Protocols to Support Autonomy and Control for NFV in Software Defined Networks

Ali Mohammadkhan*, Guyue Liu‡, Wei Zhang‡, K. K. Ramakrishnan*, and Timothy Wood‡
*University of California, Riverside
‡The George Washington University
Email: {amoha006, kk}@ucr.edu and {timwood, guyue, zhangwei1984}@gwu.edu

***Abstract*—The use of Network Function Virtualization to run network services in software enables Software Defined Networks to create a largely software-based network. We envision a dynamic and flexible network that can support a smarter data plane than just simple switches that forward packets. This network architecture needs to support complex stateful routing of flows where processing by network functions (NFs) can dynamically modify the path taken by flows, without unduly burdening or depending on the centralized SDN controller. To this end, we specify a protocol across the different components of an SDN-NFV environment to support the creation of NFs required by a service graph specification, using an orchestrator speaking to an NF Manager running on each host. We take advantage of, and extend, the concept of the SDN controller-to-node protocol (OpenFlow being the most popular) and tagging flows to support complex stateful routing. Output generated by NFs processing packets may be returned to the NF Manager to influence dynamic route changes based on a priori rules defined through a service graph specification provided by network administrators. We envisage the SDN controller setting up these rules based on the output from NFs, the flow specification as well as global tags. By not treating tags as an independent component for routing, we show that we can dramatically reduce the number of tags required across the entire network. Further, by providing the right autonomy in decision making at the NF Manager and the individual NFs in our hierarchical control framework, we significantly reduce the load on the SDN controller.**

***Index Terms*—Software Defined networking, network function virtualization, network function placement.**

I. Introduction

Software Defined Networking (SDN) promises to provide greater flexibility for precisely directing packet flows using a software-based control plane for the network. The underlying assumption is that the data plane is simple, comprising commodity switches, with little or no state other than the forwarding table of 'match-action' rules populated by a logically centralized SDN controller. However, today's networks are much more than just packet forwarding entities, with complex network services prevalent in custom-built middlebox systems at the edges of the network. Network Function Virtualization (NFV) has emerged as a technique to run high performance network services as software running in virtual machines (VMs) on commodity servers. NFV thus enables easy deployment of software-based network functions dynamically in the network, at a much lower cost. Both these technologies promise to radically alter how networks are deployed and managed, offering greater flexibility and enabling network services to be added on demand.

In our SDNFV architecture [1], we use the SDN controller and the NF orchestrator, coordinated by an SDNFV Application to create a more flexible and dynamic network: the SDN controller manages the network control plane [2] and the NF Orchestrator manages NFV instances and their flow state [3]. The SDNFV architecture is consistent with ETSI NFV architecture [4], with some differences, as we extend the ETSI model's adherence to the stricter control plane - data plane separation espoused by the SDN architecture. For example orchestration roles of NFV Management and Orchestration unit is assigned to the NF Orchestrator and management duties are assigned to SDN controller. A placement engine decides how functions are instantiated across the network, and the NF VMs on each host are controlled by an NF Manager. This paper focuses on the protocols that split control across this hierarchy—exploiting the application-awareness of individual NFs, the host-level resource management capabilities of the NF Manager, and the global view provided by the SDN Controller and the SDNFV Application.

One of the approaches for managing complex middlebox deployments considered in the previous work [5], [6] is to introduce additional information in the form of *tags* to mark packets processed by a middlebox, and use the tag to indicate the result of the processing to the SDN controller. The SDN controller may then use this information to re-route the flow, potentially on a path that might appear to have loops when viewed at the network layer, but is required to route the flow through different middleboxes connected to the same switch [5]. In the FlowTags approach [6], an NF first receiving a tagged flow will 'consume' the tag after contacting the controller to elicit its meaning. However, existing approaches result in significant overhead due to the need for frequent communication with the SDN controller to generate and interpret tags. Further, the amount of packet header space dedicated to tags may become unreasonable since a large number of unique tags are needed to differentiate each hop in the path, for all the flows in the network.

