REINFORCE: Achieving Efficient Failure Resiliency for Network Function Virtualization-Based Services

Sameer G. Kulkarni, Guyue Liu, K. K. Ramakrishnan, *Fellow, IEEE, ACM*, Mayutan Arumaithurai, Timothy Wood, and Xiaoming Fu, *Senior Member, IEEE, Member, ACM*

***Abstract*— Ensuring high availability (HA) for software-based networks is a critical design feature that will help the adoption of software-based network functions (NFs) in production networks. It is important for NFs to avoid outages and maintain mission-critical operations. However, HA support for NFs on the critical data path can result in unacceptable performance degradation. We present REINFORCE, an integrated framework to support efficient resiliency for NF service chains. REINFORCE includes timely failure detection and consistent failover mechanisms. REINFORCE replicates state to standby NFs (local and remote) while enforcing correctness. It minimizes the number of state transfers by exploiting the concept of external synchrony, and leverages opportunistic batching and multi-buffering to optimize performance. Experimental results show that, even at line-rate packet processing (10 Gbps), REINFORCE achieves chain-level failover across servers in a LAN within 10ms, incurring less than 10% performance overhead, and adds average latency only ∼400 μs, with a worst-case latency of less than 1ms. REINFORCE also recovers from software failures within the same node in less than 100 μs, incurring less than 1% performance overhead and adds less than 5 μs latency during normal operation.**

***Index Terms*— Network function virtualization (NFV), service function chaining (SFC), network functions (NF), service function chains (SFC), fault-tolerance, availability, resiliency.**

I. Introduction

FAULT Tolerance (FT) and High Availability (HA) are important concerns for many network services. Studies show that middleboxes fail [1], [2] and software failures [3], [4] occur often. Recent work [1] estimates roughly 40% of network failures are caused by middleboxes, and Gill *et al.* [2] indicate that load balancers have the highest failure probability. Nearly a third (31%) of device failures are attributed to software related issues. Since these middleboxes operate inline with the network forwarding path, a software failure can significantly disrupt network operations.

Failure recovery time and the overhead for providing resiliency depends on the type of failure. *E.g.*, a software component crash can be quickly detected and fixed by the host operating system (OS) within a few microseconds, while recovery from OS failures may take at least a few milliseconds (e.g., 10-50ms for lightweight unikernels like ClickOS [5] and Mirage [6]) to reboot and restore the device. Hardware failures such as link and node failures may take seconds or more.

Network Function Virtualization (NFV) implements network services and middlebox functions (e.g., load balancers, firewalls, NATs, caching proxies) in software which can then be run on off-the-shelf commodity servers, avoiding the use of dedicated purpose-built hardware. However, an NFV-based data plane must compensate for the potential lower reliability of commodity hardware [7]. In addition, the presence of multiple layers of software including hypervisors or container libraries, guest OSes, system and application software, increase the chance of software failures. In this paper, we present REINFORCE, an integrated framework to support efficient resiliency for NFs and NF service chains.

Previous work such as FTMB [8] and Pico Replication [9] have tried to address reliability challenges for individual network functions. Such approaches introduce excessive overhead when adopted for service chains. Some chain-level approaches [10] seek to provide reliability guarantees across several NFs, but incur high latency due to packet buffering delays. Further, intra-node commit operations (such as the evaluation with FTMB [8] where the logging and storage components are co-located on the same node) may not factor the network latency that can impact overall system performance. Hence, a design choice that compensates for the network round trip latency incurred for the inter-node (*i.e.,* going to a different physical node) commit operations is desirable. CHC [11] addresses chain-wide failure resiliency with stateless NFs and an externalized datastore. The chain-wide correctness approach in CHC is operationally similar to our work, except that we operate with traditional stateful NFs, while CHC relies on the datastore to reliably maintain and update different versions of the NF state.

The goal of REINFORCE is to design comprehensive failover mechanisms that can *efficiently* provide *fast* failure recovery for NFs *and* service chains. Note the service chain can be either co-located on a single node or span multiple

Manuscript received February 25, 2019; revised August 11, 2019 and December 21, 2019; accepted January 21, 2020; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor P. P. C. Lee. This work was supported in part by the EU FP7 Marie Curie Actions CleanSky ITN Project under Grant 607584, in part by the U.S. NSF under Grant CRI-1823270, Grant CNS-1763929, Grant CRI-1823236, and Grant CNS-1422362, in part by the Department of the Army, U.S. Army Research, Development and Engineering Command under Grant W911NF-15-1-0508, and in part by a Futurewei Technologies Inc. grant. *(Corresponding author: Sameer G. Kulkarni.)*

Sameer G. Kulkarni and K. K. Ramakrishnan are with the Department of Computer Science and Engineering, University of California at Riverside, Riverside, CA 92521 USA (e-mail: sameer.sameergk@gmail.com).

Guyue Liu is with Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213 USA.

Mayutan Arumaithurai and Xiaoming Fu are with the Institut für Informatik, University of Göttingen, 37073 Göttingen, Germany.

Timothy Wood is with the Department of Computer Science, The George Washington University, Washington, DC 20052 USA.

Digital Object Identifier 10.1109/TNET.2020.2969961

nodes. Additionally, we aim to guarantee consistency properties for NF state and packet content under a variety of failure conditions including software, hardware (single node), and link failures. In order to ensure chain-wide correctness of operation and to address non-determinism, we maintain two distinct types of information for effective failure resiliency in an NFV environment: the application state (state for an NF or chain of NFs); and the packet processing progress (which can be characterized by a per-flow logical timestamp, as long as packets can be replayed after a failure). We employ *lazy checkpointing* (periodic, coarse timescale commit) of application state to reduce the overhead on normal operation and buffer input packets at a predecessor node in-between checkpointing instants. These input packets are replayed to the standby (backup) node upon a failure. Keeping track of packet processing progress of all the flows at least requires a per-flow timestamp, which is the critical information necessary to enforce correctness when the packets are replayed. The application state (state of NF or chain of NFs) can then be correctly recovered through replay. This allows us to commit the minimum amount of lightweight per-flow timestamp information at a finer timescale, while committing the more heavyweight application state at a coarser timescale. This distinction from prior work enables REINFORCE to achieve high performance under normal, failure-free operation.

A key insight of our work is to carefully separate how resiliency is provided for deterministic packet processing (replay and lazy checkpoints) from non-deterministic behavior (which requires full checkpoints to ensure consistency). Deterministic packet processing occurs when replaying the same set of input packets at an NF, or chain of NFs, has the same result, in terms of both the state of NF(s) as well as packets that are output on the wire from the NF (and/or NF chain). In contrast, non-determinism (ND) may not produce the same result upon replay. ND is not uncommon, *e.g.,* a NAT or load balancer may make a random choice for a flow's first packet [12], [13]. We annotate code paths which exhibit ND, and only when replay is not possible, trigger more expensive checkpoints.

REINFORCE guarantees correctness and achieves external synchrony (*i.e.,* Synchrony defined by externally observable behavior) [14] by speculatively processing packets and compactly committing per-flow timestamps to the standby node. We precisely replay to the backup only those input packets that had been processed by the primary between the checkpoint instant and a failure and coordinate checkpointing with non-deterministic actions to avoid inconsistencies. Thus, unlike FTMB [8], REINFORCE incurs no overhead for deterministic packet processing, eliminating the need for per-packet access logs at NFs and strict orderings of packets while replaying at the backup. Unlike Pico Replication [9], REINFORCE hides the replication latency and improves throughput by batching and overlapping multiple commit transactions, while allowing NFs to continue speculative execution. These improvements result in a dramatic performance improvement over existing approaches. To summarize, our key contributions include:

- **Integrated resiliency framework**: We present an efficient NFV resiliency framework for DPDK [15] based network functions and service chains with distinct local and remote redundancy schemes (§III-A).
- **Lightweight NF checkpointing**: We design mechanisms to minimize the state that needs to be replicated to the backup by taking advantage of logical clocks, external synchrony, 2-phase commit, and dirty state tracking to enforce correctness before releasing packets from an NF service chain (§IV-A).
- **Chain wide recovery**: We develop low overhead and low latency approaches for consistent recovery of all network functions in a chain within or across hosts (§III-B-§III-C).
- **Fast failure detection**: We devise ways to quickly detect NF (of the order of μs), link and node failures (in ms) (§III-D).
- **Optimization techniques**: We exploit non-blocking, pipelined NF processing with judicious batching and buffering to maximize throughput, minimize latency and avoid overheads during the normal operation of NFs.

II. Design Considerations

The key requirements for REINFORCE include:

Correctness and recovery transparency: NF state must be preserved and consistently recovered across the replica nodes in the event of a failure. In addition, for a service chain, it is necessary to ensure that all the NFs in the chain are able to process flows without interruption, by preserving the necessary processing state of each of the NFs in the chain.

Low overhead: NFs are typically expected to process millions of packets per second and serve large numbers of flows. CPU cycles and memory bandwidth are at a premium. Hence, it is necessary to minimize the performance impact of resiliency.

Generality: Given the diversity of network services and deployment patterns, it is necessary to ensure that the resiliency solution can be easily adopted for different types of NFs with minimal modifications to their code or infrastructure.

A. Deployment and State Management

Deployment: Our implementation focuses on NFs running inside containers, although many of our techniques can be generalized to other approaches. Containers enable low overhead snapshots using tools like CRIU (Checkpoint Restore In Userspace) [16]. Unfortunately, these cannot be trivially applied to NFs because they do not interact cleanly with user-space I/O frameworks like DPDK [8], [15]. Further, they cannot provide consistent checkpoints across groups of NFs run in different containers. For this reason, we develop a resiliency abstraction that can identify just the key NF state that needs to be backed-up for each container.

Service Chaining: NFs are typically chained, to efficiently process flows through multiple functional components. For example, we may have a Service Function Chain (SFC) for HTTP traffic to be processed through a NAT, Firewall, IDS, and Load-balancer NFs [17]. The ordering of NFs needs to be preserved, even when failures cause flows to be routed to a replica. The NF chain (ordered list) needs to be treated as a unit of processing rather than as individual NFs in isolation.

