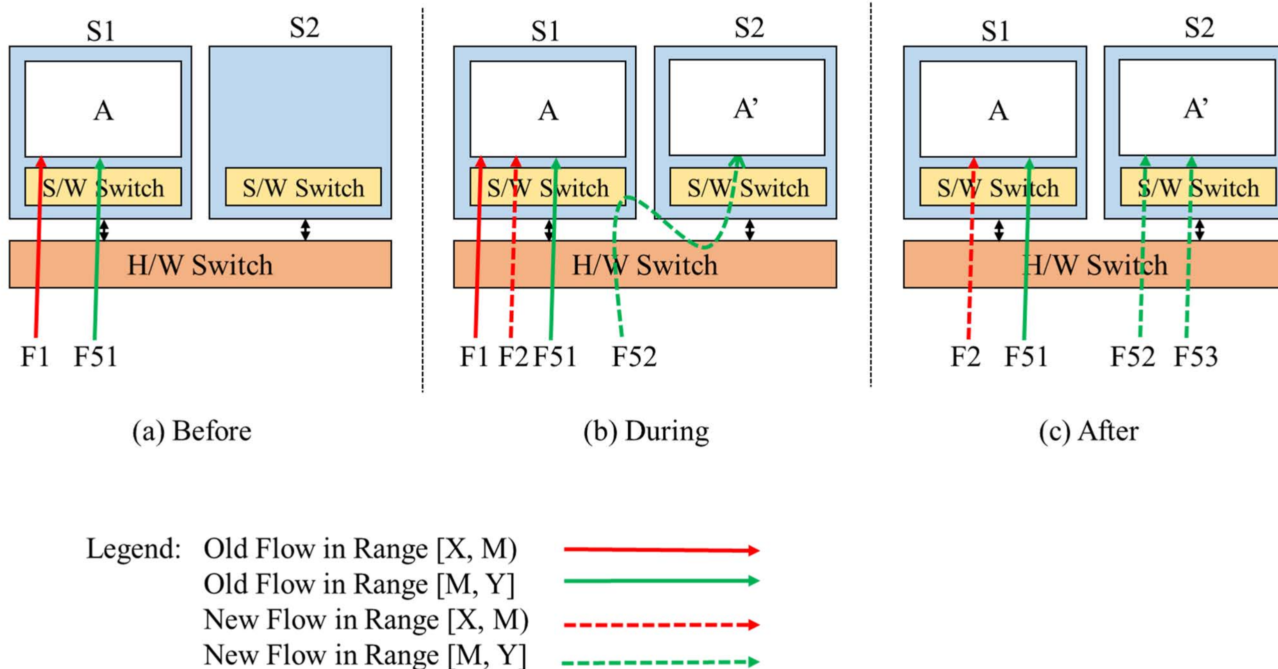

Figure 5: Example traffic flows using migration avoidance

Figure 5 shows an example of packet forwarding before, during and after migration avoidance. In the figure, assume that a range [0,100] is split as [0,50) and [50,100]. Note that F_i denotes a flow with ID equal to i. Before the split, all flows are processed by the original instance *A* as shown in figure 5 (a). After *A′* is instantiated, all flows are still sent to S1 by the switch. New flows that belong to [50,100] are redirected by software switch to instance *A′* (Flow 52). All existing flows (F1, F51), and new flows in the range for instance *A* are processed by *A* as shown in figure 5 (b). As existing flows in the range [50,100], e.g. F51, die out and the number of flows falls below a certain threshold, the controller inserts exception rules and split rules in the hardware switch. Thus in figure 5 (c) all the flows are redirected by the hardware switch to the corresponding server:

- Old flows in $F_{old}(A)$ that fall in the range [M,Y] are sent to S1 (Flow F51)
- Flows that fall in the range [X,M) are sent to S1 (Flow F2)
- Flows not in $F_{old}(A)$ that fall in the range [M,Y] are sent to S2 (Flows F52 and F53)

The process described here is based on the assumption that the hardware switch will be able to handle the diminishing number of exception rules (as stated in step 6 above) once the number of old flows drops below the threshold value. In the experimental results stated in clause A.1, that assumption is shown to hold true. However, there may be other types of physical element combinations where the selected hardware switch may not be able to handle the flow routing as described in step 6.

In such cases, an alternative method could be considered whereby the hardware switch simply directs all flows to *A* and *A′* depending on where they fall on the range:

- [X, M) ➔ *A*
- [M, Y] ➔ *A′*

All remaining old flows $F_{old}(A)$ that get directed to *A′* are then rerouted by *A′* to *A* - see figure 6.

