# Verifiable Network Function Outsourcing: Requirements, Challenges, and Roadmap

Seyed Kaveh Fayazbakhsh*, Michael K Reiter†, Vyas Sekar*

*Stony Brook University, †UNC Chapel Hill

## ABSTRACT

Network function outsourcing (NFO) enables enterprises and small businesses to achieve the performance and security benefits offered by middleboxes (e.g., firewall, IDS) without incurring high equipment or operating costs that such functions entail. In order for this vision to fully take root, however, we argue that NFO customers must be able to *verify* that the service is operating as intended w.r.t.: (1) functionality (e.g., did the packets traverse the desired sequence of middlebox modules?); (2) performance (e.g., is the latency comparable to an "in-house" service?); and (3) accounting (e.g., are the CPU/memory consumption being accounted for correctly?). In this position paper, we formalize these requirements and present a high-level roadmap to address the challenges involved.

## Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network management*; D.4.6 [**Operating Systems**]: Security and Protection—*Verification*

## Keywords

Network function outsourcing (NFO); verification; middlebox

## 1. INTRODUCTION

Many recent efforts have argued for bringing the benefits that virtualization and cloud computing offers—reduced capital costs, reduced operating costs, and the ability to dynamically scale services—to network deployments [5, 17, 26, 28]. This type of *network function outsourcing* (or NFO for short) is especially relevant in the context of expensive and compute-intensive middlebox functions (e.g., firewalls, intrusion detection systems, and application-level performance accelerators). The high-level vision here is that third-party providers (these could be traditional cloud providers, ISPs, or CDNs) can offer such in-the-cloud middlebox services. (We will use the terms provider, NFO provider, and cloud interchangeably.)

Given the critical role that middlebox functions serve in meeting performance, security, and policy compliance goals, middlebox NFO will likely be a significant aspect under the broader vision of network function virtualization (NFV) [9].[1] Our focus in this work is to address a fundamental concern surrounding *customer expectations* when they cede control over such functions to providers. At a high level, customers will be more comfortable with adopting NFO services if they can be assured that the services will be conceptually comparable to running those services in-house and yet offer significant savings. We argue that addressing this concern is critical to the adoption and eventual success of the NFO vision.

In this respect, we highlight three key correctness properties that an NFO service must satisfy (Section 3):

- **Functionality:** We need to ensure that the remote middlebox functions and the composition (i.e., service chaining) of these functions work as intended. For instance, the customer may want to verify that the web firewall or spam filter did apply the intended set of signature rules on the incoming packets, analogous to the case where these functions were running on dedicated "boxes" inside the customer's network under her direct control.

- **Performance:** Since the middlebox functions may be on the critical path of end-to-end applications, the customer needs to be assured that cloud-based processing and cloud networking effects do not induce unnecessary overheads w.r.t. processing delay, network latency, or end-to-end throughput.

- **Accounting:** A significant driver for NFO is reduced capital and operating expenses. Thus, the customer would naturally like to ensure that the received bill is indeed commensurate with the offered workload; e.g., she is not being charged for spurious packets or compute resources.

Achieving these correctness properties is challenging on three fronts. First, middlebox actions depend on packet contents and history of observed traffic—NFO customers may not have visibility into such effects. Second, network-level effects may introduce non-reproducible and unforeseen effects (e.g., packet reordering, transient congestion) that make it hard to reason about the correctness properties. Finally, middleboxes involve complex (and proprietary) software that may be difficult to verify.

In this paper, we highlight one possible roadmap of a *vNFO* (Verifiable NFO) architecture that seeks to address these challenges and achieve the above correctness properties (Section 4). At a high level, we leverage the hypervisor as a potential "root-of-trust" to log low-level system events to provide a basis for verifying correctness properties [7, 18]. Doing so, naively, however, will lead to significant performance and scalability overhead (e.g., logging every packet) and also fall short of directly satisfying the correctness requirements (e.g., due to the lack of visibility into middlebox state or

[1] We use NFO to scope the problem to the middlebox outsourcing context rather than the broader NFV term that encompasses several other deployment scenarios.


*HotMiddlebox'13,* December 9, 2013, Santa Barbara, CA, USA.

http://dx.doi.org/10.1145/2535828.2535831.

transformations). To address these concerns, we sketch a promising solution building on consistent system-wide sampling [13] and additional tools to provide the necessary middlebox context [14].

In many ways, the correctness requirements we highlight and the solution strategies for NFO have natural parallels in the cloud computing and distributed systems literature (see Section 5). That said, the proprietary and traffic-dependent nature of NFO services and the network-specific effects these workloads entail make these correctness properties conceptually and practically more relevant than these other use-cases, and yet significantly more challenging to verify. Our specific contribution here is in synthesizing these ideas in the specific NFO context and in providing one candidate roadmap for the future.