State Characterization: NFs keep a variety of state information, including configuration parameters, counters, flow connection status, and application specific variables. We focus on stateful NFs, *e.g.,* NAT, DPI or IDS, which may maintain global configuration state, as well as per-flow or per-connection state. We further classify state updates as either deterministic or non-deterministic (refer §II-D). Although many common middleboxes (*e.g.,* firewall, IDS, IPS) do not modify packet headers at all, or if they do, modifications are

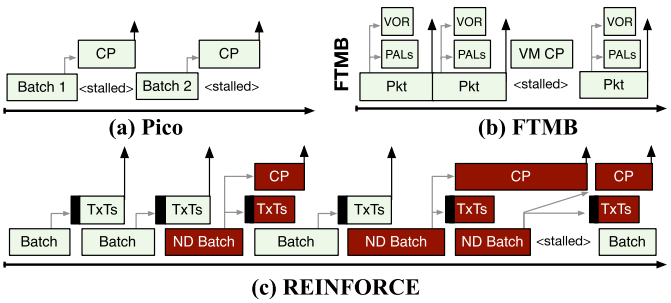


Fig. 1. Comparison of NFV resiliency mechanisms.

deterministic for the given input packet headers [18], we must also consider the packets traversing a service chain as state themselves, because other NFs may modify the packet data (*e.g.,* NAT). Therefore we must track their progress through the service chain. For correctness, all of this state must be properly synchronized to a backup for every NF in the chain.

B. Failure Model and Detection Schemes

Fast failure detection is a key to providing fast failover. Here, we only consider fail-stop software and hardware failures: software crashes, link status changes, power outages, *etc.*

Software Failures: We rely on low level kernel events like signals, traps, and syslogs that can be effectively checked (queried or polled) to determine the status of individual software NFs. REINFORCE assumes that we can recover from such failures by reloading the NF with a checkpoint of its recent state and reprocessing any intermediate packets.

Link and Server Failures: For hardware failure detection, we considered various state-of-the art Layer-2/Layer-3 schemes such as the Link Aggregation Control Protocol (LACP) and Open Shortest Path First (OSPF), including Software-defined networking (SDN) and Openflow-based Echo and Fast Failover schemes. Ultimately, we selected Bidirectional Forwarding Detection (BFD) [19], which is a lightweight, protocol-independent liveness detection protocol that can detect link failures at millisecond timescales. Although the timers can be set as low as 1 μs, we observed that such aggressive timeout values can result in excessive false-positives. We experimented with the default BFD values across a link connecting two nodes back-to-back with about 50% background traffic and observed that a timeout value below 1000μs still resulted in occasional false-positives.

C. Recovery: Replay vs. No-Replay

Pico Replication [9] first proposed NF resiliency with a pure checkpointing (i.e., no-replay) scheme that buffers all the output and stops NF processing until the completion of checkpointing (CP), to assure state consistency as shown in Figure 1(a). However, this buffering results in high latency and degraded throughput during normal operation. In order to ensure correctness of the state replicated to the backup, we need to pause NF packet processing and also hold the processed buffer until the NF state gets replicated.

An alternative proposed in FTMB [8] is to maintain input packet logs (at a predecessor node) and replay the log to reconstruct lost state after a failure. With this approach, output packets can be preemptively released before creating a full NF checkpoint since the state can be recreated by replaying the packets. However, to ensure state correctness during replay, it becomes necessary to track and commit all possible source of non-deterministic processing before releasing the packets. FTMB logs each access to every shared variable in a packet access log (PAL), and a vector clock (VOR) of PALs across all the threads to ensure the correct ordering of accesses to the shared variables during replay, as shown in Figure 1(b). FTMB can output packets without waiting for NF state to be checkpointed. By replaying the log of input packets, and the packet-access logs, the replica can recover the lost state and be reinstated correctly in the role of the primary NF. This approach overcomes the latency impact for a majority of the packets, but adds complexity to NF development and can incur high overhead to enforce sequential ordering. FTMB needs to do full system CP periodically and halt processing for the duration of CP (order of a few milliseconds), which is shown to result in high tail latencies and impact overall throughput.

REINFORCE uses a combination of infrequent (lazy) CP and replay of packets. The key is to maintain external synchrony: *i.e.,* rather than strict synchronization where NFs block until replication completes, REINFORCE allows NFs to continue speculative execution while replication is performed. Relaxing the constraint of synchronous replication and adopting external synchrony means that processing can continue through the service chain, and for subsequent packets in the flow, while still providing consistency guarantees to clients receiving the packets. When a failure occurs, the backup node can replay packets that have to be processed since the last checkpoint snapshot and update the NF application state on the backup. A logical timestamp is used to determine the packets that have been released since the checkpoint so that the replay process does not transmit those packets unnecessarily as duplicates downstream, while updating the backup state.

D. Non-Determinism

Let us examine the impact of non-determinism (ND) in some more detail. NFs operating on the same input (flow of packets) can still diverge in their internal state across multiple executions due to implicit or explicit ND in the processing [8], [20]. ND can occur due to i) dependence on hardware whose outcome cannot be predicted, such as hardware clocks, random number generators, *etc.* , ii) race conditions in accessing shared variables among NF threads, or iii) when the intermittent packets are marked for ECN/lost/dropped; then the order of packet arrival and subsequent processing may become nondeterministic [21]. For example, a load balancer (even with the "Active:Active" redundancy configuration) that assigns one server from a pool of backend servers for each TCP connection can end up choosing different backend servers for the same flow when the selection logic is based on system specific calls like `random()`. Similarly, during replay, a rate limiter that restricts the number of maximum sessions for a given client can end up rejecting different connections due to races in the NF threads accessing a shared connection variable.

FTMB [8] overcomes ND by rigorously tracking and ensuring that all the events that can potentially lead to non-determinism (any shared state access and outcomes of unpredictable system calls) are captured and committed to a stable log before releasing the packets. This way, even benign accesses to shared variables or non-deterministic calls (*e.g.,* shared counters) whose impact is unrelated to packet processing (*i.e.,* do not impact the external view) are logged and enforced at the replay node. The result is not only excessive logging overheads but also a limit on an NF's

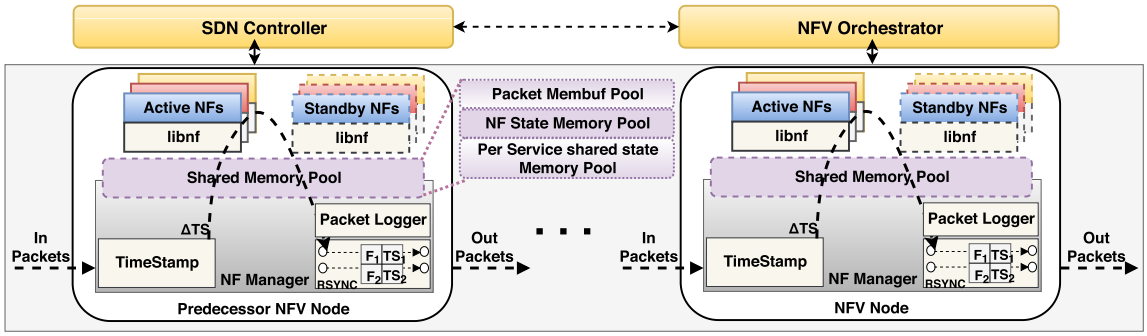


Fig. 2. Architecture of REINFORCE: Each NFV node hosts multiple NFs (*i.e.,* entire or part of NF chain) Memory pool is shared across NF Manger, active and local standby NFs. Operational symmetry is retained across all the nodes in the chain.

throughput during normal, failure-free operation. Further, with multi-threaded NFs, during replay FTMB enforces a strict ordering for accesses to any shared resources across multiple processing threads. Enforcing this ordering requires more intricate instrumentation of the NF's code and affects both normal and recovery mode performance.

To address ND, we present an alternative simpler approach without the need for per packet access logs or the need to enforce strict ordering of packet access to the shared variables. We exploit the fact that non-deterministic updates may typically be tied to specific packets, e.g., the first packet in a flow that causes non-deterministic updates at several NFs, while subsequent packets do not. However, we do not make any assumption on when non-determinism can occur. For example, L4-L7 NFs (say load balancing) may exhibit non-determinism after receiving/processing a specific byte stream and can be anywhere in the middle of the flow. Regardless, when an NF performs any non-deterministic state update (for which we require the programmer to annotate such operations) we link it to the packet (batch) which triggered it. Then, taking advantage of external synchrony, we only need to ensure that by the time the packet reaches the end of the service chain and is ready to be sent out, all of its dependent non-deterministic state has been checkpointed to the standby, avoiding the need to replay it after a failure. For example, in the load balancer example, it is sufficient to track the initial connection state update at the start of the flow, rather than tracking and enforcing the access to shared global counters for every packet processed by load balancer NF threads. Through timely checkpointing of the non-deterministic state updates, we eliminate the need to maintain per-packet access logs.

III. Architecture and Design

We present the key components of REINFORCE shown in Fig. 2, and briefly discuss their roles. We describe how REINFORCE performs failure detection and handles both local and remote failures while guaranteeing correctness.

A. REINFORCE Components

The NF Orchestrator is responsible for provisioning the NF Manager nodes and designating the active and standby nodes for different service chains. It also configures the BFD settings on each of the NF Managers in the cluster.

The SDN Controller is responsible for populating the flow entries and forwarding rules at each node's NF Manager. It pro-actively configures the back-up path options: a) with multiple links, it configures the alternate output ports on the predecessor nodes of the designated active node; and b) configures the flow rules on designated replica standby nodes.

The NF Manager is the core component of REINFORCE. It acts as the in-host controller for coordinating NF functionality, using DPDK's framework for zero-copy delivery of packet data to and between NFs of a service chain within the host. The NF manager tracks the liveness of associated network ports (links) and the NFs provisioned on it. It also provisions and provides the shared memory pools to the NFs to exchange packets, shared memory state, and message notifications. In addition, NF manager implements the "packet logger" module to log and timestamp all incoming packets, and "RSync" module to provide consistent state replication service to the NFs. We leverage both proactive and reactive configuration schemes along the lines of [7], [22].

Active NFs process incoming packets delivered to NFs via the NF Manager. Each NF is integrated with a "*libnf*" library that provides the necessary hooks to facilitate state checkpointing and recovery, thus minimizing changes required on the NF.

NF Standbys can be run on either the primary host (for local failover) or a secondary host (for remote failover). We choose an "Active–Hot Standby" configuration for NF resiliency, where the state updates from the active NFs are consistently committed on the corresponding standby NFs. Software failures that can be recovered by NF instances within the same host can be provisioned for 1:1 redundancy of active:standby NFs. We protect each individual NF instance in a chain, thus allowing REINFORCE to be resilient to multiple NF failures on the same node. We also support failures at the chain level, where all network function standbys of a chain are provisioned on a remote node (which can also host other active NFs). This supports link and node failures for both hardware and software. Multiple NF chains on different active nodes can be configured to share the same standby node.