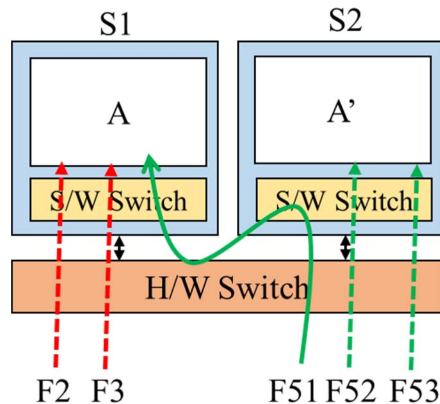

Figure 6: The Final Step (Step 6) with Alternative Method

Investigation of the pros and cons of these two alternatives will depend on the selection of the physical element configurations; it is a topic for further study.

6.3 Lightweight Rollback Recovery

6.3.1 Overview

Recovery is triggered when a VNF fails due to failure at the server, e.g. hardware, driver, or host operating system. The objective of recovery systems is to quickly enable a "backup" or "replica" to assume the role of the lost VNF immediately after it fails, thus masking the failure to clients. This method applies to the non-trivial but common case for recovery of single server aggregate state.

The goals for recovery are stated as follows. The principal requirement is that recovery should be correct - A system recovers correctly if its internal state after a failure is consistent with the observable behaviour of the system before the failure [i.1]. It is important to note that reconstructed state need not be identical to that before failure. Instead, it is sufficient that the reconstructed state be one that could have generated the interactions that the system has already had with the external world. This definition leads to a necessary condition for correctness called "output commit", which is stated as follows: no output can be released to the external world until all information necessary to recreate internal state consistent with that output has been committed to stable storage. The nature of this necessary information varies widely across different designs for fault-tolerance as does the manner in which the output commit property is enforced. In the context of VNFs, the output in question is a packet and hence to meet the output commit property, the process needs to ensure that before the VNF transmits a packet p, it has successfully logged to stable storage all the information needed to recreate internal state consistent with an execution that would have generated that packet p.

In addition to performing correct recovery, there are four other desirable properties for a recovery architecture:

1) Low overhead on failure-free operation - Recovering lost state necessitates some instrumentation to record and ensure that it is backed up; this instrumentation will inevitably introduce some performance overhead under normal operation. As will be shown, this overhead typically manifests in increased packet latency. As discussed previously, typical VNF latency should be in the 10-100s of microseconds; while a recovery architecture may increase latency (even under failure-free operation), mechanisms that commensurately introduce no more than 10-100s of microseconds of added delay to packet latency are preferred.

2) Fast Recovery - Recovery from failures shall be fast to prevent degradation in the end-to-end protocols and applications. Recovery times that avoid endpoint protocols like TCP entering timeout or reset modes are preferred. The minimum TCP timeout is 200 ms; hence the goal for recovery times is under this threshold value.

3) Generality - A general approach is preferred; the approach should not require complete rewriting of VNF applications nor needs to be tailored to each VNF application.

4) Passive Operation - Dedicated replicas for each VNF application are not desired, instead solutions that only need a passive replica are preferred, such that the backup server can be shared across active master instances.

A general illustrative architecture for recovery is shown in figure 6.

NOTE: This architecture can be implemented in many ways. For example, the Input and Output Handler functions can be implemented in the NICs (figure 3). Detailed examination of architectural options is a topic for further study.

The requirement for any architecture is that it needs to be sufficiently robust in order to support the recovery of a failed VNF. The goal here is to demonstrate how this general architecture can be used to implement three forms of rollback-recovery:

- Approach #1: Checkpointing
- Approach #2: Checkpointing with Buffering
- Approach #3: Checkpointing with Replay