## 2. CHALLENGES WITH NFO

In this section, we highlight potential problems that a customer might encounter with NFO. These are different manifestations of the concern that outsourcing the network functions cedes control to the NFO provider, and, consequently, the customer cannot reason about the middlebox functions. We also highlight how the nature of the middlebox workloads and functions make this problem uniquely challenging for NFO customers.[2]

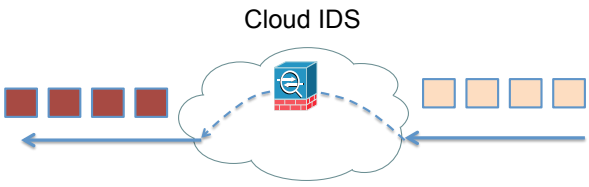



*Figure 1: Packets can be modified by the cloud when they are not expected. In this example, the NFO provider is modifying the packet payloads while the requested function (i.e., IDS) should not alter packet contents.*

**Processing semantics:** Let us consider the simple example in Figure 1 with an enterprise using a cloud-based IDS/IPS service. The trouble here is that the enterprise may only observe the processed packets and does not have any visibility into the original packets. For instance, a misconfigured NFO provider may rewrite or introduce spurious advertisements (e.g., [23]) or incorrectly modify the packet payloads. Similarly, the enterprise may not know if the IPS incorrectly dropped some packets or if these drops were due to network effects (e.g., full buffers).

**Performance guarantees:** The NFO customer needs to ensure that her applications do not suffer due to performance overheads introduced by outsourcing. This is especially challenging due to the non-deterministic performance effects that middlebox processing and network-level operations raise. For instance, in Figure 2 it is difficult to tease out the different delays caused by the NFO provider vs. those induced by the network en-route. The proprietary and stateful nature of middlebox actions may further introduce non-deterministic effects; e.g., batch processing packets [19] or delaying them for more redundancy elimination [3].

**Verifying the bill:** A key driver for NFO is the promise of reduced capital and operational costs and dynamically scaling the deployment. NFO customers would naturally like to be assured that the bill they receive from the NFO provider is commensurate with the actual work being done. This is, however, easier said than done as the customer does not see the actual workload, middlebox functions, and the resources consumed. For instance, in Figure 3, even though the output seen by the customer is equivalent, the cloud provider is running a different middlebox from what was intended and thus charging for resources that were never actually consumed. Network-level effects and traffic dependence further exacerbate this problem; for instance, a provider may charge for processing cycles that eventually dropped a set of packets, but the customer cannot check if these packets are real or spurious. Finally, due to complex processing involved, it is not easy to determine in advance the exact resource consumption, and thus customers may not be able to estimate their expenses a priori [16].

[2]One might argue that this lack of control exists even with locally-owned middleboxes. Our goal is to provide a conceptual equivalence between running the (possibly black-box) middlebox function locally vs. remotely.

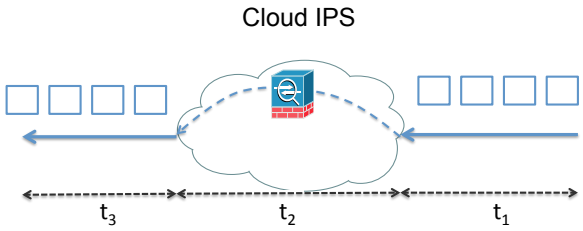
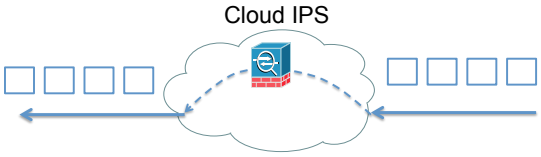
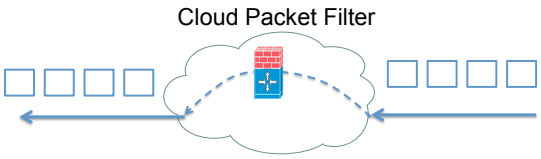



*Figure 2: The NFO customer cannot infer the performance of the service just by looking at the observed traffic. In this example, the customer cannot determine what fraction of total delay is due to round trip communication (i.e., $t_1+t_3$) vs. actual cloud processing (i.e., $t_2$).*




*(a)* The customer has requested an IPS function.




*(b)* The cloud runs a different and perhaps cheaper/easier to run function (e.g., a packet filter).