To achieve our goal of having hierarchical control where both a smarter data plane and a logically centralized SDN controller have the capability to route flows appropriately as well as dynamically instantiate functions in the network, we also leverage the idea of using tags as in FlowTags [6], but

978-1-4673-6884-1/15/$31.00 ©2015 IEEE

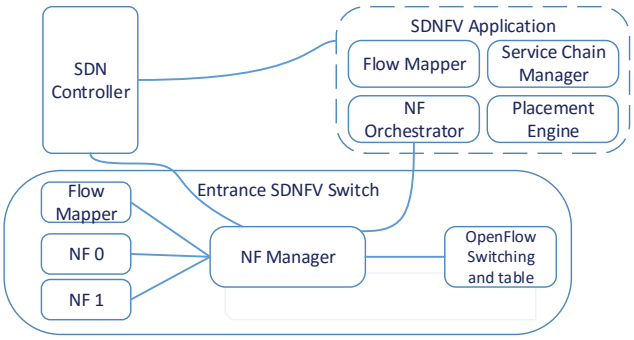

Fig. 1. System components and their interactions (All the other switches are similar to the entrance switch but they do not have the Flow Mapper in them.)

suitably enhanced. We use the tags only when their existence is crucial and a forwarding decision would be ambiguous without using the tags. We have designed a set of protocols so that NFs can impact the route for a flow (or a class of flows), as well as communicate information such as the flow's state to the higher levels in the control hierarchy (the NF Manager and the SDN Controller). Our work makes the following contributions:

- A hierarchical architecture to efficiently and flexibly combine SDN and NFV.
- An approach for supporting diverse NF service chains that support dynamic routing of flows based on the output of the NFs, even for NFs changing headers such as a NAT or for NFs affecting upstream NFs in the chain.
- Leveraging an SDN controller's knowledge about network topology and NF placement to limit the use of tags only to essential cases, where routing is ambiguous without their existence.
- Protocols across the SDN/NFV hierarchy that minimize the amount of communication with the SDN controller by exploiting the intelligence in the software-based data plane.

We evaluate the benefits provided by our architecture in terms of reducing SDN control message overhead and the number of distinct tags (and thus the additional header space) required network-wide.

II. System Components

Our SDNFV architecture provides a hierarchy of control with NFs at the bottom and the SDNFV Application at the top, as illustrated in Figure 1. In this section, we explain the role of the components in the architecture.

SDNFV Switches: The commodity servers running NFs in our platform are called SDNFV Switches, since we view them as "smart switches" that can be deployed in the network to not only forward packets, but also apply complex functionality to packet flows. As software based switching achieves near-wire-rate performance [7], [8]—even in virtual environments [9], [10]—the distinction of having NFs in COTS servers versus specialized switching hardware is likely to blur, and our SDNFV architecture recognizes this evolution. SDNFV switches interact with both the SDN controller and the NF orchestrator.

Network Functions (NFs): Today's networks comprise many different functions beyond simple forwarding. These include middleboxes such as firewalls, proxies, network address translators (NAT), etc. In our architecture, NFs are able to generate an output when they process a packet. This output is send to NF Manager by using the shared memory between the NFs and the NF Manager.

NF Manager: Each SDNFV Switch may run multiple NFs coordinated by a single NF Manager on that host. The NF Manager is a software layer between the NFs and the operating system of the SDNFV switch. The NF Manager is responsible for managing NFs residing on the host (creating instances, forwarding packets to and between NFs), and controls the OpenFlow-like forwarding table. All the interaction to external network components (such as the SDN Controller, the NF Orchestrator) with NFs is mediated through the NF Manager.

SDNFV Application: At the top of the hierarchy is the SDNFV Application. This is where the network administrator specifies the overall application logic that will control where NFs are instantiated and how packet flows will be routed through them. The SDNFV Application has four main components:

Service Chain Manager: This provides an interface to define sequences of NFs (e.g., middlebox functions) traversed in a specified order. These service chains can be dynamic, in that the output of an NF can result in forwarding the packet (or subsequent packets of the flow) to one of several different NFs. Thus, the chain is actually a graph with branches based on NFs outputs, rather than a linear sequence. Several example service chains are shown in Figure 2. Each service chain is given a *Chain ID* and a priority. When a packet flow matches the criteria for multiple service chains, the service chains are applied in priority order.

Flow Mapper: After defining a service chain, we need to assign it to the appropriate packet flows. In its simple form, the mapping may be predetermined by a network administrator based on the IP 5-tuple, including the wild-carding of some fields. For more complex situations, where the decision has to be based on the state of the flow (or related to multiple flows), the flow mapping may be done dynamically as packets arrive. This task is carried out by a flow-to-service chain mapping engine implemented as a part of the SDNFV Application or as one of the NFs in the entry switches. In the former case, new flows are sent to SDN controller and in the latter, the result of flow mapper NF is sent to SDN controller. Using NFs to perform flow classification, as shown in Figure 1, may be preferable if detecting the flow type is computationally intensive.