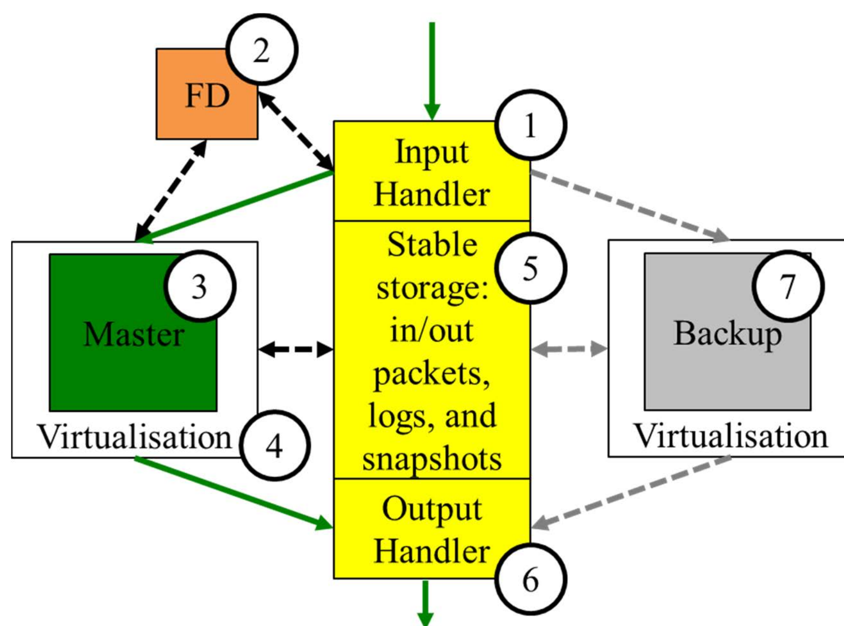

Figure 7: Rollback Recovery Architecture

In all three approaches, inbound packets flow from the input handler to the master VNF to the output handler and finally, to egress. Each of the three approaches make different trade-offs among the stated goals; these trade-offs are summarized in clause 6.4. Checkpointing and checkpointing with buffering are the most general: they work with any legacy VNF binary, even those which are not specifically configured for high availability. However, checkpointing achieves generality by sacrificing correctness, and checkpointing with buffering achieves generality by sacrificing latency. Checkpointing with replay achieves all stated goals. However, it can be applied only if the VNF source code is available for modification (via automated instrumentation) such that the required application state can be exported. These trade-offs are presented as part of the design of each approach.

6.3.2 Checkpointing

The key idea behind Checkpointing is to periodically take complete system snapshots of the running VNF [i.3]; Backup VNFs can be provisioned immediately by restoring (on a separate host) the most recent snapshot of the failed VNF.

The tasks for each component (figure 7) are as follows:

1) Input Handler. The input handler performs VNF selection. It forwards inbound traffic to either the Master VNF (if a failure has not been detected) or the Backup VNF (after receiving a signal from the Failure Detector). Once the Failure Detector declares the Master VNF to have failed, the Input Handler shall not forward traffic to the Master VNF any longer. This is because an intermittent failure could lead to two copies of the same VNF running at once and packets are duplicated.

2) Failure Detector. The failure detector declares whether or not the Master VNF has failed. If so, it signals to the Input Handler to cease forwarding to the Master VNF and begin forwarding to the Backup VNF. It also signals to the Backup VNF to begin the recovery process by loading the most recent snapshot from stable storage.

3) Master VNF. In this approach, the Master VNF is only utilized for normal traffic processing.

4) Virtualisation Layer. The virtualisation layer hosting the Master VNF periodically takes system snapshots and delivers them to stable storage; these snapshots encapsulate the state of the entire running codebase of the VNF. These snapshots may be virtual machine [i.4], [i.5] or container or process snapshots, depending upon the system implementation.

5) Stable Storage. The stable storage records the snapshots from the Master VNF.

6) Output Handler. In this approach, the Output Handler is not necessary.

7) Backup VNF. The Backup VNF is idle until receiving a signal from the Failure Detector, at which point it loads the most recent snapshot from Stable Storage and begins processing traffic received from the Input Handler.

Trade-offs: This approach imposes little latency penalty and can apply to any legacy VNF binary. However, this approach fails the stated goals quite fundamentally. Checkpointing alone cannot guarantee correct recovery after failure because snapshot restoration leads to loading a Backup which represents an "old" version of the Master VNF: most checkpointing approaches can perform snapshots at best on the order of every few tens or hundreds of milliseconds.

Note that checkpointing is done at periodic intervals. Assume for example that a snapshot is taken at time $t = n$ and the next one is scheduled at time $t = n+1$. Suppose the VNF fails after time $t = n$ and before time $t = n+1$. Then the snapshot available to the Backup VNF does not represent the latest state of the Master VNF.

Recovering to an old snapshot would lead to unknown behaviour for any connection whose packets were processed in the interval between when the last snapshot was taken, and when the Master VNF failed; such behaviours might be connection resets (in the case of appliances such as NAT), or failure to detect attacks (in the case of appliances such as IDS). Consequently, this approach in general is not recommended.

6.3.3 Checkpointing with Buffering

Checkpointing with buffering overcomes the correctness problem of checkpointing by withholding packets in a buffer until after a checkpoint completes. By introducing this delay, the system can guarantee that no connection ever observes a packet such that the state relevant to that packet has not already been checkpointed, hence guaranteeing correct recovery from failure.

The tasks for each component (figure 6) are as follows:

1) Input Handler. The Input Handler performs VNF selection as in Approach 1. In addition, the Input Handler is responsible for ensuring that failover does not result in a burst of packet loss. To achieve this, the Input Handler keeps a copy of each packet received and forwarded to the Master VNF since the last checkpoint at the Master VNF. After failure, the Input Handler begins forwarding incoming traffic to the Backup VNF, starting from these copied packets.

2) Failure Detector. The Failure Detector is the same as in Approach 1.

3) Master VNF. The Master VNF is the same as in Approach 1.

4) Virtualisation Layer. The virtualisation layer is the same as in Approach 1.

5) Stable Storage. The Stable Storage is the same as in Approach 1.

6) Output Handler. The Output Handler is responsible for enforcing the output commit property and hence ensuring correctness. The Output Handler keeps a holding buffer and does not allow packets to egress the system until after a checkpoint has completed, i.e. a snapshot has been recorded and acknowledged. Once a checkpoint is complete, all packets which arrived at the Output Handler prior to taking the snapshot can safely be released to the network.

7) Backup VNF. The Backup VNF is the same as in Approach 1.