*Figure 3: The NFO provider is not using the resource it is billing the customer for. Note that the black-box functionality of the cloud is correct on the shown sequence of packets.*

In summary, we see that three factors make it difficult for the customer to reason that NFO service is running as intended: (1) lack of visibility into the workload, (2) dynamic, traffic-dependent, and potentially proprietary actions of the middleboxes, and (3) stochastic effects introduced by the intermediate network. In the rest of the paper, we begin by formalizing the correctness requirements before outlining a promising candidate solution.

## 3. NFO CORRECTNESS PROPERTIES

In this section, we attempt to formally specify the correctness properties for an abstract NFO service. This formalism is useful on two fronts. First, it helps us to systematically model the motivating scenarios from the previous section. Second, it also serves to in-

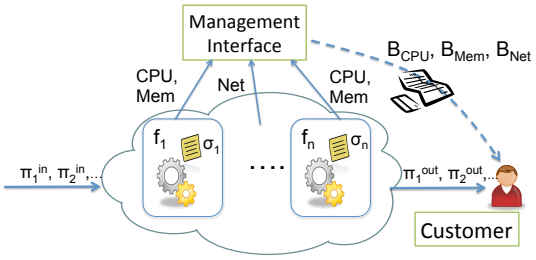



*Figure 4: The system parameters necessary to specify the formal correctness requirements of NFO: the sequence of input packets (i.e., $\pi_1^{\mathsf{in}}, \pi_2^{\mathsf{in}}, \ldots$) is processed by a sequence of functions (i.e., $f_1^{\mathsf{pkt}}, f_2^{\mathsf{pkt}}, \ldots$) in the NFO provider; the sequence of processed packets (i.e., $\pi_1^{\mathsf{out}}, \pi_2^{\mathsf{out}}, \ldots$) is then sent to the customer along with a bill that represents usage of various resources (e.g., $B_{CPU}$, $B_{Mem}$, $B_{Net}$).*

form the design of a verifiable NFO by highlighting the system and environment effects that need to be captured.

**Preliminaries:** Let $f : (\Pi \times \Sigma) \rightarrow (\Pi \times \Sigma)$ denote a *primitive middlebox function* that takes as input a *packet* and a *state*, and outputs a packet and a new state (see Figure 4). More specifically, $\Pi$ denotes the set of all packets, and $\Sigma$ is the set of *reachable* states for $f$. For convenience, we specify that $\Pi$ includes a special symbol "$\perp$" and that each primitive function $f$ satisfies $(\perp, \sigma) \leftarrow f(\perp, \sigma)$ for all $\sigma \in \Sigma$. The symbol "$\perp$" captures packets that are dropped by $f$ (e.g., a packet that matches a drop rule at a firewall function). Let $f^{\mathsf{pkt}}(\pi, \sigma)$ and $f^{\mathsf{st}}(\pi, \sigma)$ denote the packet and state outputs of $f(\pi, \sigma)$, respectively.

Let superscripts $\mathsf{in}$ and $\mathsf{out}$ denote whether the corresponding parameter is an input or an output of a function. We assume that a customer has contracted with the NFO provider to subject its packets to the *service chain* $f_1, f_2, \ldots, f_n$ of primitive functions. That is, if $\vec{\pi}^{\mathsf{in}} \in \Pi^*$ denotes the sequence of packets that a customer expects to be processed in the cloud, then she expects to receive in return a sequence $\vec{\pi}^{\mathsf{out}} \in \Pi^*$ of the same length. (Some elements of $\vec{\pi}^{\mathsf{out}}$ might be $\perp$, indicating that the corresponding packet in $\vec{\pi}^{\mathsf{in}}$ was dropped.) Let $\pi_{ji}$ denote the packet output by $f_i$, as run by the NFO provider, corresponding to the $j$th packet of the input sequence $\vec{\pi}^{\mathsf{in}}$ as its input. Also, let $\Sigma_i$ denote the state space of $f_i$. Informally, the $j$-th element of $\vec{\pi}^{\mathsf{out}}$, denoted by $\pi_j^{\mathsf{out}}$, should be produced by setting $\pi_{j0} \leftarrow \pi_j^{\mathsf{in}}$ and then applying

$$\pi_{j1} \leftarrow f_1^{\mathsf{pkt}}(\pi_{j0}, \sigma_1^{\mathsf{in}}); \sigma_1^{\mathsf{out}} \leftarrow f_1^{\mathsf{st}}(\pi_{j0}, \sigma_1^{\mathsf{in}})$$
$$\pi_{j2} \leftarrow f_2^{\mathsf{pkt}}(\pi_{j1}, \sigma_2^{\mathsf{in}}); \sigma_2^{\mathsf{out}} \leftarrow f_2^{\mathsf{st}}(\pi_{j1}, \sigma_2^{\mathsf{in}})$$
$$\vdots$$
$$\pi_{jn} \leftarrow f_n^{\mathsf{pkt}}(\pi_{j(n-1)}\sigma_n^{\mathsf{in}}); \sigma_n^{\mathsf{out}} \leftarrow f_n^{\mathsf{st}}(\pi_{j(n-1)}, \sigma_n^{\mathsf{in}})$$

and then setting $\pi_j^{\mathsf{out}} \leftarrow \pi_{jn}$. Note that in the above formulation the output state of $f_i$ (i.e., $\sigma_i^{\mathsf{out}}$) will be used as its input state (i.e., $\sigma_i^{\mathsf{in}}$) in the next invocation of $f_i$.