Placement and Routing Engine: The Placement Engine receives information about the flows, the service graph for those flows and the current network conditions (e.g., available capacity, topology) to decide where to instantiate NFs. We have proposed a mixed integer linear programming (MILP) solution for this complex problem, as well as heuristics to obtain results close to the optimal solution, in reasonable time [11]. The heuristics also enable us to solve the problem

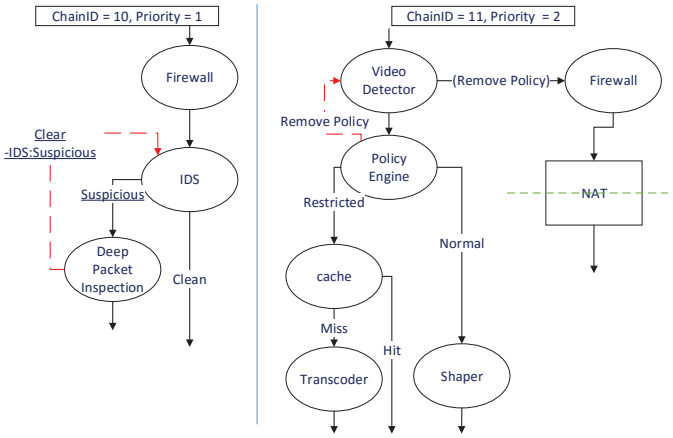

Fig. 2. A sample cluster of service chains

incrementally so that assignment of NFs to already established flows are not impacted by new incoming flows.

NF Orchestrator: The NF Orchestrator receives instantiation or migration requests from the Placement Engine (e.g., when a new service chain is deployed) and from NF Managers (e.g., when a server is becoming overloaded and a new replica NF is needed). It then deploys new VMs to run the appropriate functionality. It also handles the movement of state and syncing of the state between different replicas of an NF in conjunction with the respective NF Managers [3].

SDN Controller: The SDN Controller learns of the overall network goals from the SDNFV Application via its northbound interface, and communicates with NF Managers via its southbound interface. The protocol between the controller and the SDNFV switch needs to be enhanced, so that stateful control can be exercised by the NF Manager.

As is becoming the common practice, we favor a proactive setting up of the default forwarding ('match-action') rules by the SDN controller rather than reacting to the first packet of newly recognized flows. We also proactively set up the different alternate paths defined by the service chain for a flow that would be chosen dynamically as a result of the processing at an NF.

III. Handling Service Chaining

In this section we focus on the characteristics and special requirements of different service chains. As mentioned earlier, a service chain is a sequence of services that a flow needs to pass before leaving the network. We discuss the details of our protocol messages in Section IV.

A. Static Chains

A service chain is *static* when the sequence of NFs in the service chain is defined in advance, and it is not dependent on the output of NFs. Otherwise, a service chain is dynamic. F1 and F2 in Figure 3 are examples of static service chain and F3 is an example of dynamic service chain. While static chains may at first seem trivial to handle, in fact they may still require flow tags to properly route flows if the implementation of the service chain includes a cycle, causing a switch to be visited more than once.

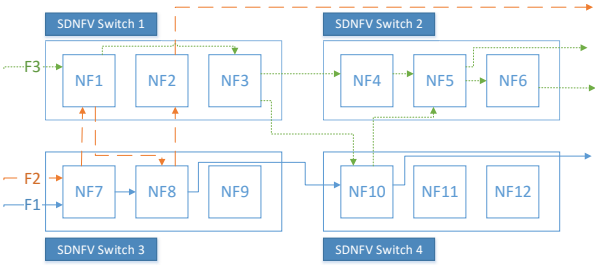

Fig. 3. A set of three service chains traversing NFs at four SDNFV Switches.

Unambiguous static flows: We first address the simplest case where the static flow is recognized unambiguously at an SDNFV switch. This occurs when the flow just traverses this switch once, or if it is traversed multiple times, each visit is distinguishable from other visits. For example the input interface may be different between different visits. In this case, the forwarding decision can be made using available information and without any need for additional tags. Note that this applies even if a flow has to be processed consecutively at multiple NFs that reside on the same SDNFV Switch as a virtual port can be leveraged to recognize the flow's state. This is shown in Figure 3 where flow $F1$ traverses three different NFs on two different SDNFV Switch hosts; on each host the NF Manager enforces the correct NF processing order.

Ambiguous static flows: When a flow has to visit the same SDNFV switch more than once and the input port is the same, there is a need to differentiate the arrivals so as to forward the packet to the NFs correctly. For example, if Flow $F2$ in Figure 3 does not have a tag (or some kind of persistent state in the NF, which we avoid) there is no way to determine which NF is supposed to examine the packet next since its n-tuple will be identical both times it arrives at the host.