The Predecessor Node is a server prior to the node hosting active NFs, which is responsible for logging the incoming packet stream. This is used to handle server or link failures that require the packets to be replayed to the standbys on a remote node for recovery. When the NF chain spans more than one node, the symmetry in our design allows for multiple predecessor nodes similar to the one shown in Figure 2. But this does not incur any additional inter-node communication overheads, as chain management and routing is governed locally by the NF managers on each of the nodes. Also, only the first predecessor node in the chain timestamps the packets.

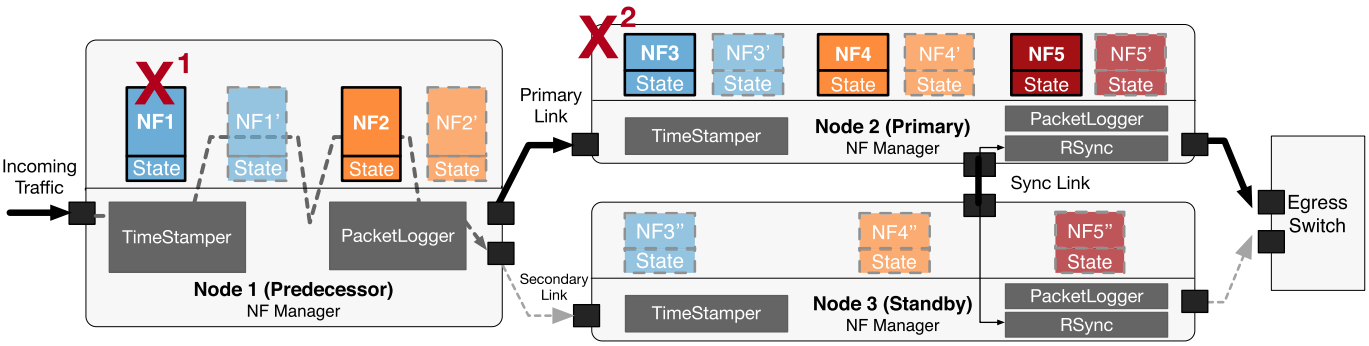


Fig. 3. On the left side is the local failover of NF Instance. Upon NF1 failure in Node-1, NF Manager initiates failover seamlessly to local replica NF1'. The right side represents the remote failover of NF chain (NF3, NF4, NF5) to the remote standby node. Upon failure of Primary (Node-2), the predecessor nodes (Node-1) initiates failover by replaying the packets from its logged buffer and also redirects the subsequent packets to the standby node.

The *libnf* routine exports the necessary interfaces and provides the common state replication/management functions to NF developers. NF developers need to annotate/set a bit field in packet headers to indicate the occurrence of a non-deterministic update (`pkt->header.nd = 1`) for any packet(s) in a batch. *libnf* checks for non-deterministic updates and correspondingly updates the NF processing state machine to decide whether to continue the NF processing or block/stall the processing. NF processing is blocked if there is another non-deterministic state update while an outstanding non-deterministic state update remains to be committed. Further, *libnf* combines the batch processing of packets to the NFs and handles the selective NF state synchronization (*i.e.,* only the dirtied memory is copied) to a local standby NF at the end of processing the batch of packets. NFs only need to specify the memory size and offset for the state update, then the *libnf* routine updates the corresponding bitmap (64-bit integer) indicating the dirtied/updated NF state. We expend additional CPU cycles in the active NF to synchronize the modified NF state to the local standby NF. Nevertheless, this is transparent to the NF and is completely handled by a set of library functions implemented in *libnf*.

B. Local Resiliency

In scenarios where only software crashes have to be tolerated, the standby NF (a.k.a local replica) is provisioned locally by the NF Manager to provide resiliency from NF instance failures as shown in the left part of Figure 3. After initialization, the standby NF remains in 'Paused' state until the NF Manager signals to wake it up. For more details refer to our work [23].

C. Remote Resiliency

We employ both checkpointing and packet "replay" to provide resiliency from host node failures and link failures (that result in loss of connectivity) when the backup is on another node, as shown at right in Figure 3. Note that our remote resiliency solution builds on top of the local resiliency and leverages the consistent state replicated at the local standby NFs for the state replication on remote node.

Standby server: The NF Orchestrator designates a standby node and notifies the NF Managers at both the node with the active NFs of the chain (Primary) and the predecessor node serving the NF chain. The node with the Active NFs and the predecessor node monitor the liveness status using BFD (more detail in §III-D). If an alternate route to the primary server exists after a link failure (*i.e.,* an alternate output port has been configured by SDN controller), the predecessor node simply redirects the traffic. If a link or node failure makes the primary unreachable, the predecessor node initiates the replay mode on the designated standby/backup node.

Chain-wide state checkpointing: REINFORCE relies on five key concepts, i.e., i) Packet logging with timestamps, ii) Pipelined replication, iii) Latch buffers for external synchrony, iv) Atomic state updates and v) Replay-based recovery to assure consistent and efficient failure resiliency of chains replicated to a secondary host. We describe these now.

1) **Packet logging with logical time stamping**: In REINFORCE, all the incoming packets at the first predecessor node are appended with a logical timestamp (*e.g.,* simple 64 bit packet counter).[1] All the outgoing packets are logged in per-port rotating log buffers at each of the predecessors. The packet log at the predecessor node is used to replay packets to the standby node when an active node fails. At the active NFV node, the timestamped value of each packet is used to track the packet-processing progress for a flow. This information is maintained in a Transmit Timestamp (*TxTs*) table replicated across the primary and backup nodes. The buffered packets are flushed upon notification by the active node's NF Manager (after being successfully sent out).
2) **Pipelined replication**: Our remote replication scheme simplifies consistency and improves performance by leveraging the local checkpoints that we already provide for software failures on the primary host. The local replicas have their state updated at the end of each batch of packets, as described in Section III-B, which gives a consistent version of the state that can be copied to the remote server without any need to pause the primary replica. As discussed previously, an important feature is that REINFORCE differentiates between deterministic and non-deterministic updates to either NF state or packet data. Deterministic updates can be recovered via replay on the remote host, so state checkpoints can be replicated in a lazy fashion to reduce overhead. On the other hand, non-deterministic state updates cannot be replayed, so packet batches with any

[1]A single non-decreasing counter is sufficient on the Logger to provide monotonic per-flow counters on the primary node

non-determinism need to have a checkpoint replicated to the backup before they are released from the primary. Fortunately, this replication can be parallelized. There are two possible ways. First, it can be performed concurrently with subsequent packet processing in the remainder of the chain. Second, as shown in Figure 1(c), an NF can continue to speculatively execute batches of deterministic packets as the checkpoint completes. It will only stall its processing if a second batch with a non-deterministic packet is executed before the checkpoint completes. To maintain packet ordering, deterministic packets that are processed after the non-deterministic packets are also not released until the non-deterministic packets are released. This gives us the ability to continue making progress subsequent to a non-deterministic packet processing as long as no other non-deterministic packet processing occurs before state corresponding to the first non-deterministic packet processing is replicated. Otherwise, we have to stall packet processing to ensure correct recovery of the state at the remote replica in the event of failures while the state is being checkpointed and copied to the replica.

3) **Transmit latch buffers**: In order to provide consistent state update with external synchrony, packets processed on the primary node are buffered (not released) until sufficient state related to packets has been replicated to the backup. To this end, we employ latch buffers at the end of service chain to store the processed packets. If all packets processed within a batch are deterministic (which is often the case), then they can be released more quickly since the standby must only be updated with the TxTs table in order to know which packets must be replayed in the event of a failure. Once a TxTs table 'commit' acknowledgment arrives from the standby's RSync component indicating the timestamps for deterministic packet batches are recorded, packets are released to downstream external nodes. On the other hand, replay is unsafe for packets with ND, so REINFORCE proactively pushes checkpoints for any batch that contains non-determinism.
4) **Atomic State Updates**: REINFORCE follows a 2-phase commit protocol to provide atomicity between state updates at the backup and packets released at the primary. Our commit protocol begins when the primary sends its updated Transmit Timestamp counters and any necessary non-deterministic state updates. The secondary associates the logical clock values (flow-specific Transmit Timestamps) with the arriving checkpoint state, and ensures that both of these are fully received for all NFs in the chain before acknowledging back to the primary. Then, the primary can release packets in its Latch Buffer to be transmitted towards their destination. The primary then notifies the secondary, so that the latter can commit the checkpoint state. State updates resulting from deterministic operations are transmitted periodically; once this state has been received, the predecessor node can be notified to clear its input log.
5) **Replay:** The use of latch buffers and atomic state updates guarantees "external synchrony," i.e., the state maintained at the standby can be made consistent with the output packets released from the primary server. Note: since deterministic application-level state is only replicated periodically, it is possible for the standby to recover to a state where the TxTs table says that some packets have been released, but the standby's state does not yet reflect the deterministic updates they should have caused. Thus, upon a failure, the standby NF chain must rollback to the last checkpoint and replay any subsequent packets so that the standby's state matches the external view of the system (outside the chain) irrespective of the failure. However, since a chain has multiple NFs and their state updates may arrive at different times, it is possible for a packet to be replayed through some NFs which have already processed it. We believe that NFs are already designed to be robust to receiving duplicate packets–duplicate transmissions are not uncommon in networks, and thus this does not require special handling. The exception to this is processing packets involving non-determinism, which is why we ensure tight state consistency for them–such packets are only released once their state has been confirmed by the standby, avoiding replay.

D. Failure Detection

NF Instance Failure Detection: NF Managers are responsible to track the liveness of all provisioned NF instances. The NF manager detects NF instance failures in two ways. First, it captures 'voluntary' NF instance failures, by registering for event notification and messages that are triggered via OS (Linux) signals and NF instance-specific messages, when any catchable exception occurs at an NF instance. Second, for involuntary NF terminations, the NF manager performs periodic (every 100 μseconds) checks via the `kill(nf_pid,0)` signal to check and deduce the status of all the registered active NFs. This operation is carried out by the NF Manager's "Monitor thread" which is also responsible for other tasks such as NF registration, de-registration and logging of statistics. The 100 μseconds probe interval is system configurable and can be tuned with a REINFORCE macro at the time of compilation. Even at this frequency, the CPU overhead is less than 1%, to track the liveness of 64 NFs.