Suppose each invocation $f_i(\pi_{j(i-1)}, \sigma_i^{\mathsf{in}})$ consumes a set of measurable computational resources $\mathcal{R}[f_i(\pi_{j(i-1)}, \sigma_i^{\mathsf{in}})] = \langle Res_1, \ldots, Res_R \rangle$ in units suitable for each resource; e.g., CPU cycles or instantaneous memory consumption. Let each invocation of $f_i$ be associated with invocation and completion timestamps $T^{\mathsf{in}}(\pi_{j(i-1)}, f_i)$ and $T^{\mathsf{out}}(\pi_{ji}, f_i)$, respectively.

The correctness properties, which are discussed next, use the notion of the *reference implementation* of function $f_i$, denoted by $\widehat{f}_i$, which serves as a point of reference for verifying each property. We assume the customer has access to the reference implementation. For instance, $\widehat{f}_i$ may represent the case of running the actual VM image of the $i$th function locally in the customer's site.

## 3.1 Functional Correctness

Our first goal is to verify the semantic behavior of the outsourced functions. We describe this correctness as occurring at two levels: at the level of an individual primitive function and then at the level of the entire pipeline or chain composed of multiple primitive functions.

As a starting point, we list two properties that must be guaranteed for each primitive function:

- *Black-box primitive equivalence:*
  Given $\pi_{j(i-1)}, \pi_{ji} \in \Pi$:
  $\exists \sigma_i^{\mathsf{in}} \in \Sigma_i : \pi_{ji} = \hat{f}_i^{\mathsf{pkt}}(\pi_{j(i-1)}, \sigma_i^{\mathsf{in}})$

  This property states that if we can log the incoming/outgoing packet at a given middlebox, then there is some instantiation of the state variables for the reference function $\widehat{f}_i$ that could have output the observed packet.

- *Snapshot primitive equivalence:*
  Given $\pi_{j(i-1)}, \pi_{ji} \in \Pi$, and $\sigma_i^{\mathsf{in}} \in \Sigma_i$:
  $\pi_{ji} = \hat{f}_i^{\mathsf{pkt}}(\pi_{j(i-1)}, \sigma_i^{\mathsf{in}})$

  The guarantee provided by the black-box primitive equivalence is a weak form of correctness, as it just states that there is some possible execution sequence. The snapshot primitive equivalence provides a stronger notion of correctness by binding the execution to the known current state $\sigma_i^{\mathsf{in}}$.

Building on this, we extend the correctness properties to the full pipeline of functions as follows:

- *Black-box pipeline equivalence:*
  Given $\pi_j^{\mathsf{in}}, \pi_j^{\mathsf{out}} \in \Pi$:
  $\exists \sigma_1^{\mathsf{in}} \in \Sigma_1, \ldots, \sigma_n^{\mathsf{in}} \in \Sigma_n :$
  $\pi_j^{\mathsf{out}} = \hat{f}_n^{\mathsf{pkt}}(\ldots \hat{f}_2^{\mathsf{pkt}}(\hat{f}_1^{\mathsf{pkt}}(\pi_j^{\mathsf{in}}, \sigma_1^{\mathsf{in}}), \sigma_2^{\mathsf{in}}), \ldots, \sigma_n^{\mathsf{in}})$

  This is the most basic requirement where we want to make sure that there is some instantiation of the internal states of the middleboxes and the intermediate packets that could have resulted in the observed input-output behavior.

- *Snapshot pipeline equivalence:*
  Given $\pi_j^{\mathsf{in}}, \pi_j^{\mathsf{out}} \in \Pi$, $\sigma_1^{\mathsf{in}} \in \Sigma_1, \ldots, \sigma_n^{\mathsf{in}} \in \Sigma_n$:
  $\pi_j^{\mathsf{out}} = \hat{f}_n^{\mathsf{pkt}}(\ldots \hat{f}_2^{\mathsf{pkt}}(\hat{f}_1^{\mathsf{pkt}}(\pi_j^{\mathsf{in}}, \sigma_1^{\mathsf{in}}), \sigma_2^{\mathsf{in}}), \ldots, \sigma_n^{\mathsf{in}})$

  As in the earlier case, we are strengthening the correctness requirement by additionally binding the internal states of the respective functions at the time of processing.