To solve this problem we use a flow tag carried in the packet to indicate the stage of processing. Rather than use a unique tag for each stage (as has been done in previous work [6]), we only generate a tag for the ambiguous hops in a path where they are needed, and we interpret the tags in conjunction with the flow identified by the n-tuple in the packet header and the switch input port. This reduces the number of tags required across all the flows in the network.

Tags that assist with routing across SDNFV Switches are added to packets by the NF Manager. When the SDN Controller configures the flow table rules in the SDNFV Switch, for example to make Flow $F2$ send packets from NF8 to NF2, it will include a tagging action as part of the flow table entry. Likewise, the flow rules on the switch receiving the packet will be configured to include the tag as part of the match criteria to determine processing. We call this tag a global tag or *GTag* since it is not local to a single host, but transmitted as part of the packet.

B. Dynamic Chains

Active NFs can produce different outputs for different packets. For example, a cache may produce a Hit or a Miss as outputs. Flows may need to go to different NFs based on a

Hit or Miss. The chain containing these active NFs is called a *dynamic chain*.

In this paper, the outputs of active NFs are called *NFOutputs*. The NFOutput is written to the packet descriptor, a software data structure in the NFV platform, not in the packet itself, since it is only used locally at that host. (An approach similar to E2 [12] can be used.) The NF Manager, using rules defined in the SDNFV Application and installed by the SDN Controller, knows how to interpret this output to select one of the possible next hops. This allows NFs to provide input on how packets should be routed, but leaves the SDNFV Application in charge of determining how those outputs are interpreted. It also reduces communication overheads by eliminating excessive and unnecessary calls to the SDN Controller.

Ambiguity in routing can arise, in a manner similar to static chains. We use the following solution. The SDN Controller is aware of all the possible paths of a flow. It creates a union of all the paths of this flow, and executes an algorithm similar to what was used for a static chain, on the resulting set. Thus, we can recognize the points where there is ambiguity for a flow for which a global tag is needed. For example, in a static chain, if a flow were going from Switch 1 to Switch 3 twice, we would consider it an ambiguity, needing a tag to resolve the ambiguity. With dynamic chains, even if some packets of a flow first go from Switch 1 to Switch 3 in Path 1, while other packets of that same flow go from Switch 1 to Switch 3 in Path 2, there is still a point of ambiguity needing global tags to resolve it.

NFs may affect the routing of flows in different ways. For example, they can affect the routing at an upstream or downstream node. Also they can affect the routing temporarily or permanently. The temporary impact on routing by the output of an NF is only on the current packet. Whilst an NF's output may affect the routing of a flow permanently such that all the packets of the flow follow a different route until another output from an NF is observed. Underlined edges in Figure 2 show the permanent edges in those chains. Affecting the downstream means the route to subsequent NFs for the flow is altered. Upstream implies the routing to NFs upstream of the NF that produces the output is altered, for future packets of this flow. Dashed lines in Figure 2 indicates the upstream edges, while the other edges are downstream edges. Handling downstream and temporary actions is done in a manner similar to static chains. Here, we only focus on persistent and upstream actions.

Persistent Actions: In some cases, it is desirable for an NF to make a decision not only on a single packet, but on all the remaining packets in the flow. For example, an IDS NF may mark individual packets as being 'clean' using *temporary* actions, but when it detects something 'suspicious', it may wish to send all remaining traffic to a Deep Packet Inspection engine for further analysis. Our platform supports this through *persistent* actions, which allow an NF to produce an NFOutput that will be used by the NF Manager to adjust the flow table entry, causing all subsequent packets to bypass the NF which produced the tag and proceed directly to the next NF.

Upstream Actions: Sometimes it is necessary to change the routing upstream based on the output of an NF downstream. In this case, the NF Manager may have rules set by the SDN controller to forward information back to the SDN controller. The SDN controller uses this information to indicate that a prior NF in the chain should cancel its persistent action, or cause a new action to take place. For this, we extend the OpenFlow messaging framework to enable the SDN controller to send the name of the NF to the upstream NF Manager, and include in it the identity of the flow (IP n-tuple) and the specification of the new action (which we describe in the next section). Upstream actions can work in two ways. First they can reverse the effect of persistent output like the "Clear" edge in Figure 2). They can also simply change the path from an upstream switch (e.g., "remove policy" edge in Figure 2).