Trade-offs: Checkpointing with buffering guarantees correctness where simple checkpointing does not. However, delaying packets at the Output Handler imposes increased latency on the order of the checkpoint interval; currently available VM Checkpointing approaches can perform checkpoints at best on the order of every few tens of milliseconds. As shown in clause A.2, an off-the-shelf VM checkpointing application with buffering implemented in this way increases median per-packet latencies to around 50 milliseconds.

6.3.4 Checkpointing with Replay

Checkpointing with Replay [i.6] does not withhold packets until a checkpoint completes, and hence it has a lower latency penalty during failure-free operation than checkpointing with buffering does. Recovery under replay consists of two steps. After the input handler detects failure, the Backup loads the most recent snapshot from Stable Storage received from the Master VNF just as in checkpointing with buffering. However, packets have been released by the output handler since the most recent snapshot (unlike checkpointing with buffering, where packets are delayed between snapshots), hence this snapshot is not a correct replacement for the Master VNF. The next step then is to restore the VNF state which was created or modified since the last snapshot, but before the Master VNF failed. To do this, the Backup VNF processes all duplicate stored packets from the input logger. These packets enable the backup to replay locally what actions the Master VNF performed prior to failure. If the VNF is deterministic, re-processing the same packets the Master VNF saw in the same order is sufficient to recover the lost state. However, as discussed in clause 5, VNFs exhibit a high degree of non-determinism. Hence, for correct state recovery, the Backup VNF requires not only the input packets, but a secondary set of logs from the Master VNF, where the logs constitute a record of every non-deterministic event occurring at the Master VNF since the last snapshot. This is known as the determinant log. They include records of all accesses to variables accessible by multiple cores, as well as calls to fundamentally non-deterministic functions such as gettimeofday().

The tasks of each component (figure 7) are as follows:

1) Input Handler. The Input Handler is the same as in Approach 2.

2) Failure Detector. The Failure Detector is the same as in Approaches 1 and 2.

3) Master VNF. The Master VNF processes packets as in Approaches 1 and 2. Additionally, the Master VNF will create logs recording every non-deterministic event which occurs in the system, e.g. accesses to cross-core state and calls to gettimeofday(). Whenever a nondeterministic event occurs, the Master VNF generates a log packet and transmits this log over the same egress interface that it uses to transmit data packets (hence sending the logs to the Output Handler). There are many possible approaches to recording nondeterministic events, see [i.6].

4) Virtualisation Layer. The virtualisation layer is the same as in Approach 1.

5) Stable Storage. The Stable Storage is the same as in Approach 1 and 2, with the addition that it also stores the logs for the Master VNF and, just as the snapshots, transmits them to the Backup VNF at failure time.

6) Output Handler. Just as in Approach 2, the Output Handler is responsible for enforcing output commit. However, the Output Handler no longer delays packets until a snapshot completes. Instead, it only delays packets until all determinant logs needed to replay the system up to and including that packet have been recorded to stable storage.

7) Backup VNF. Just as in Approach 2, the Backup VNF begins recovery by loading a snapshot from Stable Storage. It then enters a "replay mode" and begins re-processing packets which were originally processed by the Master VNF prior to failure. Whenever it attempts an action which may be non-deterministic, it consults the log from the Master VNF run and repeats the behaviour used in the original execution. All packets which are processed in replay mode are dropped rather than forwarded to the network, since clients have already seen copies of these packets from the Master VNF's original execution. Finally, after the log has been re-processed, the Backup VNF exits replay mode and can begin processing traffic as normal.

Trade-offs. Checkpointing with replay guarantees correctness. Further, the high latency introduced by checkpointing with buffering is not a problem with replay. This is because the VNF application records data that is required for recovery; any VNF codebase can be configured to take this approach, so long as the source code is available for modification (required modifications can be done via program analysis).

6.3.5 Summary Trade-offs of Rollback Approaches

Table 1: Trade-offs Between Rollback Recovery Algorithms

Approach	Correctness	Latency Overhead	Generality
Checkpointing	Not guaranteed	0 microseconds	Any legacy VNF binary
Checkpointing with buffering	Guaranteed	10 s of milliseconds (see note)	Any legacy VNF binary
Checkpointing with replay	Guaranteed	10 s of microseconds (see note)	Any legacy VNF source code
NOTE: Results in clause A.1.			

7 Recommendations

7.1 Conclusion

The present document demonstrates how Scalable Architecture techniques currently adopted in Cloud/Data Centres can be adapted for use in telecommunications networks in an NFV environment. In particular, two techniques have been described:

1) Migration Avoidance: This technique enables dynamic scaling of VNFs when existing VNFs are unable to cope with unexpected bursts of incoming telecommunications traffic.

2) Lightweight Rollback Recovery: This technique enables the recovery of failed VNFs without degrading existing traffic flows.

The development of specific architectures to support these techniques in actual network conditions is a topic for further study.