As we will see in the next section, the different properties entail different monitoring requirements. For instance, the blackbox properties only require the input/output packets, whereas the snapshot properties also need to account for middlebox state.

## 3.2 Performance Correctness

Suppose the NFO customer and provider have contractually agreed on a set of performance measures $\mathcal{M} = \langle Metric_1, \ldots, Metric_M \rangle$ as part of their service-level agreement. Each $Metric_m$ is computed as a summary statistic w.r.t. metric $m$ (e.g., average delay) over the sequence of packets processed by the functions; i.e., $Metric_m =$

$PerfSum_m(\{\pi_{ji}, T^{\text{in}}(\pi_{j(i-1)}, f_i), T^{\text{out}}(\pi_{ji}, f_i)\}_{j,i})$. To simplify the presentation, we use a generalized form of the performance computation that includes all time stamps and packet samples. In practice, some performance summary metrics may not need all the inputs. For instance, end-to-end delay metrics may not need the intermediate $\pi_{ji}$ values.

In order to decouple wide-area network effects from the in-cloud processing effects (see Figure 2), we want to compare the performance w.r.t. the reference implementation $\widehat{f_i}$ for each of the functions. If $\widehat{Metric_m}$ represents the corresponding performance with the reference implementation, then we want the following property:

$$\forall m : Metric_m \in \Delta_m^{Metric}(\widehat{Metric_m})$$

Here, $\Delta_m^{Metric}$ represents the space of acceptable values (along with some additional slack) depending on the specific performance metric $m$ and the SLA. For instance, the agreement might be that the 95%ile of the end-to-end processing delay is at most 10 ms higher than the reference implementation.

## 3.3 Accounting Correctness

Similar to performance correctness, accounting correctness needs a contractually agreed billing specification. Let $\mathcal{B} = \langle B_1 \ldots B_R \rangle$ be the measured values of the billed resources such as CPU or memory consumption. Each billed resource $r$ is measured as $B_r = ResSum_r(\{\mathcal{R}[f_i(\pi_{j(i-1)}, \sigma_i^{\text{in}})]\}_{j,i})$, where the value of each $\mathcal{R}[.]$ is the instantaneous resource consumption, and the summary function $ResSum_r$ is customized for different types of resources. For instance, the provider may charge for every CPU cycle but for memory it may use the 95%ile of memory consumption.

Given this, we consider two natural notions of accounting correctness:

- *Consistency:* Given a trusted log of the $\{\mathcal{R}[f_i(\pi_{j(i-1)}, \sigma_i^{\text{in}})]\}_{j,i}$ values and the specification of the $ResSum_r$ functions, the customer can determine whether the charged value $B_r$ is consistent with the actual physical consumption.

- *Minimality:* The above notion of consistency, while useful, does not tell us if the customer *should have* incurred the consumption vector $\{\mathcal{R}[f_i(\pi_{j(i-1)}, \sigma_i^{\text{in}})]\}_{j,i}$. Specifically, there may be a larger footprint due to provider oversight or misconfigurations; e.g., due to poor VM scheduling or network retransmissions. Thus, we also define a stronger check w.r.t. the reference implementation:

  $$\forall i, j, r : \mathcal{R}[f_i(\pi_{j(i-1)}, \sigma_i^{\text{in}})] \in \Delta_r^{Res}(\mathcal{R}[\widehat{f_i}(\pi_{j(i-1)}, \sigma_i^{\text{in}})])$$

  As in the performance case, $\Delta_r^{Res}$ captures the space of allowable values denoting the acceptable slack due to stochastic effects.

# 4. ROADMAP FOR vNFO

In this section, we outline a high-level Verifiable NFO or vNFO architecture that can address the requirements from the previous sections.

**Assumptions:** To make our discussion concrete, we scope the problem along two dimensions. First, we assume that each middlebox function is implemented as a *virtual appliance*[3] and that each NFO customer is assigned a dedicated set of VMs that are used exclusively to process its traffic. Thus, we do not focus on isolation and security issues due to multiplexing across customers. Second, we consider an operational model where the NFO provider is incentivized to provide customers with attested logs via trusted modules (e.g., TPM). We believe that this is aligned with provider interests, as they can incentivize adoption by providing additional assurances to otherwise wary customers, or use vNFO to differentiate their offering in a commoditized market, or offer a premium vNFO service to offset overheads [7, 25].

[3]We are agnostic to whether these are customer specified, NFO provider owned, or if they are third-party offerings.