Handling header changing NFs: There are circumstances where an NF may change the packet header (e.g., network address translation (NAT)), thus requiring additional information to ensure the flow is handled appropriately. However, unlike [6], we divide the service chain into multiple segments—the first is from the entry switch to the first NF that changes the header. The flow is routed through SDNFV switches normally using the n-tuple of the packet. For the segment of the service chain after the 'header changing NF', tags are added to the packet as necessary. Note that not all of the fields of a header might be changed by the NF. For example, a NAT does not change the destination IP address and port number for an outgoing packet (towards the public Internet). Thus, given that the tag can be associated with specific fields of a packet, we associate it with the destination IP and port. Thus, the same tag value can be re-used for different destination IP and port pairs, thus again resulting in a significant reduction in the number of tags required (especially as there is likely to be diversity in the destinations of the flows traversing the NAT). Our solution can support multiple service chain segments if needed, although we expect this to be a rare situation.

IV. Protocol and interfaces

We now describe the interfaces and protocol primitives for information exchange among SDNFV's system components. We note that instead of making changes and adding new fields and matching rules to OpenFlow to support concepts such as GTag and NFOutput, it is possible to reuse unused fields in the OpenFlow protocol. This is possible because the number of unique GTags and the number of bits necessary for communicating the NFOutput needed in our proposed method is small compared to existing approaches.

A. Protocol between NF Manager and SDN controller

The OpenFlow protocol addresses a number of interactions between an SDN Controller and a network switch. These are also used in SDNFV switches for setting up flow tables. We now look at the functionality needed in the context of the SDNFV switch hosting NFs. The administrator, and thus the SDNFV Application, SDN Controller and NF Orchestrator know a priori the capabilities of the network nodes and links.

We anticipate that most of the Flowtable rules would be set proactively and the necessary NFs for flows based on defined service chains have been instantiated by the orchestrator in coordination with the SDNFV application.

Unambigious static flow routing: When routing flows from a passive NF that is part of a static service chain, the SDNFV switches use the OpenFlow tables set up by the SDN controller with the existing OpenFlow protocol primitives (e.g., based on a match-action rule using the IP 5-tuple) through SDNFV switches as in a typical switch. This would be the case also when the flow is routed through a static path on the SDNFV switch, since the flow's route is unchanged. Note, we use a different logical port at the SDNFV switch to enable forwarding to the different local NFs.

Unambigious dynamic flow routing: With active NFs, we use NFOutputs to modify the route out of an SDNFV switch. The match-action rule would be enhanced by the SDN controller as:

$$\{\text{5-tuple, NFOutput, In port}\} \rightarrow \text{outPort}$$

Thus, the NF Manager includes the NFOutput produced by the NF and uses this to assist in selecting the next stage in the service chain.

Routing ambiguous flows: Global tags or GTags are used to enable NFs to influence the path of dynamic flows and to handle a flow arriving at the same SDNFV switch multiple times. In the latter case, when an SDNFV switch is shared among the different paths of a flow, a global tag is used to carry the current 'state' (visit) of the flow to enable the switch to forward the packet to the proper next hop. Rather than having a packet with a tag be forwarded to the controller as in [6], we depend on the SDNFV switch to make the routing decision using the tag. The forwarding table rule from the SDN controller to the NF Manager is enhanced as:

$$\{\text{5-tuple, NFOutput, GTag, In port, Push, Pop}\} \rightarrow \text{outPort}$$

GTag is the global tag. When an SDNFV switch has to insert a tag header as the packet leaves the switch, the Push flag is set in the flow table. When Push is 0, an SDNFV switch simply forwards based on the tag header. The tag header is removed by the NF Manager at a switch where the flow table entry has the Pop flag set to 1, possibly based on the GTag value. GTag is added to the packet before the switch where the ambiguous visits of the packets occur. For example if the packets of a flow go from Switch 1 to Switch 5 two times in different parts of their path, routing in switch 5 is ambiguous and GTag is added to the packet on Switch 1 and it is used at Switch 5 for routing the packet to next steps. GTag may be changed to another GTag later, based on need, to address further ambiguities. The GTag is removed by the last switch in the path.

Request for a change upstream: The output of an NF that changes routing upstream or decisions of upstream NFs has to be mediated by the SDN controller. The reversal of a decision at an upstream NF is communicated by the downstream NF (through its NF Manager) to the controller, to then be used to set the flow table entry at the upstream SDNFV switch as:

$$\{\text{5-tuple, NFOutput, GTag, In port, persistent}\} \rightarrow \text{outPort}$$

The flow is identified by the 5-tuple, potentially along with GTag. The NFOutput is used to communicate information to the local (downstream node's) NF Manager. The 'persistent' flag when set to 1 is used to indicate that the persistent decision at the upstream NF (as defined by the service chain) must be changed. Otherwise, it is a request for a normal change (without the tag) at the upstream NF.