Link and Node Failure Detection: We improve BFD [19] and adapt its parameters for both link and host failure detection. For more details refer to our work [23]. (refer §3.4.)

E. Chain-Wide Correctness

The goal of REINFORCE is to ensure external synchrony and enforce chain-wide correctness. Here we briefly describe how correctness is ensured and provide the formal proof of REINFORCE in §VII.

We consider two distinct modes of operation to ensure remote state consistency. For *deterministic packet processing*, packets from the primary are released only after committing updates of packet-processing progress to the standby's Transmit Timestamps (TxTs table). However, the remote standby's NF chain state can be out-of-sync and lag the primary state. Upon failure, replaying of the packets from predecessor node makes the standby NFs' state roll-forward and be synchronized to what was in the primary before the failure. All the NFs process the packets to update their state. Based on the committed TxTs table, the NF Manager on the standby

discards duplicate packets that have already been sent. Thus, the standby node's NFs are synchronized to the external view of the state, which was that of the primary at the time of the failure.

If a batch of packets results in *non-deterministic processing*, then the packets from primary are released only when both the TxTs table and NF state checkpoint are committed to the remote standby for the entire chain. This ensures the external view to be in sync with the state at *both* the primary and standby nodes across the NF chain. Note that when *any* NF in the chain results in non-deterministic state updates, the state for the entire NF chain is committed before releasing the packets. Thus standby NFs are always in sync with the primary for released packets.

IV. Implementation

REINFORCE (source [24]) is built on OpenNetVM [25], a DPDK [15] based NFV platform that enables to run the NFs in containers or as separate processes. We implemented the following modules i) Packet logging: to add a logical timestamp to all input packets and to log all outgoing packets to a stable store; ii) RSync: to enforce external synchrony and perform the two-phase commit transaction using multiple latch buffers; and iii) Liveness Monitoring: to monitor the liveness of locally provisioned NFs and BFD sessions across the configured links. These functions use 1 CPU core each.

The control framework coordinates failover via pause and resume event notifications to active and standby NFs, and performs failover actions for remote link failures. To account for the transmit state timestamp of each flow and enforce 2 phase commit transactions, packets leaving the last NF in a chain are stored in latch buffers in the RSync component before being released to the DPDK NIC ports.

A. Remote Failover

Atomic Two-Phase Commit Transaction: We use a simple UDP transport to deliver updates to the backup and use sequence numbers to identify any missing packets (reliable transport like TCP can also be used). We use a custom Ethernet type to differentiate state update packets from regular NF destined packets. If packets are lost, we abort the transaction and resend new updates. State transfer information is included in the header fields to indicate the type of packet transferred (either state transfer or acknowledgement packet), type of state (NF state, service configuration information, or Tx Timestamp), packet size, base offset address, packet sequence number, and 'last packet' flags.

Accounting for Failed Transactions: When the acknowledgement for Tx state update commit is not delivered to primary, the NF manager may get blocked resulting in port buffers getting full and subsequent processing by NFs being discarded. To avoid this, we have a transaction timeout after which the NF manager aborts the current transaction and continues to process subsequent packets and send new updates. To ensure continuity, we choose to continue processing subsequent packets and drop the packets corresponding to the failed transaction. End-to-end retransmission of these dropped packets is expected to update the standby NF state appropriately.

Tx Timestamp State Update Overhead: We opportunistically perform the transmit timestamp (TxTs) table state updates as often as possible. The frequency of operation is limited by the RTT and number of configured latch buffers on the system. Assuming a best case RTT (between two directly connected nodes) of 100μ seconds, performing each Tx timestamp checkpoint in the worst case needs to transfer the entire TxTs table of 64KB (64 1KB packets). This is an overhead of less than 5.25% on the 10Gbps link. For checkpointing, using large latch buffers (8K), we can checkpoint at a slow rate (roughly once every 5 RTTs) *i.e.,* performing checkpoints once every 500μ seconds, reducing the 10Gbps link overhead to less than 1.05% at the cost of added latency.

TABLE I
NFs and NF Chains Used in Experiments

NF	Type	Characteristic	
		Shared Memory	Non Determinism
Simple Forward (SF)	Stateless	-	-
Basic Monitor (MON)	Stateful	64 KB	5 SV, 1 ms
Vlan Tag(QoS)	Stateful	64 KB	3 SV, 1 ms
Load Balancer (LB)	Stateful	64 KB	1 SV, 1 ms
DPI	Stateful	4 MB	2 SV, 1 ms
Chain 1	Stateful	QoS, BM	
Chain 2a	Stateful	QoS, BM, LB	
Chain 2b	Stateful	QoS, BM, SF	
Chain 3	Stateful	QoS, BM, SF1, SF2	

Shared variables(SV) and rate of non-deterministic updates.

V. Evaluation

Our experimental testbed used five Intel(R) Xeon(R) CPU E5-2697 v3 @ 2.60GHz servers, each with 157GB RAM, two sockets with 28 cores each, running Ubuntu SMP Linux kernel 3.19.0-39-lowlatency. The topology for the primary, standby and predecessor nodes is as shown in Figure 3. In addition, we have a source and sink node at the two ends. Table I lists the NFs and NF chain used in evaluations. For the NF chain scenario, we deploy the entire NF chain on single (primary) node.[2] For these experiments, nodes were connected back-to-back with dual-port 10Gbps DPDK compatible NICs. We use the DPDK-based high speed traffic generator, Moongen [26] to generate line rate traffic consisting of UDP and TCP packets, apache-bench [27], and wrk [28] to flood HTTP download requests. We vary the number of flows and the NF chain setup as needed for each experiment. We configured the number of stages of latch buffers (*i.e.,* multiple transaction buffers) to 3, with each stage having 4K packet buffers.

A. Overhead Analysis

Our profiling indicates the cost of memory scan and updating of the dirty state for a 64KB memory and 1KB chunks to be 55-80 CPU ticks; and the copy overhead for a 4KB page is CPU ticks. Copy operation for a batch of processed packets drastically reduces the overhead during normal processing. Next, we consider DPI, a compute-intensive NF, and lightweight monitor (MON) NFs, and subject them to line rate traffic for different packet sizes. We observe from Figure 4 that REINFORCE is able to achieve performance identical to baseline for all DPI. For MON, even the worst case performance impact is less than 15% (12.6 Mpps with REINFORCE compared to 14.88Mpps baseline).

[2]We observed similar results with the NF chain setup across two nodes.

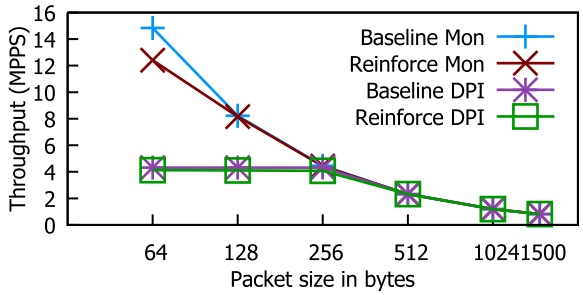


Fig. 4. Throughput of different packet sizes.

TABLE II
PERFORMANCE IMPACT OF NON-DETERMINISM

Time (ns)	1	10^3	10^5	250us	500us	10^6	10^7	10^8	10^9
Tput (Mpps)	0.22	0.22	0.24	7.03	11.19	12.65	13.34	13.34	13.34
Latency (μs)	1370	1370	1370	1361	790	698	670	617	617

Impact of Non-Determinism Rate: REINFORCE performs a 2-phase commit of the chain-wide packet-processing progress and NF states. To ensure correctness, any non-deterministic updates result in chain-wide NF state checkpointing. Table II shows the impact on performance (throughput and maximum latency) for different ND rates, which is varied from 1 pkt. every nano second up to one pkt. every second. REINFORCE is able to provide near line-rate processing for the non-deterministic rate that is less frequent than one every $250\mu sec$, while more frequent non-determinism reduces the throughput, eventually dropping to 0.22Mpps. This is due to the round-trip latency of 2-phase commit, causing the NF processing to stall if a previous non-deterministic batch of packets has to be processed and state committed to the standby.

B. Operational Correctness and Performance

We assess the application level of failover through the following end-to-end tests. *i) DPI based protocol detection:* We feed a set of PCAP traces (MPEG, Hangout, Youtubeupload, Snapchat, QUIC) available at NTOP [29], [30] to the DPI NF. We observe that the DPI NF identifies the protocols correctly both when there are no failures and when the primary fails and REINFORCE fails over to the standby NF.

ii) HTTP Downloads: We route HTTP downloads through a service chain of 2 NFs (QoS and MON). We start repeated HTTP download requests for a period of 60 seconds, and trigger failures at the 30 second mark. a) We induce a QoS NF instance failure to account for local NF failover, and b) we induce, NF MGR failure on the primary node to trigger a chain-wide remote failover. We compare the *baseline* operation *i.e.,* failure-free operation with both NF instance failure (local failover) and node failure (remote failover) cases. We observe with REINFORCE that HTTP downloads succeed for both the NF instance failure (local failover) and node-failure (remote failover) cases. We also observe very little impact on the application, resulting in just 3-4% reduction in the total number of requests serviced per second, and a negligible reduction in throughput as shown in Table III. Note: In Resiliency mode, with no failures, we observed similar results as reported in Baseline w/o failure in Table III.

Failover Times: We measured the time for local and remote failovers from the instant we induce a failure. For local failover: mean $= 56\mu$s and maximum $= 114\mu$s over 100 iterations. For remote failover: mean $= 3280\mu$s and maximum $= 3517\mu$s over 10 iterations. This includes failure detection time with BFD and for the predecessor node to initiate the failover at the backup by starting replay of buffered packets. We do not account for the time needed to complete the replay and send the first new packet, as it varies based on the processing chain and non-determinism intervals. But, we did measure the time needed to initiate and prepare for replay at the predecessor node (*i.e.,* to notify the standby node, open a pcap file and start replay). The average time taken was 60-100μs, and just below 2ms to replay approximately 3K packets from the predecessor node to the standby. Replay execution of the 3K packets (20Mbits) on the standby took approximately 2ms.