7.2 Guidelines for Scalable Architecture Components

Some observations can be made regarding the components of the two techniques; the intent is to assist in developing formal requirements for achieving high availability as follows.

1) Controller (Migration Avoidance): A controlling mechanism is necessary to oversee the process of dynamic scaling as described in clause 6.2. It can be implemented as a standalone device in support of this process or the functionalities can be implemented in other devices such as an orchestrator. The specific functions that need to be supported include:

   a) Receive notification of message indicating overload condition of a VNF and initiate migration avoidance process.

   b) Determine location (server) of new VNF instance and instantiate it.

   c) Configure rules in software and hardware switches to gracefully phase out existing old flows as described.

2) Overload Detector (Migration Avoidance): The role of this detector is to determine if existing VNFs become overloaded. This workload overload detection may be achieved by active monitoring methodologies described in clause 7 of the ETSI NFV "Report on active Monitoring and Fault Detection" Work Item [i.7].

3) Failure Detector (Lightweight Rollback Recovery): The role of this detector is to determine if a VNF (e.g. Master VNF in clause 6.3) is in a failed state. The fault detector proposed here may be conceived as a combination of active monitoring techniques for fault detection described in clause 7 of the ETSI NFV "Report on active Monitoring and Fault Detection" Work Item [i.7] and the architecture proposed by the OPNFV Doctor Project [i.8] which relies on NFVI analytics.

7.3 Future Work

Possible areas for future work identified in the present document include the following:

1) Multi-server Aggregate State Recovery - As described in clause 6.1, the multi-server case has certain complexities and will require enhanced techniques for:

   a) Dynamic Scaling of cross-server aggregate state

   b) Recovery of cross-server aggregate state

2) For Migration Avoidance, the process described in clause 6.2 depicts the role of the hardware switch in configuring the necessary rules for gracefully phasing out old flows. For the components selected for this study, the hardware switch had no problem accommodating these rules. However, this may not be the case depending on the selection of switch types. An alternative process that invokes the use of the software switch to configure the final rules for old flows can then be considered. The pros and cons of the two alternatives can then be evaluated for various types of selected components.

3) The checkpointing with buffering applications tested here indicated somewhat higher levels of latency than those experienced with checkpointing with replay. Additional methods/algorithms should be examined to reduce the checkpointing with buffering latency.

4) The checkpointing+replay process described in the present document was tested on hypervisor-based checkpoints. This process resulted in a long tail for latency (clause A.2.2). It is expected that this tail will shorten significantly if the checkpoints are done at the application layer given that the amount of data that needs to be written will typically be less. It would be fruitful to quantify the improvements expected from application-layer checkpoints.

5) Checkpointing/Logging Functions (Lightweight Rollback Recovery): The role of these functions is to correctly capture the state of existing flows in a VNF. This functionality could be considered as a mechanism for Passive Monitoring of VNF state information. The proper role of Checkpointing/Logging in particular and Passive Monitoring techniques in an NFV environment is a topic for further study.

Finally, it would be beneficial to demonstrate the performance of these techniques in telecommunications network type settings involving architectures and components that reflect expected NFV environments. Such demonstrations will enable the development of efficient architectures and subsequent formal requirements in order to support these processes.

Annex A (informative): Experimental Results

A.1 Migration Avoidance Results

This clause presents experimental results using a prototype of migration avoidance run on a small-scale experimental testbed of the form shown in figure 1 (clause 5.1 - Hardware Infrastructure). The testbed uses an Intel FM6000 Seacliff Trail Switch interconnected with 48 10G ports and 2 048 flow table entries. The switch interconnects four servers with a 10 Gbps link each. One server uses the Intel Xeon E5-2680 v2 CPU with 10 cores and two sockets and the remaining use the Intel Xeon E5-2650 v2 CPU with 8 cores and two sockets, for a total of 68 cores. The prototype's system architecture is as shown in figure 2 (clause 5.1 - System View) and uses SoftNIC [i.2], a software switch built on top of Intel DPDK. Results specific to the performance of SoftNIC are highlighted below. The controller that coordinates migration avoidance runs on a standalone server that connects with dedicated 1 Gbps links to each server and to the management port on the switch. All experiments use a traffic source and sink connected to four external ports on the switch. Unless stated otherwise, the results presented are based on a traffic workload of all minimum-sized 64B packets; qualitatively similar results were observed with workloads of different packet sizes.

The test evaluates migration avoidance for the scenario whereby a single VNF instance is replicated. The test uses 1 Gbps of input traffic, with 2 000 new flows arriving each second on average and flow length distributions drawn from published measurement studies [i.15]. This results in an input traffic load with an average of 10 000 concurrently active flows and hence dynamic scaling requires re-balancing load equivalent to 5 000 flows from the original VNF instance to the new instance. The test results that follow report the time it takes to redistribute load across the original and new VNF instance.