Information for header-changing NFs: When an NF changes the header of the packet (e.g., by a NAT) and routing decisions at subsequent SDNFV switches have to be set by the controller, a global tag has to be added by the NF Manager at that switch. As discussed before, the global tag (GTag) is interpreted along with the unchanged parts of the packet header's 5-tuple, to effectively re-use the global tags. The controller sets the rules, as before:

$$\{\text{5-tuple, NFOutput, GTag, In port, Push, Pop}\} \rightarrow \text{outPort}$$

B. Communication between SDNFV Application, SDN Controller and NF orchestrator

Network status to placement engine: The information about available SDNFV switches, their capacity to run different NFs, the NFs currently running on switches and the link capacities, has to be communicated by the SDN controller to the SDNFV Application, and in particular to the placement engine. This will also include current performance information, such as load and traffic statistics to enable the placement engine to make incremental placement decisions, as described in [11].

Instantiation request to orchestrator: When the placement engine needs to create a new instance of an NF or the NF Manager wants to instantiate a new NF, the orchestrator needs to be involved. The request to the orchestrator includes:

$$\{\text{SwitchID, NFName}\}$$

Orchestrating an NF Move: The SDN controller may initiate the move of an NF from one SDNFV switch to another for reasons of load balancing or to optimize the routing for a flow. For this the controller provides to the orchestrator:

$$\{\text{Src-SwitchID, Dest-SwitchID, NFName}\}$$

to cause the move of NFName from Src-SwitchID to Dest-SwitchID. The orchestrator and controller coordinate to perform a 'make-before-break' for re-routing the flow prior to removing the old NF instance.

Placement request for a flow: When a new flow arrives and the NF instances have not been created a priori, the SDN controller needs to make a placement request for the flow to the SDNFV Application (i.e., placement engine), with the flow information of the entry switch, the exit switch, and the assigned service chain. Upon creation of the NF instances by the NF orchestrator in coordination with the placement engine, the SDN controller updates the corresponding NF Managers with the flow table entries.

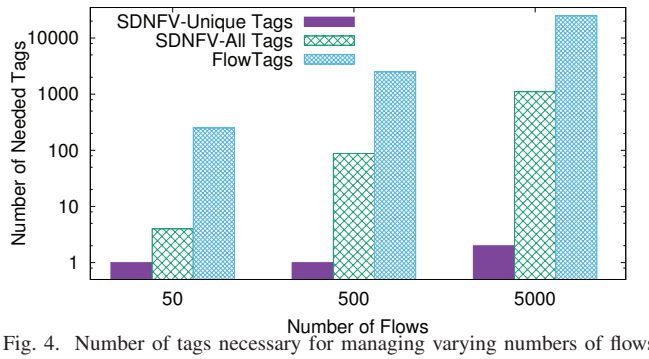

Fig. 4. Number of tags necessary for managing varying numbers of flows

V. Evaluation

We evaluate the protocol framework for the SDNFV architecture proposed here along two dimensions: the number of tags necessary network-wide and the number of messages exchanged with the SDN controller. The first is important to ensure that we use the limited number of available bits in packet headers wisely, by effectively using the global tags when it is necessary. The second is to reduce the load on the SDN controller as much as possible. Both of these seek to make our architecture scale better.

The topology for our evaluation is from Rocketfuel [13] (AS-16631) with 22 nodes and 64 links. All the flows go through a service chain of length five and start at a random node in the topology and exit at another random node. Each SDNFV switch is able to support two NFs and the placement and routing are decided by our placement engine [11]. We use a Java-based simulator to estimate the number of global tags needed.

A. Network with only NFs that don't alter headers

We first look at a network that only has NFs that do not alter the packet headers. The global tags then are used only for clearing ambiguities in the routing of the flows. Then we compare the number of tags needed in our approach with described method in FlowTags [6]. The result is shown in Figure 4. Since FlowTags needs a new tag for each flow on an NF, it needs 25000 tags for managing 5000 flows in this network when we have a static service chain of length 5. However in our approach, the number of tags needed depends on the paths of the flows. A tag is used only if an ambiguity exists in routing of the flow. As a first step, this itself reduces the number of tags needed. Furthermore, the number of unique tags needed can be reduced significantly by re-using the tag for different flows. For example if 10 flows need 2 tags each, the total number of necessary tags is 20, however we can use the same tags for all these 10 flows. Hence, the number of necessary unique tags is 2. The decision to route a flow then is based on the combination of the packet header's 5-tuple *and* the global tag. The consequence is that the number of unique tags is much smaller for SDNFV-Unique Tags, as shown in the Figure 4.