TABLE III
EFFECT OF FAILURE ON HTTP DOWNLOADS

	Baseline	Resiliency	
	w/o failure	Local Failover	Remote Failover
Requests/sec	10.52	10.32	10.22
Transfer/sec (GB)	1.08	1.06	1.02

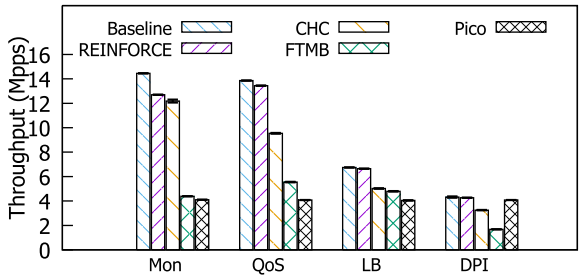


Fig. 5. Performance impact of different FT systems on the normal operation for different NFs.

C. Failure-Free Operation

We compare REINFORCE with CHC [11], FTMB [8] and Pico Replication [9]. Note: We implement: a) simplified version of CHC, where the NF state is externalized to a Redis datastore [31]. The NFs cache the state locally and perform asynchronous state update operations after processing a batch of 256 packets[3] ; b) simplified FTMB logic with parallel releases, by storing packet access logs (PALs) per shared variable *i.e.,* NFs transmit all the associated PALs before releasing packets to the NF manager; the NF manager simply transmits the packets without blocking for output commit; and c) Pico Replication: NF state checkpointing with output commit policy. For a number of different NFs, we compare: i) the overhead during normal operation by measuring the throughput; and ii) additional latency of packet processing (for state update operation), for individual NF instances. Figure 5 shows throughput for normal operation, in Mpps (with error bars showing the standard deviation in throughput). REINFORCE performs almost as well as baseline (no resiliency case), achieving near line rate ($\sim$13.5Mpps) throughput for most NFs. REINFORCE's remote replication outperforms Pico

[3] All our experiments with REINFORCE use a small batch size of 32 packets. However for CHC, we set batch size to 256 packets, as the smaller batch (32) limited CHC's throughput to less than 3Mpps. Note: Our results are better than those presented in the CHC paper [11], and the results may depend on the actual CHC implementation and its optimized datastore.

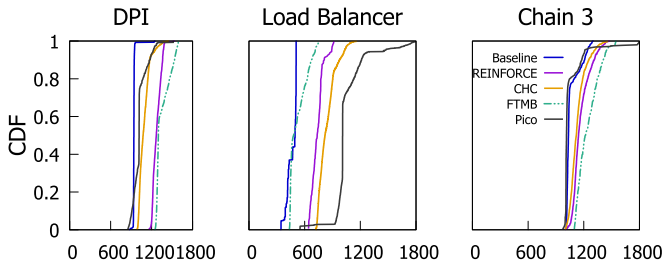


Fig. 6. RTT with different FT Systems.

TABLE IV

MULTITENANCY AND RESILIENCY MODES. (a) LATENCY (μ) WITH DIFFERENT RESILIENCY MODES. (b) RATE OF ND FOR NF1 AND NF2 AT DIFFERENT TIME INTERVALS

Mode	Min	Median	Max
Simple Forward NF			
Baseline	210	221	373
Local Replication	211	229	402
Remote Replication	540	601	789
Basic Monitor NF			
Baseline	268	334	695
Local Replication	270	338	699
Remote Replication	596	623	840

(a)

Time (sec)	NF1 (μs)	NF2 (μs)
1-10	1	1000
11-20	100	750
20-30	250	500
30-40	500	500
40-50	750	250
50-60	1000	100
60-70	1000	1000

(b)

replication by 2 orders of magnitude. Fig. 6 shows the impact on packet latency for two selected NFs as well as for Chain3. The median and 99%ile latency for REINFORCE and CHC are comparable, and is better than Pico and FTMB. Also, both REINFORCE and CHC have comparable throughput for a lightweight Monitor NF. But, for heavier-weight NFs, REINFORCE outperforms all the alternatives. In particular, the throughput of CHC drops for heavy-weight NFs and for a NF chain, likely due to the overhead of frequent context switches from user-space to kernel space. This overhead is avoided for the user-space DPDK NFs in REINFORCE.

D. Multi-Tenancy and Resiliency Levels

We demonstrate the benefits of REINFORCE in supporting multi-tenant NF execution and in providing performance isolation for flows configured with different resiliency levels.

1) Multi-Tenancy: In a typical multi-tenant environment, network functions from different tenants can be co-located and share the same CPU. We expect that NFs from different tenants see different workloads and are consequently subject to different rates of non-deterministic at different time intervals. When the non-deterministic rate is high, the CPU may be idle/underutilized because of frequent stalls. But, REINFORCE takes advantage of efficiently multiplexing NFs that exhibit non-determinism to improve CPU utilization and overall system throughput. We multiplex 2 NFs, NF1 and NF2, from different tenants on the same core. They exhibit different non-deterministic rates at different time intervals as shown in Table IVb. Figure 7 shows the ability of REINFORCE to efficiently multiplex these 2 NFs, resulting in the improvement in both the CPU utilization and aggregate throughput.

2) Differing Resiliency Levels: We demonstrate the benefit of REINFORCE's ability to support different flows configured with different resiliency levels. REINFORCE provides the desired resiliency while isolating the flows from each other. We have two Monitor NF instances, configured as: (a) one NF instance with only local resiliency (backup on the same node) for flow-1; and (b) a second NF instance with node-level resiliency (remote standby) for flow-2. We also perform

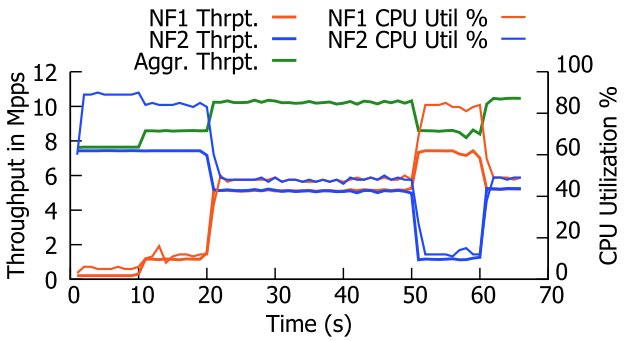


Fig. 7. Two isolated NFs with varying Non-Determinism rates, running on the same CPU core.

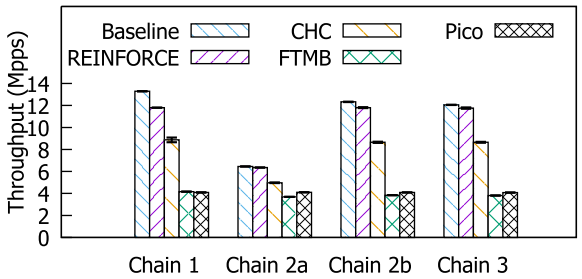


Fig. 8. Different NF Chain processing performance.

a similar experiment with the Simple Forward NF as well. The latencies for the two flows differ, as shown in Table IVa. We observe the impact on latency is minimal for the flows configured with only local resiliency (less than 30ms in the worst case). But, flows configured with remote resiliency (local + remote) incur nearly 2x higher latency for both the Simple Forward and Basic Monitor cases. This shows the ability of REINFORCE in providing different levels of resiliency while isolating one flow from another – an essential and desirable characteristic for multi-tenancy.

E. Impact of Chain Length

We consider experiments with multiple chains having different lengths as described in Table I. We compare the following cases: baseline (no resiliency), REINFORCE, CHC, FTMB, and Pico Replication[4]. Figure 8 shows that performance of REINFORCE remains consistent with the baseline for varying chain lengths, unlike FTMB and Pico. In fact, with increased chain lengths, the overheads of REINFORCE are amortized allowing the throughput to be closer to the baseline. With FTMB and Pico, to ensure correctness, the output commit must be performed individually for each NF in the chain. In fact, the throughput we show for FTMB and Pico is likely to be optimistically high compared to a full implementation.

VI. RELATED WORK

NF state migration: Split/Merge [32] defines state access APIs to read and update the internal state of virtualized NFs being moved across hosts. It relies on the ability to identify per-flow state to provide consistent migration. Stratos [33] provides an orchestration layer for NFs by using an SDN

[4]For both FTMB and Pico Replication, we implement the state replication only at the end of the chain, rather than at every NF. Also, with CHC, we do not implement the version control for the state variables. This serves as a simplified, optimized approach, but does not ensure correctness.

controller to migrate the instances and redistribute the traffic to less congested nodes. Likewise, we take advantage of SDN controller to set up the forwarding rules but rely on NF Managers to efficiently migrate the NF instances. OpenNF [10] presents a control plane architecture to have loss-free transfer of NF state. Unlike OpenNF's controller-based orchestration and event buffering mechanism, REINFORCE relies on the NF manager to perform state migration across the designated active and standby NF nodes while allowing NFs to simultaneously process packets. This results in lower latency and less performance impact on normal processing. Unlike [32], REINFORCE does not require the NFs to depend on specific state update APIs, but only requires the NFs to annotate the state updates sufficiently to distinguish between deterministic and non-deterministic changes. StateAlyzr [34] complements our work by enabling the NF developers to analyze programmatically and to identify just the right amount (minimal) of NF state, and correctly annotate the NF state that needs to be migrated to ensure consistent state replication. S6 [35] provides a framework for elastic scaling of NFs. It implements NF state as distributed shared state objects, and supports object replication to facilitate NF state sharing across multiple NF instances.

Fault tolerance and high availability: Pico Replication [9] is an application level NF state checkpointing based high availability framework built on top of Split/Merge. It provides fine-grained flow level state replication and employs flow group-based NF state transfers. To enforce correctness, it buffers all output packets during NF state checkpointing, thus delaying outputs even during failure-free operation. FTMB [8] is a replay based framework that logs all input packets and the per packet access log of all the components (*i.e.,* the shared variables in NF that account for non-determinism) that are necessary to restore the state on the replica during replay. In addition, to amortize the cost of input logging, it also employs periodic check-pointing of NFs. Thus, FTMB guarantees correctness of operation during replay mode by ensuring strict ordering of packet processing (guided by the packet access logs) at the replay node. In essence, FTMB's notion of correctness emulates strict idempotent per packet behavior across the active and replica nodes. This comes at the cost of maintaining multiple per packet access logs, which becomes a potential bottleneck for NFs with 5+ shared variables, for packet rates of 1.25Mpps (ref. §5 of [8]), resulting in more than ∼30% overhead traffic. In addition, due to periodic VM checkpointing, the tail latencies drastically increase from less than 100μs, at the 50th%-ile, to nearly 810 μs, at 95th%-ile and 18ms at 99th%-ile. Also, both of these works don't account for chains of network functions. REINFORCE fills this gap with an efficient chain level replication mechanism that does not excessively impact (add latency) normal operation, nor does it place limits on processing rates to enforce correctness of the replica state. Plover [36] presents a virtualized state machine replication system to address general VM fault tolerance. By enforcing the same total order of inputs for a VM replicated across hosts, it can keep most memory pages updated and only transfer the few divergent pages between primary and secondary. This is effective in alleviating checkpointing overhead, while maintaining external consistency.