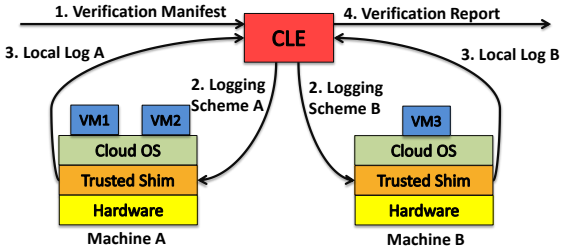



*Figure 5: Overview of vNFO: The customer specifies the logging requirements (e.g., sampling rate). The CLE configures the shims and the VMs with the FlowTags [14] and trajectory sampling [13] parameters. The CLE integrates the local logs and sends the verification report to the customer.*

## 4.1 High-level Approach

We begin by considering an extreme design point, where the NFO customer receives an attested log of every packet processing event, snapshots of each middlebox VM image at packet processing time, and associated metadata (e.g., resource footprints and timestamps). Given these logs and the reference implementation $\widehat{f_i}$, the customer can simply replay the trace with local instances to verify the correctness properties (e.g., [18]).

While this idealized view helps us identify an intuitive solution, there are two obvious practical challenges:

1. *Visibility:* Middleboxes may dynamically modify packet headers and contents. Consequently, even with elaborate logging, we may not be able to collate system-wide logs, as the same packet may have different incarnations [14].
2. *Scalability:* Pervasive logging may impose significant overhead for the provider (e.g., instrumenting and collecting logs) and the customer (e.g., receiving and replaying logs).

Next, we discuss potential solutions to each challenge.

## 4.2 vNFO Overview

We envision a lightweight trusted *shim* that sits between the physical platform and the middlebox VMs as shown in Figure 5.[4] The shim is responsible for logging the relevant packet arrival and departure events and other system-level measurements (e.g., RDTSC for CPU cycles). This offers a trusted vantage point to obtain relevant events rather than having to modify every single middlebox VM. A trusted Central Logging Entity (CLE) generates the system-wide view by correlating logs from different shim instances and also records other relevant external events (e.g., network congestion).

As discussed above, there are two key concerns w.r.t. visibility and scalability. First, to tackle the visibility challenge, we leverage recent work on enhancing middleboxes with the FlowTags API [14]. The intuition is that middleboxes export the relevant

[4]As the shim is functionally minimal, unlike a full-fledged hypervisor, it may be amenable to static checking.

packet in-out mappings using tags in the packet header to the CLE. For example, a NAT or a proxy tells the CLE how it mapped the packet headers. These tags enable the CLE to track the trajectory of (modified) packets throughout the network.

Second, we rely on *sampling* to reduce the logging overhead. If we verify the correctness properties for a subset of (pseudo-randomly) chosen packets, then we can provide a tunable degree of assurance. Note, however, a naive uniform sampling approach may not work. For instance, if we use uniform random sampling at different middleboxes, we cannot reason about the pipeline/compositional properties in terms of functionality, performance, or accounting. What we need is a consistent *trajectory* sampling scheme, where a given packet (and its attendant metadata) is logged at *every* element on its logical path [13]. At a high level, trajectory sampling works as follows. Each node in the network computes a hash of the *invariant* fields of the packet header and logs packets if the hash falls within the assigned and globally common hash range. (Invariant fields are the ones that do not change as the packet traverses the network; e.g., TTL is not invariant.) Since every element is logging the same subset of packets, we can reconstruct the packet trajectory. In the context of middleboxes that may modify headers, this necessitates the use of FlowTags rather than the actual IP 5-tuple/payload to provide these invariants.

## 4.3 End-to-end View

The customer and provider agree on a *verification manifest*, which indicates a logging specification (e.g., keys to the sampling hash function and the desired sampling frequency) along with the required logging granularity (e.g., packets and/or middlebox VM state). The provider generates an attestation that the intended shim and the middlebox VMs are running using traditional TPM-based mechanisms [1].

The CLE instruments shim instances with the relevant sampling parameters. Each shim reports the set of logged packets, associated VM memory context, and metadata regarding packet in/out timestamps, FlowTags labels, and resource consumptions for the subset of packets (e.g., CPU cycles, memory usage). The CLE periodically sends this report to the customer. To reduce the bandwidth costs of communicating the reports, they can be compressed as needed, for example, using diffs between memory snapshots.

Next, we discuss how the customer can verify the correctness properties within the vNFO framework:

- *Functional correctness:* Given the packet logs and the detailed VM snapshots, the customer uses a deterministic replay scheme by running the VM from the reported memory snapshot and checks if $\widehat{f}$ functionally behaves in the same way [18]. In case the customer decides to only enable packet logging (e.g., to reduce bandwidth costs), she can revert to blackbox correctness as follows. Using the reference VM image for each $\widehat{f}$ and the logged packets, we use symbolic checking to ensure that there is some valid input state that could have resulted in the output packet (e.g., [11]). Because the customer has the entire packet trajectory, we can speed up the checking for the pipeline equivalence by pruning the search space w.r.t. intermediate outputs.