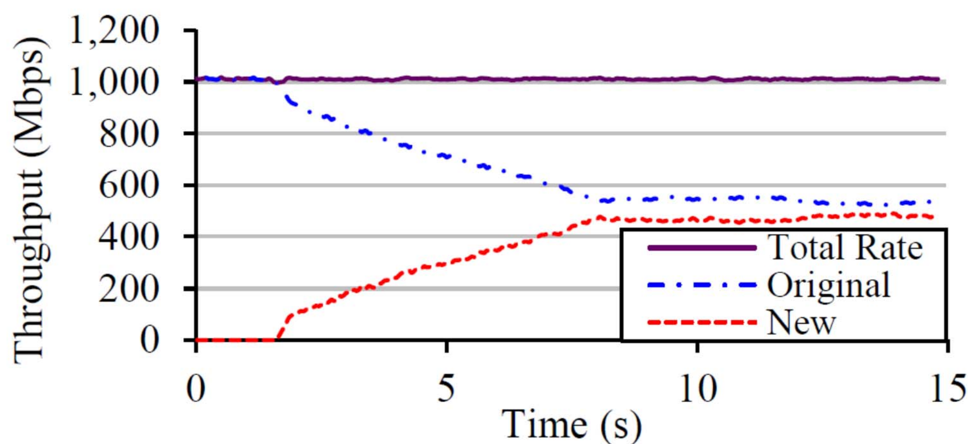

Figure A.1: Traffic Load at Original and New VNF Instance with Migration Avoidance

Figure A.1 shows the traffic load on the original and new VNF instances over time. Migration avoidance is triggered at time 2 seconds. The graph shows that migration avoidance is effective at balancing the load across the two instances: the difference in load on the two VNF instances is less than 10 % and convergence takes 5,6 seconds. This degree of imbalance and the time to convergence are largely a consequence of the test traffic used [i.15]. Specifically, the degree of imbalance that persists even after convergence (i.e. at the 7,5 second mark) depends on the size and duration of long-lasting flows. Similarly, the convergence time depends on the rate of arrival of new flows and the distribution of flow sizes (which determines the rate of flow completion).

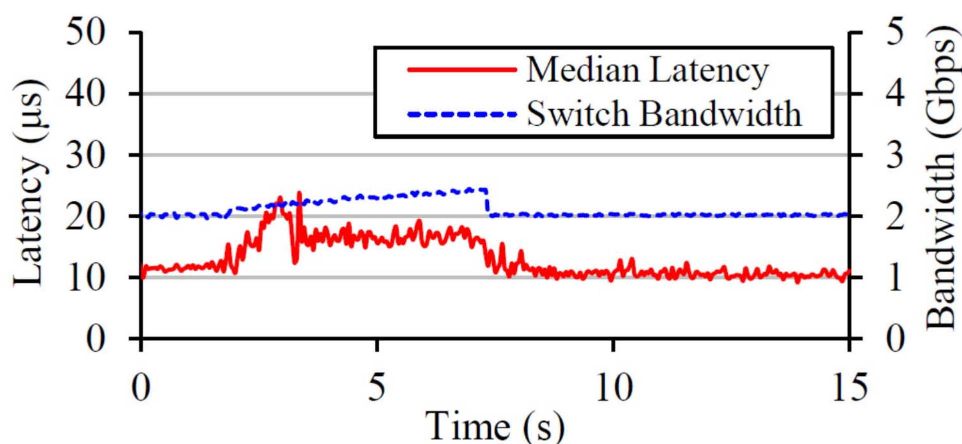

Figure A.2: Latency and Bandwidth Overheads of Migration Avoidance

The next set of results look at how active flows are impacted *during* the process of migration avoidance. Figure A.2 shows the corresponding packet latency and switch bandwidth consumption over time. The figure shows that packet latency and bandwidth consumption increase during the period (roughly between the two and eight second markers) when traffic is 'detoured' through the software switch at the original VNF. The graph shows that this (temporary) degradation is low: in this experiment, latency increases by less than 10 µsecs on average, while switch bandwidth increases by 0,5 Gbps in the worst case, for a small period of time. The latency overhead is a fixed cost that depends on the forwarding latency through the software switch and hence the absolute latency overhead reported in figure A.2 reflects the overhead of the DPDK-based software switch used in this experiment. The bandwidth overhead depends on the arrival rate of new flows in the input traffic trace.

In summary: migration avoidance balances load evenly (within 10 % of ideal) and within a reasonable time frame (shifting load equivalent to roughly 5 000 flows in 5,6 seconds) and does so with minimal impact to active flows (adding less than 10 µseconds to packet latencies) without requiring per-flow rules in the switch flow table.

A.2 Lightweight Rollback Recovery Results

A.2.1 Introduction

Results of recent studies on the performance of checkpoint + buffer fault- tolerance and checkpoint + replay fault-tolerance are presented here (note that checkpointing alone is not guaranteed to perform correct recovery).