Alternative architectures: StatelessNF [37] and CHC [11] are alternative approaches that externalize NF state to in-memory databases like Redis [31]. Such an approach intends to enable scaling of NF instances and failover to different NF instances without replicating the NF state to a distinct apriori selected replica. Additionally, it does not need to distinguish and treat deterministic vs. non-deterministic state updates differently. However, a major challenge arises especially with cached state and asynchronous updates in ensuring the correctness and consistency of the externalized state w.r.t. the failed NFs that might have partially processed the packets and updated the states, before crashing. In such a scenario with CHC [11], both the NFs and the database need to maintain additional version control for each state update variable, so that the state updates can be validated before commit, which might further impact the NF throughput. Architecturally and conceptually having externalized state for NFs may be well-suited for micro-services, but the challenges highlighted above in terms of performance and complexity of operation require careful consideration of its applicability for high performance NFs.

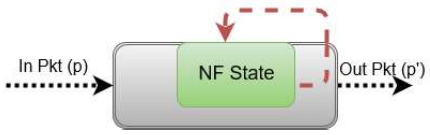


Fig. 9. NF Packet processing and state machine abstraction.

VII. Proof of Correctness of REINFORCE

A. NF Packet Processing Model

We represent NFs as finite state machines that process a stream of incoming packets p_{ij} (*i.e.* i^{th} *packet of* j^{th} *flow*). When the NFs process packets, they update and transition their internal state and output one or more packets p'_{ij}, as shown in Fig. 9. For the NF Chain, packets are sequentially processed by distinct NFs in the chain resulting in state update(s) at each of the NFs. Packets are then output from the chain.

B. Definitions

Based on previous literature [8], [14], we provide some of the basic definitions for NF packet processing (deterministic and non-deterministic) and external synchrony.

Definition 1 (Deterministic Processing): In a given NF state, processing a packet p_i always results in same deterministic state transition $P * S_i \mapsto S_j$ and deterministic output p'_i.

Definition 2 (Non-Deterministic Processing): In a given NF state, processing a packet p_i each time may result in different a state transition, $P * S_i \mapsto \{S_i\}$ and yield different a output, $\{p'_i\} \subseteq (p'_0, p'_1, p'_2, \ldots, p'_i)$.

Definition 3 (External Synchrony): NF state synchrony is defined by the externally observable behavior and not by the actual internal system (NF or Chain of NFs) state. On failover, the externally observable state remains consistent and independent of the actual NF state across the replicas.

C. Operational Correctness and Proof

Theorem 1 (Correctness of Operation): For remote replication, REINFORCE preserves external synchrony and ensures correctness of operation for NF chains.

TABLE V
NOTATIONS USED FOR CORRECTNESS ANALYSIS

$\mathbb{B}$	Set of buffered packets at the predecessor node $= (p_0, p_1, p_2, \ldots, p_x)$
$\mathbb{P}_i$	Set of processed packets at the Primary NF (i) $= (p'_0, p'_1, p'_2, \ldots, p'_n) \ni \mathbb{P}_i \subseteq \mathbb{B}$
$\mathbb{S}_i$	Set of abstract states of NF (i) $= (S_0, S_1, S_2, \ldots, S_x)$
$\mathcal{T}_j$	Timestamp of last packet committed for flows(j). $= (t_{ij}) \ni {}'_i forflow(j)$
$\mathbb{R}$	Packets released by the Primary node. $= (p'_0, p'_1, p'_2, \ldots, p'_m) \ni \mathbb{R}_i \subseteq \mathbb{P}_i \subseteq \mathbb{B}$
$\mathcal{P}.\mathcal{V}_i$	State of Primary NF (i) $= \mathbb{P}_i * S_i \mapsto S'_i$
$\mathcal{P}.\mathcal{V}$	States of all Primary the NFs. *i.e.,* Primary View (PV) $= (\mathcal{P}.\mathcal{V}_0, \mathcal{P}.\mathcal{V}_1, \mathcal{P}.\mathcal{V}_2, \ldots, \mathcal{P}.\mathcal{V}_i)$
$\mathcal{E}.\mathcal{V}$	Observed states of all Primary NFs. *i.e.,* External View (EV) $= (\mathcal{E}.\mathcal{V}_0, \mathcal{E}.\mathcal{V}_1, \mathcal{E}.\mathcal{V}_2, \ldots, \mathcal{E}.\mathcal{V}_i)$
$\mathcal{S}.\mathcal{V}_i$	State committed at Secondary NF(i). $= \mathcal{S}.\mathcal{V}i \subseteq \mathcal{P}.\mathcal{V}_i$; and non-committed Primary state $= \overline{\mathcal{P}.\mathcal{V}_i}$
$\mathcal{S}.\mathcal{V}$	States of all Secondary NFs. *i.e.,* Secondary View (SV). $= (\mathcal{S}.\mathcal{V}_0, \mathcal{S}.\mathcal{V}_1, \mathcal{S}.\mathcal{V}_2, \ldots, \mathcal{S}.\mathcal{V}_i)$

We prove Theorem 1 by the methods of "Proof by case" and "proof by contradiction". We know that the processing of a packet by an NF can result in either non-deterministic or deterministic state updates in an NF. Accordingly, in REINFORCE the state replication is based on the following two propositions:

Proposition 1 (Packet Processing Progress): To preserve external synchrony, in the case of only deterministic state updates, it is sufficient to track and update the packet processing progress information across the NF chain.
This implies that REINFORCE only updates the packet processing progress information $\mathcal{T}_j$ before releasing the packets that update the external view and then lazily (periodically) updates the NF state to the replica node later, as shown in the right side of the Figure 10. Hence,

$$\mathcal{S}.\mathcal{V}i \subseteq \mathcal{E}.\mathcal{V}_i \ni \mathcal{E}.\mathcal{V}i \subseteq \mathcal{P}.\mathcal{V}_i and \mathbb{P}_i \subseteq \mathbb{B} \qquad \text{(VII.1)}$$

Proposition 2 (External Synchrony with ND): In order to preserve, external synchrony in the event of non-deterministic packet processing, it is necessary to synchronize and commit the NF state at replica $\mathcal{P}.\mathcal{V}_i$ before releasing the packet p'_i.
This implies that REINFORCE first updates the NF state to the replica node and only then releases the packets. Hence as shown in the left side of the Figure 10,

$$\mathcal{E}.\mathcal{V}i = \mathcal{S}.\mathcal{V}_i \ni \mathcal{S}.\mathcal{V}i \subseteq \mathcal{P}.\mathcal{V}_i \qquad \text{(VII.2)}$$

Case 1: Let us consider the case when the primary node fails, given that packet processing (initial or after the last packet processing progress commits) update $\mathcal{T}_j$ resulted in only deterministic state updates in any of the NFs in the chain. In such case, by Definition 1, reprocessing of any $t_{ij} \in \mathcal{T}$ packets in $\mathbb{B}$ only updates the NF state at the secondary NF, but these packets are subsequently dropped by the replica node and hence do not modify the external view, $\mathcal{E}.\mathcal{V}$. However, when the secondary NF(i) processes the packets, this implies: $\mathcal{S}.\mathcal{V}_i = \mathbb{P}_i * S_i \mapsto S'_i$ resulting in $\mathcal{S}.\mathcal{V}_i = \mathcal{P}.\mathcal{V}_i$ If we consider the contradiction that $\mathcal{S}.\mathcal{V}_i \neq \mathcal{P}.\mathcal{V}_i$, then that violates Definition 1, and thus cannot be true.

Moreover, for the remaining $[\mathbb{B} - \{\mathcal{T}_i j\}]$ packets, the $\mathcal{P}.\mathcal{V} = \emptyset$ and as per Proposition 1 $\mathcal{E}.\mathcal{V} \subseteq \mathcal{P}.\mathcal{V} = \emptyset$. Hence, in either case, external synchrony $\mathcal{E}.\mathcal{V}$ is preserved.

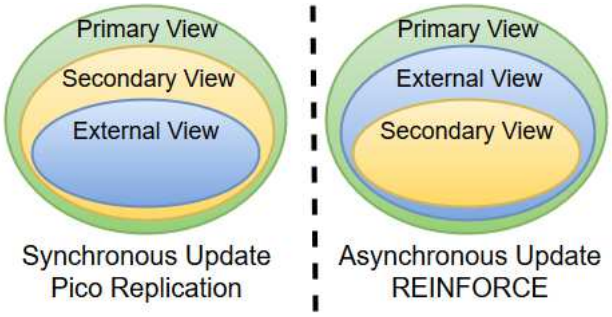


Fig. 10. Relationship of NF States across Primary (PV), Secondary/Replica (SV), and External view (EV).

Case 2: Consider the case when the primary node fails given that packet processing (initial or after last packet processing progress commit) updates timestamp $\mathcal{T}_j$ results in ND at any NF in the chain. Then, the initial condition (at the time of failure) is: $\mathcal{S}.\mathcal{V} \subseteq \mathcal{E}.\mathcal{V} \subseteq \mathcal{P}.\mathcal{V}$. By Proposition 2 for the replay condition, we have, $\mathcal{E}.\mathcal{V} \subset \mathcal{P}.\mathcal{V} \ni \mathcal{E}.\mathcal{V} \bigcap \overline{\mathcal{P}.\mathcal{V}_i} = \emptyset$, *i.e.,* External view does not contain any ND updates of the Primary NFs which are not synced to secondary. Further, by Definition 1, reprocessing of any packet in $\mathbb{B}$ with timestamp $t_{ij} \in \mathcal{T}$ only updates the NF state at the secondary NF, but the packets $\mathbb{P}_i < \mathcal{T}$ are subsequently dropped by the replica node and hence do not modify $\mathcal{E}.\mathcal{V}$. However, as the secondary NF(i) processes these replayed packets, their state rolls forward to the external state, *i.e.,* $\mathcal{S}.\mathcal{V}_i = \mathbb{P}_i * S_i \mapsto S'_i$ resulting in $\mathcal{S}.\mathcal{V}_i = \mathcal{E}.\mathcal{V}_i$ Hence, the final status is: $\mathcal{S}.\mathcal{V} = \mathcal{E}.\mathcal{V} \ni \mathcal{S}.\mathcal{V} \neq \mathcal{P}.\mathcal{V}$ Thus, external synchrony is preserved, but the Primary (failed) and Secondary NFs may differ in internal states due to possible different outcomes, because of ND.