  For the rest of this discussion, we assume that the customer has the VM snapshots as well.
- *Performance correctness*: The above replay step provides a baseline reference for $\widehat{Metric}_m$ for each metric $m$. Given this reference value, the customer can check if the reported value $Metric_m$ is within the distribution specified by $\Delta_m^{Metric}(\widehat{Metric}_m)$. As discussed earlier, $\Delta_m^{Metric}$ provides some slack to account for natural variability in performance. Note that we are not verifying the wide-area network characteristics [4, 22, 29]; we can only assure the customer that there are no undesirable effects introduced by the NFO provider in order to help the customer to disambiguate the different sources of observed poor performance (Figure 2).
- *Accounting correctness*: We ensure the basic consistency property using the sampled trace along with the (attested) $\{\mathcal{R}[f_i(\pi_{j(i-1)}, \sigma_i^{\text{in}}]\}_{j,i}$ values. We need to renormalize the $ResSum_r$ functions by the sampling rate and then check if the billed value $B_r$ is within the sampling-induced error. To verify the minimality property, the customer uses the execution replay with the $\widehat{f}$s and then checks if the ideal resource consumption vectors (for the sampled packets) are within the slack specified by the $\Delta_r^{Res}$ functions.

# 5. RELATED WORK

In addition to the specific related work highlighted inline, our vision is related to cloud computing, network monitoring, and trusted computing. We briefly discuss some of these and highlight the new challenges or opportunities that NFO entails.

**Accountable networking:** ICING ensures path compliance in an adversarial environment via cryptographic techniques [22]. Keller et al. suggest the use of trusted switch interface cards to provide guarantees in virtual networks [20]. Zhang et al. [29] and Argyraki et al. [4] provide mechanisms to pinpoint nodes that maliciously drop packets. Many of these efforts focus on wide-area or federated routing scenarios. In contrast, we want to ensure that the packets within an NFO provider traverse the intended sequence of middleboxes. Given the virtualized environment, we use a trusted shim layer instead of relying on heavyweight cryptographic techniques or new trusted hardware.

**Auditing cloud outsourcing:** NFO inherits several natural concerns associated with the lack of control with cloud outsourcing [10]. Alibi focuses on providing accounting consistency for traditional cloud applications [7]. vNFO considers a different use case of outsourcing and additionally focuses on functional and performance correctness. CloudCmp is a framework to compare performance and cost of different cloud providers [21] and is complementary to vNFO.

**Performance monitoring and debugging:** The idea of using tracing to debug and pinpoint performance problems is far from new and has natural parallels in operating systems (e.g., [2,15]) and networking (e.g., [13]). Our contribution is in the synthesis of these techniques to satisfy the new correctness requirements that NFO raises.

**Remote attestation and verification:** vNFO builds on a rich literature in the areas of security and distributed systems on trusting remote software agents (e.g., [6, 12, 24]). Recent research provides more efficient methods for verifying specific types of computations (e.g., [27]) and for efficient replay for virtual machines (e.g., [18]).

**Middlebox NFO:** NFO/NFV is an idea that has been around for a while (e.g., [28]) and has recently seen renewed interest given the success of cloud computing and virtualization [5, 8, 17, 26]. These prior efforts focus on the viability of NFO and architectural support for NFO, and do not discuss the correctness properties discussed here.

# 6. CONCLUSIONS AND FUTURE WORK

The NFO space is still evolving — how middlebox modules are realized (i.e., VMs vs. hardware), shared vs. dedicated VMs, single vs. federated provision models and many other alternatives. Our

goal in this paper was to take a first step to articulate the correctness requirements for NFO and present an initial sketch of a solution for a specific point in this problem space. We do not claim that vNFO is the only feasible solution or that it is optimal in any way.

Our immediate near-term goal is to make the vNFO vision practical and demonstrate that the performance overhead of doing so is minimal. Our longer-term agenda is to extend our vision to other points in the NFO problem space; e.g., environments where middleboxes (or VMs) are multiplexed across customers; dealing with security/privacy concerns such as potential information leakage, and considerations for federated NFO offerings (i.e., functions distributed across providers).

## ACKNOWLEDGMENTS

This work was supported in part by grant number N00014-13-1-0048 from the Office of Naval Research and by Intel Labs' University Research Office.

## 7. REFERENCES

[1] Trusted Computing Group. `http://www.trustedcomputinggroup.org/`.