B. Network with NFs altering headers

For this case, the third NF in the service chain alters the packet header. We did two experiments. In the first one, the NF does not change the destination address. This is therefore leveraged to further reduce the number of tags necessary. In the second case, the destination and the destination port are unaltered. In our experiment, the port number was uniformly distributed across 56K port numbers. The dramatic reduction in the number of tags needed, to just 2 or 3, is shown in Table I. Finally, an important criterion for scalability is the number of messages sent to SDN controller. Our approach and FlowTags are compared in Figure 5. In SDNFV, each flow needs to contact the SDN controller once. However, with FlowTags each switch contacts the SDN controller to first get the necessary rules for that FlowTag. Further, each middlebox contacts the SDN controller twice, first to generate the tag and secondly to consume the tags for a flow. SDNFV's hierarchical control dramatically reduces the overhead on the controller.

No. of Flows	SDNFV				FlowTags
	All tags		Unique Tags		
	Dst.	Dst.+Ports	Dst.	Dst.+Ports	
50	7	2	1	1	250
500	285	8	1	1	2500
5000	4681	65	3	2	25000

TABLE I
NUMBER OF TAGS IN NETWORK WITH NFS ALTERING HEADERS

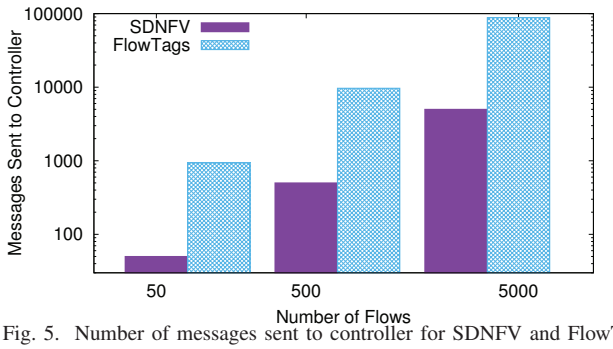

Fig. 5. Number of messages sent to controller for SDNFV and FlowTags

VI. Related Work

Recently there has been work to increase packet performance on COTS servers [9][14]. NetVM[9] uses zero-copy transfer packets from/to VM and between VMs at wire-speed. ClickOS [14] maps packets buffers to the VM's address space to reduce overhead for achieving high performance packet forwarding.

While there has been a large body of work on the core SDN protocols, we focus here on the work that examines SDN-control of middleboxes and network functions, especially for the dynamic use of middlebox functionality in the network.

SIMPLE[5] is an SDN-based policy enforcement framework for steering the traffic of middleboxes. The primary goal is to support complex routing to have flows traverse middleboxes. SIMPLE uses tags in a packet assigned to recognize a routing loop. However, SIMPLE does not address the need to support dynamic service chains and the necessary routing through them, especially when the service functions for a flow (or a class of flows) has to be updated based on the processing at a network (middlebox) function. The use of flow tags minimizes the size of the flow tables, which is a desirable characteristic we exploit in this work as well. However, our view is that tags

can serve a much wider purpose of NFs communicating to the controller or to downstream NFs the result of their processing so as to exploit the flexibility that is provided in our framework to support dynamic service chains.

Steering [15] is an SDN-based framework for dynamic routing of traffic through middleboxes. While it supports changing the service chains based on the output of the middleboxes, such an action is always mediated through the SDN controller which updates different tables in different switches. The framework is much simpler in that complex routing (e.g., loops) is not supported. Further, there is no support to dynamically place multiple replicas of an NF in a large network.

FlowTags [6] is an architecture for dynamically managing flows through middleboxes in the network. It also uses tags for complex, stateful routing of flows. A new tag is utilized for each branch of a service chain. Tags are not re-used across flows, hence they should be unique unless the flows are temporally or geographically separated. Packets of flows marked with tags are routed only based on the tags. A middlebox contacts the SDN controller to create the tag and for interpreting and consuming the tag. When the service chains are long and middleboxes have a large number of possible outputs, the number of tags required can be very large. In contrast, the autonomy we provide with our hierarchical control at the NF and NF manager levels avoids overloading the SDN controller and is thus more efficient. Moreover, by combining the tag and the full n-tuple in the flow table along with the input port and SwitchID for routing, our approach requires far fewer tags.