Case 3: Let us consider the case when the packet processing results in both deterministic and non-deterministic updates in any of the NF(s) in the chain. Here, $\mathbb{B}$ may contain packets that result in ND. For $\mathcal{P}_i \in \mathbb{B}$:

$$\text{if } \mathcal{P}_i <= \mathcal{T} \implies \text{Duplicate packet}$$
$$\text{else } \mathcal{P}_i > \mathcal{T} \implies \text{New packet}$$

This is equivalent to Case 1 for deterministic processing and Case 2 for ND updates. For $\mathcal{P}_i <= \mathcal{T}$ *i.e.,* duplicate packet processing $\mathcal{S}.\mathcal{V}_i = \mathbb{P}_i * S_i \mapsto S'_i$ results in Secondary state update $\mathcal{S}.\mathcal{V}_i = \mathcal{E}.\mathcal{V}_i$ for all deterministic packets. However, if $\mathcal{P}_i$ results in ND, then by Proposition 2 it must already be committed to secondary. Otherwise, the $\mathcal{P}_i > \mathcal{T}$ must be a new packet which updates $\mathcal{S}.\mathcal{V}_i$ and also $\mathcal{E}.\mathcal{V}_i$

Note: As, we commit state for the entire NF chain *i.e.,* all the NFs in the chain get to commit the NF state during the periodic state transfer or in the event of any non-determinism that necessitates explicit state update for any NF in the chain. Hence, with duplicate packet processing, only the secondary gets updated without impacting the EV, while the new packets result in updates to both SV and EV.

VIII. REINFORCE WORK-FLOW

Figure 11 illustrates the workflow for updating and maintaining both the local and remote replicas for the two failover schemes. To summarize, we expect an SDN controller and the NFV orchestrator to configure the resiliency mode (local only or remote), BFD, and the role setting for the predecessor, active, and standby nodes for the NF chain.

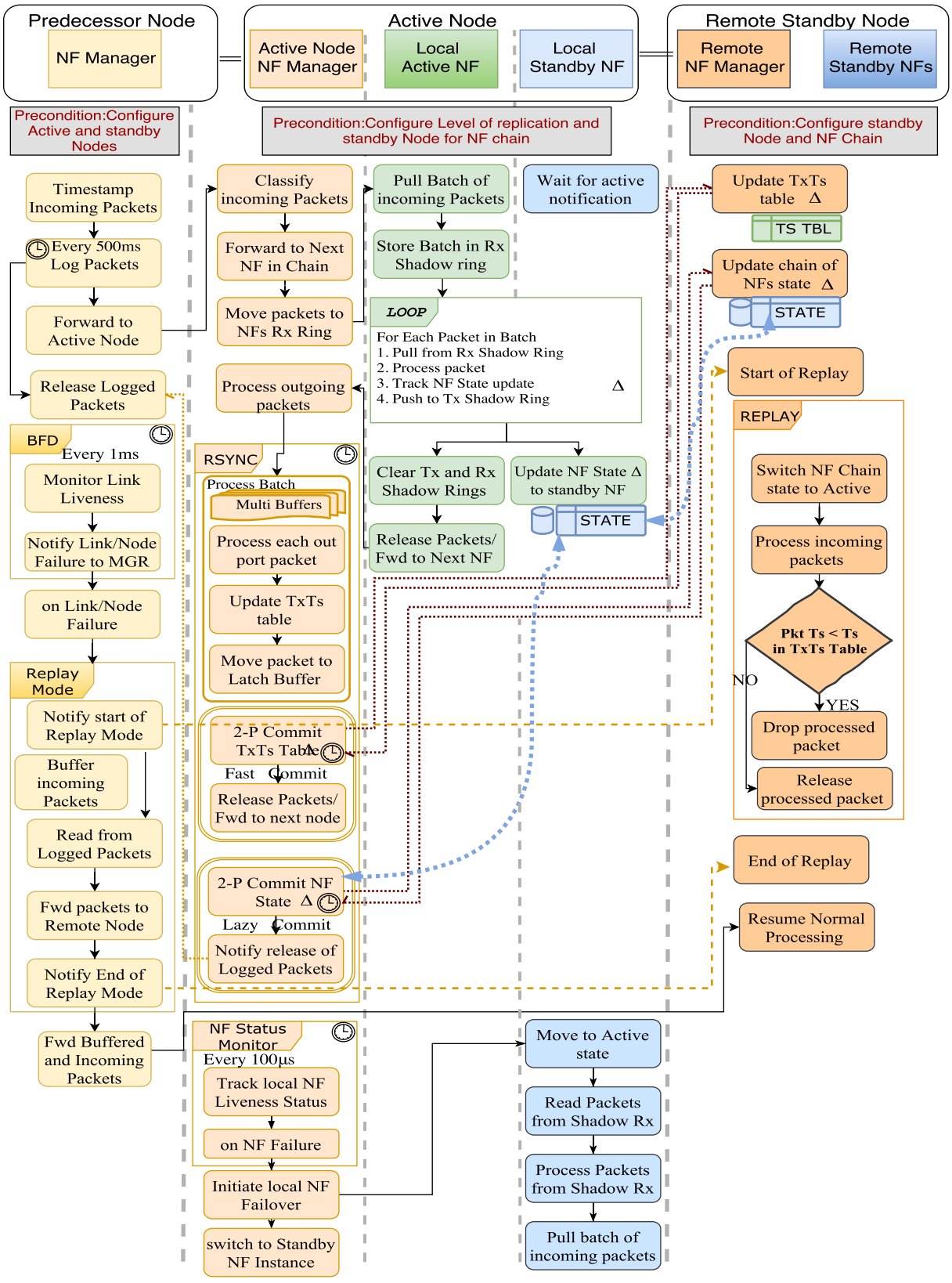


Fig. 11. Work-flow for local and remote NF replication and failover.

Predecessor node: it time-stamps the incoming packets and logs all the outgoing packets in a per-port rotating log buffer before forwarding the packets to the active node. The rotating log buffers are recycled periodically. Additionally, it monitors the liveness of the link through BFD and performs replay in the event an active node failures. To initiate the replay, it first notifies the remote standby to take-over as the active node and then replays the packets from the rotating log buffers. Until the replay is complete, the new incoming packets are temporarily buffered, and if this buffer overflows, subsequent

incoming packets are dropped. After completion of replay, it notifies the standby node to transition to normal (as new primary) processing mode and thereafter transmits interim logged packets and subsequent incoming packets.

Active node: processes incoming packets across the NF chains. For each NF a local standby NF is also instantiated. After processing a batch of packets, the state is synchronized with the local standby. When an NF chain spans across multiple nodes, all nodes except the last active node also act as the predecessor node and log outgoing packets. At the end of NF chain processing, it updates the TxTs table and initiates the 2-phase commit for the updated TxTs table. Upon successful commit, the packets are released; otherwise they are dropped. Also, periodically the NF state updates are committed to the remote standby, and in the case of ND, the state commit operation is performed immediately. Note the NF state used for remote replication is from the local standby NFs. Also, the NF manger tracks the liveness of the local NFs and in the event of an NF failure, it initiates the local NF failover and transitions the standby NF to active mode.

Standby node: during the standby mode, this node processes incoming time-stamp updates and the NF chain's state-update packets and notifies the state-update response back to the active node. Upon notification by the predecessor node to transition to replay mode, the NF-manager transitions the local standby NFs to active mode and forwards the subsequent incoming packets for processing to the NFs in the chain. Until the replay completion notification, the standby node checks for the processed packet time-stamp with the last logged time-stamp in the TxTs table to determine whether the packet is to be dropped (duplicate) or not. After the completion of replay, the standby node transitions and continues to process subsequent packets as an active node.

IX. Conclusion

REINFORCE is the first to address chain-wide NF resiliency, supporting fast detection and recovery of software, server and link failures. REINFORCE can detect NF failure and failover to a local standby within 150μs. More importantly, it provides chain-wide failover to a remote node within 5ms. In addition, REINFORCE results in minimal overhead for normal operation and achieves 2X better performance than the state-of-the-art. We distinguish the minimum state information needed to achieve efficient and consistent remote replication. The key is REINFORCE's separation of deterministic and non-deterministic NF processing, and only incurring the overhead of checkpointing and a 2-phase commit of state on the standby for non-deterministic NF processing. REINFORCE automatically tracks and replicates state to standby NFs while enforcing correctness. The amount of state replicated is minimized by using 'lazy' replication of NF application state across hosts, and packet replay is used to speed up the recovery of deterministic NF processing. Even for reasonably frequent non-deterministic packet processing, REINFORCE's performance is far superior to the other alternatives.

References

[1] R. Potharaju and N. Jain, "Demystifying the dark side of the middle: A field study of middlebox failures in datacenters," in *Proc. Conf. Internet Meas. Conf.*, New York, NY, USA, 2013, pp. 9–22, doi: 10.1145/2504730.2504737.

[2] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: Measurement, analysis, and implications," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 350–361, Aug. 2011, doi: 10.1145/2043164.2018477.

[3] H. S. Gunawi *et al.*, "Why does the cloud stop computing?: Lessons from hundreds of service outages," in *Proc. 7th ACM Symp. Cloud Comput. (SoCC)*, New York, NY, USA, 2016, pp. 1–16, doi: 10.1145/2987550.2987583.

[4] S. K. Sahoo, J. Criswell, and V. Adve, "An empirical study of reported bugs in server software with implications for automated bug diagnosis," in *Proc. 32nd ACM/IEEE Int. Conf. Softw. Eng.-ICSE*, vol. 1, New York, NY, USA, 2010, pp. 485–494, doi: 10.1145/1806799.1806870.