The experimental setup is as follows:

- Xen 4.2 at the Master
- One of six different Click-based [i.9] middlebox implementations running in an OpenSuse VM. OpenSuse is chosen for its support of fast VM snapshotting [i.13].
- Test VFs include:
  - MazuNAT (an open-source combination NAT and Firewall released by Mazu Networks [i.10]),
  - Basic NAT,
  - WAN Optimizer,
  - Traffic monitor
  - QoS device
  - Adaptive Load Balancer.
- Local network of servers with 16-core Intel Xeon EB-2650 processors at 2,6 GHz, 20 MB cache size, and 128 GB memory divided across two NUMA nodes.

For all experiments shown, a standard enterprise trace was used as the input packet stream [i.11]; results are representative of tests run on other traces.

Three different system implementations are compared:

- Fault-Tolerant MiddleBox (FTMB) [i.6], a checkpoint + replay solution;
- Pico [i.12], a checkpoint + buffer solution with checkpoints taken at the application layer;
- Remus [i.14], a checkpoint + buffer solution with checkpoints taken at the hypervisor layer.

A.2.2 Latency

Figure A.3 shows a cumulative distribution function (CDF) of per-packet latencies through the testbed when running the MazuNAT VF. A given (x,y) coordinate means that y % of packets had latencies less than or equal to x microseconds. The plot contains four lines: a baseline configuration (where packets enter and exit the middlebox with no fault-tolerance enabled), replay fault-tolerance, and two forms of checkpoint fault-tolerance (checkpointing at hypervisor and checkpointing at application layer). In the baseline setup, per-packet median latencies are about 70 µs and at worst in the low hundreds of µs. As discussed in clause 6.3, all fault-tolerance mechanisms introduce some latency overhead. As expected, the checkpoint + buffer solutions introduce higher latency than the checkpoint + replay solution.

Checkpointing + buffering increases median latency by over 50 milliseconds when checkpoints are taken at the hypervisor, and by 8-9 milliseconds when checkpoints are taken at the application layer. The checkpoint + replay approach increases median latencies by only 30 microseconds. FTMB, the checkpoint + replay system relies on hypervisor-based checkpoints rather than application-layer checkpoints which results in milliseconds of tail latency (at the 95th percentile and above) [i.6]; the expectation here is that if FTMB used application layer checkpoints, this tail would decrease. This investigation is for further study.

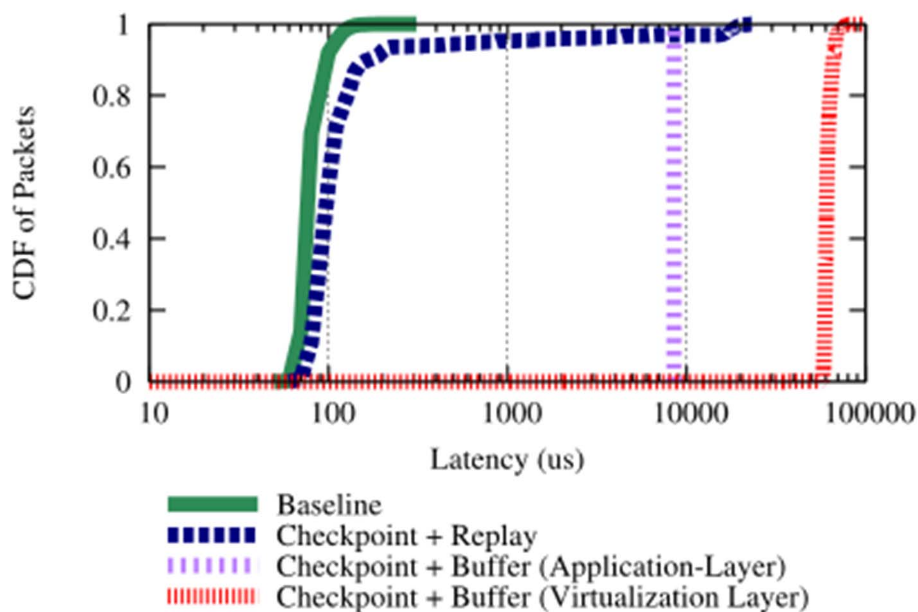

Figure A.3: CDFs of Packet Latencies

Summary: Checkpoint + replay recovery offers the lowest latency overhead of all fault tolerance approaches. Median latency penalties for FTMB were 30 µs.

A.2.3 Throughput

As discussed in clause 6.3, all fault-tolerance mechanisms introduce some throughput overhead. Figure A.4 shows baseline throughput in the testbed for six middlebox applications, compared against throughput for the FTMB Checkpoint + replay system. In the baseline configuration, all middleboxes could sustain throughputs of 1 to 6 million packets per second (Mpps). The MazuNAT and SimpleNat saw a total throughput reduction of 5,6 % and 12,5 % respectively. For the Monitor and the Adaptive Load Balancer overheads resulted in a 22 % and 30 % drop in throughput respectively. In these experiments, the two buffering implementations resulted in throughputs an order of magnitude lower, at best 100-200 thousand packets per second (Kpps).