Slick [16] is a network programming architecture that provides the opportunity for the SDN controller and middleboxes to communicate with each other. The main feature of interest to us here is that it provide triggers for network functions to contact the SDN controller. However, the framework continues to retain the 'master-slave' relationship between the network data plane and the SDN controller, unlike our approach of providing a hierarchy of control across the components.

Besides the aforementioned works which are done in SDN field, some other works such as Network Service Header (NSH) [17] of IETF are done in other fields too. However because of lacking a centralized controller, making a global optimal decision on different aspects such as assigning flows to different instances of NFs is not possible. Moreover, the dynamicity introduced in this work is limited in comparison to our selected approach. In our approach, the flows path is completely based on the dynamic output of the NFs and even an output of an NF may change the path of next packets in upstream. In NSH classifiers are separated from network functions and they cannot affect the next packets in upstream. The last difference that we want to point out is that NSH adds a variable size header to the packet, whereas in our approach because of a limited number of bits needed for defined fields, we are able to reuse the existing unused fields in other protocols.

VII. Summary

Our SDNFV architecture seeks to achieve our vision of a dynamic and flexible network with a smarter data plane. The protocols we develop also enable NFs to update a flow's forwarding rules dynamically, subject to constraints specified by the service graph, without burdening the centralized SDN controller. This allows SDNFV to base flow management decisions on characteristics that cannot be determined at flow startup. It allows changing traffic characteristics across multiple flows to affect routing behavior, for example by detecting DDoS attacks or other anomalous flows, or network policy and load-dependent modifications. Our architecture supports a rich set of NF types that dynamically modify the path of flows both upstream and downstream and ones that change packet headers, such as NATs. SDNFV enables dynamic instantiation of NFs by an orchestrator. The SDNFV architecture uses tags for NFs to communicate their output and the state of the flow by leveraging the idea of "FlowTags" but use it in conjunction with the knowledge of the network at the SDN controller, to dramatically reduce the number of tags used network-wide. We substantially reduce the overhead on the SDN controller by taking advantage of the hierarchical control possible with the smarter data plane. The smarter data plane includes the NFs, the NF Manager and ultimately, the SDN controller, in decision making.

Acknowledgment

This work was supported in part by NSF grants CNS-1422362 and CNS-1522546.

References

[1] Timothy Wood et al. Towards a software-based network: Integrating software defined networking and network function virtualization. *IEEE Network*, May-June 2015.
[2] N. Feamster et al. The road to SDN. *Queue*, 11(12), December 2013.
[3] G. J. Aaron et.al. Opennf: Enabling innovation in network function control. SIGCOMM. ACM, 2014.
[4] European Telecommunications Standards Institute. Network functions virtualization (nfv): Architectural framework. *White Paper*, 2014.
[5] Z. A. Qazi et al. Simple-fying middlebox policy enforcement using sdn. In *SIGCOMM Computer Communication Review*, volume 43, 2013.
[6] S. K. Fayazbakhsh et al. Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags. NSDI '14.
[7] L. Rizzo. netmap: A novel framework for fast packet I/O. In *USENIX Annual Technical Conference*, 2012.
[8] Intel. Intel DPDK: Getting Started Guide. 2013.
[9] J. Hwang et al. Netvm: High performance and flexible networking using virtualization on commodity platforms. NSDI, 2014.
[10] L. Rizzo et al. Speeding up packet i/o in virtual machines. In *ANCS 2013*.
[11] A. Mohammadkhan et al. Virtual function placement and traffic steering in flexible and dynamic software defined networks. In *LANMAN 2015*.
[12] S. Palkar et al. E2: a framework for nfv applications. In *ACM SOSP*, 2015.
[13] N. Spring et al. Quantifying the causes of path inflation. In *SIGCOMM*, 2003.
[14] J. Martins et.al. Clickos and the art of network function virtualization. NSDI, pages 459–473, Seattle, WA, April 2014. USENIX Association.
[15] Y. Zhang et al. Steering: A software-defined networking for inline service chaining. In *ICNP*, 2013.
[16] B. Anwer et al. A slick control plane for network middleboxes. In *SIGCOMM*, HotSDN, 2013.
[17] P. Quinn et al. Network Service Header. https://tools.ietf.org/pdf/draft-quinn-sfc-nsh-07.pdf, 2015. [Online; accessed 12-October-2015].