[5] J. Martins *et al.*, "Clickos and the art of network function virtualization," in *Proc. 11th USENIX Conf. Networked Syst. Design Implement. (NSDI)*, Berkeley, CA, USA, 2014, pp. 459–473. [Online]. Available: http://dl.acm.org/citation.cfm?id=2616448.2616491

[6] A. Madhavapeddy *et al.*, "Unikernels: Library operating systems for the cloud," *SIGPLAN Not.*, vol. 48, no. 4, p. 461, Apr. 2013, doi: 10.1145/2499368.2451167.

[7] (2013). *ETSI-GS-NFV-002-Network Functions Virtualization (NFV): Architectural Framework*. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf

[8] J. Sherry *et al.*, "Rollback-recovery for middleboxes," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, pp. 227–240, Aug. 2015, doi: 10.1145/2829988.2787501.

[9] S. Rajagopalan, D. Williams, and H. Jamjoom, "Pico replication: A high availability framework for middleboxes," in *Proc. 4th Annu. Symp. Cloud Comput. (SOCC)*, New York, NY, USA, 2013, pp. 1:1–1:15, doi: 10.1145/2523616.2523635.

[10] A. Gember-Jacobson *et al.*, "Opennf: Enabling innovation in network function control," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 163–174, Aug. 2014, doi: 10.1145/2740070.2626313.

[11] J. Khalid and A. Akella, "Correctness and performance for stateful chained network functions," in *Proc. 16th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, Boston, MA, USA, Feb. 2019, pp. 501–516. [Online]. Available: https://www.usenix.org/conference/nsdi19/presentation/khalid

[12] A. Panda, O. Lahav, K. Argyraki, M. Sagiv, and S. Shenker, "Verifying isolation properties in the presence of middleboxes," 2014, *arXiv:1409.7687*. [Online]. Available: https://arxiv.org/abs/1409.7687

[13] K. Alpernas *et al.*, "Abstract interpretation of stateful networks," in *Proc. Int. Static Anal. Symp.*, 2018, pp. 86–106.

[14] E. B. Nightingale, K. Veeraraghavan, P. M. Chen, and J. Flinn, "Rethink the sync," *ACM Trans. Comput. Syst. (TOCS)*, vol. 26, no. 3, pp. 1–26, Sep. 2008, doi: 10.1145/1394441.1394442.

[15] (2014). *Data Plane Development Kit*. [Online]. Available: http://dpdk.org/

[16] (2017). *Criu: Checkpoint Restore in Userspace*. [Online]. Available: http://criu.org/

[17] P. Quinn and T. Nadeau, Eds., *Problem Statement for Serxvice Function Chaining*, document RFC 7498, Apr. 2015. [Online]. Available: https://www.rfc-editor.org/info/rfc7498, doi: 10.17487/RFC7498.

[18] E. J. Jackson *et al.*, "Softflow: A middlebox architecture for open vswitch," in *Proc. USENIX ATC*, 2016, pp. 15–28.

[19] D. Katz and D. Ward, *Bidirectional Forwarding Detection (BFD)*, document RFC 5880, Jun. 2010. [Online]. Available: https://www.rfc-editor.org/info/rfc5880, doi: 10.17487/RFC5880.

[20] C. Cachin, S. Schubert, and M. Vukolić, "Non-determinism in Byzantine fault-tolerant replication," 2016, *arXiv:1603.07351*. [Online]. Available: https://arxiv.org/abs/1603.07351

[21] Y. Velner *et al.*, "Some complexity results for stateful network verification," in *Proc. Int. Conf. Tools Algorithms Construct. Anal. Syst.* Berlin, Germany: Springer, 2016, pp. 811–830.

[22] *Network Functions Virtualization (NFV): Resiliency Requirements*, document ETSI-GS-NFV-REL-001, 2015. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV-REL/001_099/001/01.01.01_60/gs_NFV-REL001v010101p.pdf

[23] S. G. Kulkarni, G. Liu, K. Ramakrishnan, M. Arumaithurai, T. Wood, and X. Fu, "Reinforce: Achieving efficient failure resiliency for network function virtualization based services," in *Proc. 14th Int. Conf. Emerg. Netw. Exp. Technol.*, 2018, pp. 41–53.

[24] *Reinforce Sourcecode*. Accessed: Sep. 20, 2018. [Online]. Available: https://github.com/sameergk/REINFORCE_Supplements

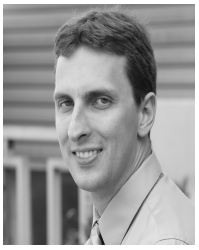
[25] W. Zhang *et al.*, "Opennetvm: A platform for high performance network service chains," in *Proc. Workshop Hot Topics Middleboxes Netw. Function Virtualization (HotMIddlebox)*, New York, NY, USA, 2016, pp. 26–31. [Online]. Available: http://doi.acm.org/2940147.2940155
[26] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "Moongen: A scriptable high-speed packet generator," in *Proc. ACM Conf. Internet Meas. Conf.*, 2015, pp. 275–287.
[27] A. Bench. *Ab-Apache HTTP Server Benchmarking Tool*. [Online]. Available: https://httpd.apache.org/docs/2.4/programs/ab.html
[28] (2018). *Wrk: A Http Benchmarking Tool*. [Online]. Available: https://github.com/wg/wrk
[29] L. Deri, M. Martinelli, T. Bujlow, and A. Cardigliano, "NDPI: Open-source high-speed deep packet inspection," in *Proc. Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, Aug. 2014, pp. 617–622.
[30] (2018). *Ndpi Test Pcap Traces*. [Online]. Available: https://github.com/ntop/nDPI/tree/dev/tests/pcap
[31] Redis Labs. (2019). *Redis*. [Online]. Available: https://redis.io
[32] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield, "Split/merge: System support for elastic execution in virtual middleboxes," presented at the 10th USENIX Symp. Netw. Syst. Design Implement. (NSDI), Lombard, IL, USA, 2013, pp. 227–240.
[33] A. Gember *et al.*, "Stratos: A network-aware orchestration layer for middleboxes in the cloud," *CoRR*, May 2013. [Online]. Available: http://arxiv.org/abs/1305.0209
[34] J. Khalid, A. Gember-Jacobson, R. Michael, A. Abhashkumar, and A. Akella, "Paving the way for NFV: Simplifying middlebox modifications using statealyzr," in *Proc. 13th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, Santa Clara, CA, USA, 2016, pp. 239–253. [Online]. Available: https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/khalid
[35] S. Woo, J. Sherry, S. Han, S. Moon, S. Ratnasamy, and S. Shenker, "Elastic scaling of stateful network functions," in *Proc. 15th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, Renton, WA, USA, 2018, pp. 299–312. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/woo
[36] C. Wang *et al.*, "PLOVER: Fast, multi-core scalable virtual machine fault-tolerance," in *Proc. 15th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, Renton, WA, USA, 2018, pp. 483–489. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/wang
[37] M. Kablan, A. Alsudais, E. Keller, and F. Le, "Stateless network functions: Breaking the tight coupling of state and processing," in *Proc. 14th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, Boston, MA, USA, 2017, pp. 97–112. [Online]. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/kablan

Sameer G. Kulkarni received the Ph.D. degree from the University of Göttingen, Germany. He is currently a Post-Doctoral Researcher at the Department of Computer Science and Engineering, University of California, Riverside, CA, USA. His current research interests include parallel and distributed computing, software-defined networks, network function virtualization, and cloud computing.

Guyue (Grace) Liu received the B.S. degree from the Beijing University of Posts and Telecommunications, and the Ph.D. degree in computer science from George Washington University. She is currently a Post-Doctoral Researcher at Carnegie Mellon University (CMU). Her research interests are in the areas of networking and systems, and she has published research papers in top conferences in these fields. She has interned and collaborated with leading research institutes, such as Microsoft Research and Hewlett Packard Labs. She has won the HP Helion OpenStack Scholarship, the First Place in GENI Competition, and an RTAS Best Student Paper Award. She was selected as one of the ten N2Women rising stars in networking and communications in 2019.

K. K. Ramakrishnan (Fellow, IEEE) received the M.Tech. degree from the Indian Institute of Science, and the M.S. and Ph.D. degrees in computer science from the University of Maryland, College Park, MD, USA, in 1978, 1981, and 1983, respectively. He was a Distinguished Member of Technical Staff at AT&T Labs-Research. Prior to 1994, he was the Technical Director and Consulting Engineer in networking at Digital Equipment Corporation. From 2000 to 2002, he was at TeraOptic Networks, Inc., as the Founder and Vice President. He is currently a Professor of computer science and engineering with the University of California, Riverside, CA, USA. He has published over 275 articles and has 180 patents issued in his name. He is a Fellow of the ACM and an AT&T Fellow, recognized for his fundamental contributions on communication networks, including his work on congestion control, traffic management, and VPN services.

Mayutan Arumaithurai received the Ph.D. degree in industrial engineering from the University of Goettingen in 2010. Meanwhile, he was working for Nokia Siemens Networks. Prior to that, he worked as a Research Scientist at the Network Laboratories of NEC Europe Ltd., Heidelberg, Germany, for two years. He is currently a Senior Researcher at the Computer Networks Group, University of Goettingen, Germany. His current research interests include information centric networking, software-defined networks, network function virtualization, and cloud computing. He has published in top conferences in his field (ACM SIGCOMM, ACM CoNext, IEEE Infocom), coauthored IETF/IRTG standards, and has led multiple million-euro EU-funded projects.



Timothy Wood received the bachelor's degree in electrical and computer engineering from Rutgers University in 2005, and the Ph.D. degree in computer science from the University of Massachusetts Amherst in 2011. He is currently an Associate Professor with the Department of Computer Science, George Washington University. His research studies how new virtualization technologies can provide application agnostic tools that improve performance, efficiency, and reliability in cloud computing data centers and software-based networks. His Ph.D. thesis received the UMass CS Outstanding Dissertation Award; his students have voted him CS Professor of the Year; and he has won three best paper awards, a Google Faculty Research Award, and an NSF Career Award.

Xiaoming Fu (Senior Member, IEEE) received the Ph.D. degree in computer science from Tsinghua University, China, in 2000. Since 2007, he has been a Professor and the Head of the Computer Networks Group, Georg-August-Universität Göttingen, Germany. He also held visiting positions at ETSI, University of Cambridge, Columbia University, Tsinghua University, and UCLA. He is a Distinguished Lecturer of the IEEE, a member of the ACM, a Fellow of the IET, and a member of Academia Europaea.