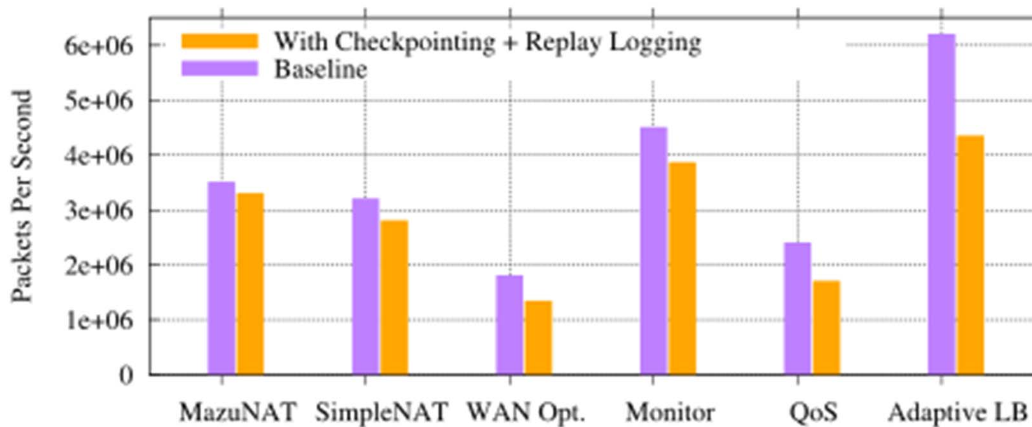

Figure A.4: Throughput Comparisons

Summary: Checkpoint + replay fault tolerance reduces VF throughput by at most 5-30 %.

A.2.4 Replay Time

On failure, checkpoint + buffer approaches load a new VNF by restoring from the most recently stored checkpoint. Checkpoint + replay approaches require a longer recovery time: not only they need to restore from the most recent checkpoint, but they also need to replay the logs stored before failure. Figure A.5 displays the impact of this replay overhead. The plot contains four bars per middlebox; each bar represents replay time when checkpoints are taken every 20, 50, 100, or 200 milliseconds. The amount of time to perform replay is dependent on:

- Complexity of the application - six VFs are tested here.
- Checkpoint interval, which determines how much data will be replayed.

The longer the checkpoint interval, the longer the replay time. Overall, replay times ranged between 10 milliseconds and 270 milliseconds, depending on these two variables. Application developers can tune recovery time by decreasing the checkpoint interval, and ensure that recovery will not result in application-layer timeouts (which are typically on the order of seconds).

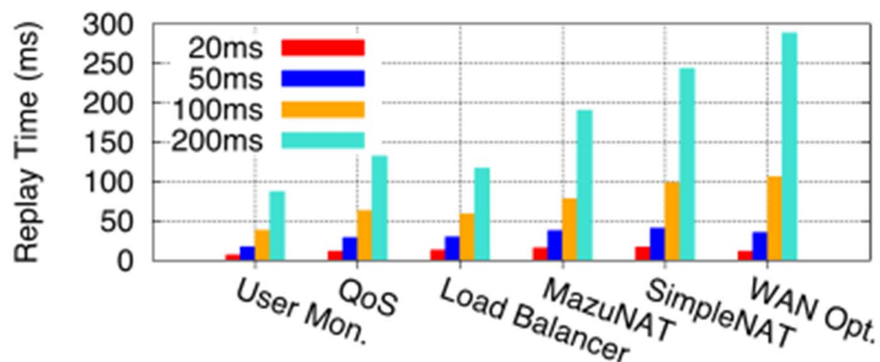

Figure A.5: Replay Time

Summary: Replay time increases time to recovery by tens to hundreds of milliseconds relative to checkpoint + buffer approaches, which is within acceptable ranges for application recovery times.

A.2.5 Conclusion

These experimental results suggest that while both checkpoint + buffering and checkpoint + replay approaches can provide correct recovery, checkpoint + replay offers a better performance trade-off. Buffering approaches have a faster recovery time, but impose latencies of 8-50 milliseconds on every packet processed under normal operation; this is prohibitively high for many NFV contexts [i.6]. In contrast, the checkpoint + replay system tested here increases median latencies under normal operation by only tens of microseconds, at the cost of a recovery time which is 10 s - 100 s of milliseconds higher, but still within typical application timeout bounds.

Annex B (informative): Authors & contributors

The following people have contributed to this specification:

Rapporteur:

Percy S. Tarapore

Contributors:

Percy Tarapore, AT&T

Richard Schlichting, AT&T

Randee Adams, Alcatel-Lucent

Stefan Antzen, Huawei

LAC Chidung, Orange

Gurpreet Singh, Spirent

Pasi Vaananen, Stratus

Carry Liu, Huawei

Acknowledgement: The Migration Avoidance and Lightweight Rollback Recovery techniques described here are based on research carried out by Sylvia Ratnasamy, Scott Shenker, Justine Sherry, and Keon Jang from the University of California, Berkeley, in association with AT&T.

History

Document history		
V1.1.1	September 2015	Publication