[2] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen. Performance debugging for distributed systems of black boxes. In *Proc. SOSP*, 2003.

[3] A. Anand, V. Sekar, and A. Akella. SmartRE: an architecture for coordinated network-wide redundancy elimination. In *Proc. SIGCOMM*, 2009.

[4] K. Argyraki, P. Maniatis, and A. Singla. Verifiable network-performance measurements. In *Proc. CoNEXT*, 2010.

[5] T. Benson, A. Akella, A. Shaikh, and S. Sahu. CloudNaaS: a cloud networking platform for enterprise applications. In *Proc. SOCC*, 2011.

[6] S. Berger, R. Cáceres, D. Pendarakis, R. Sailer, E. Valdez, R. Perez, W. Schildhauer, and D. Srinivasan. TVDc: managing security in the trusted virtual datacenter. *SIGOPS Oper. Syst. Rev.*, 42(1):40–47, Jan. 2008.

[7] C. Chen, P. Maniatis, A. Perrig, A. Vasudevan, and V. Sekar. Towards verifiable resource accounting for outsourced computation. In *Proc. VEE*, 2013.

[8] Y. Chen, B. Liu, Y. Chen, A. Li, X. Yang, and J. Bi. PacketCloud: an open platform for elastic in-network services. In *Proc. MobiArch*, 2013.

[9] M. Chiosi et al. Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action. Technical report, ETSI, 2012.

[10] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina. Controlling data in the cloud: outsourcing computation without outsourcing control. In *Proc. CCSW*, 2009.

[11] R. A. Cochran and M. K. Reiter. Toward online verification of client behavior in distributed applications . In *Proc. NDSS*, 2013.

[12] G. Coker, J. Guttman, P. Loscocco, A. Herzog, J. Millen, B. OHanlon, J. Ramsdell, A. Segall, J. Sheehy, and B. Sniffen. Principles of remote attestation. *International Journal of Information Security*, 10(2):63–81, 2011.

[13] N. G. Duffield and M. Grossglauser. Trajectory sampling for direct traffic observation. *IEEE/ACM Trans. Netw.*, 9(3):280–292, June 2001.

[14] S. K. Fayazbakhsh, V. Sekar, M. Yu, and J. C. Mogul. FlowTags: enforcing network-wide policies in the presence of dynamic middlebox actions. In *Proc. HotSDN*, 2013.

[15] R. Fonseca, G. Porter, R. H. Katz, S. Shenker, and I. Stoica. X-Trace: a pervasive network tracing framework. In *Proc. NSDI*, 2007.

[16] A. Ghodsi, V. Sekar, M. Zaharia, and I. Stoica. Multi-resource fair queueing for packet processing. In *Proc. SIGCOMM*, 2012.

[17] G. Gibb, H. Zeng, and N. McKeown. Outsourcing network functionality. In *Proc. HotSDN*, 2012.

[18] A. Haeberlen, P. Aditya, R. Rodrigues, and P. Druschel. Accountable virtual machines. In *Proc. OSDI*, 2010.

[19] S. Han, K. Jang, K. Park, and S. Moon. PacketShader: a gpu-accelerated software router. In *Proc. SIGCOMM*, 2010.

[20] E. Keller, R. B. Lee, and J. Rexford. Accountability in hosted virtual networks. In *Proc. VISA*, 2009.

[21] A. Li, X. Yang, S. Kandula, and M. Zhang. CloudCmp: comparing public cloud providers. In *Proc. IMC*, 2010.

[22] J. Naous, M. Walfish, A. Nicolosi, D. Mazières, M. Miller, and A. Seehra. Verifying and enforcing network paths with ICING. In *Proc. CoNext*, 2011.

[23] C. Reis, S. D. Gribble, T. Kohno, and N. C. Weaver. Detecting in-flight page changes with web tripwires. In *Proc. NSDI*, 2008.

[24] N. Santos, K. P. Gummadi, and R. Rodrigues. Towards trusted cloud computing. In *Proc. HotCloud*, 2009.

[25] V. Sekar and P. Maniatis. Verifiable resource accounting for cloud computing services. In *Proc. CCSW*, 2011.

[26] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making middleboxes someone else's problem: network processing as a cloud service. In *Proc. SIGCOMM*, 2012.

[27] V. Vu, S. Setty, A. Blumberg, and M. Walfish. A hybrid architecture for interactive verifiable computation. In *IEEE Symposium on Security and Privacy*, 2013.

[28] M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, and S. Shenker. Middleboxes no longer considered harmful. In *Proc. OSDI*, 2004.

[29] X. Zhang, A. Jain, and A. Perrig. Packet-dropping adversary identification for data plane security. In *Proc. CoNext*, 2008.