Exploiting ICN for Flexible Management of Software-Defined Networks

Mayutan Arumaithurai*, Jiachen Chen*, Edo Monticelli*, Xiaoming Fu* and K. K. Ramakrishnan‡
*Institute of Computer Science, University of Göttingen, Germany.
Email: {arumaithurai,jiachen,monticelli,fu}@cs.uni-goettingen.de
‡University of California, Riverside, CA, U.S.A. Email: kk@cs.ucr.edu

ABSTRACT

Networks are becoming increasingly complex and service providers incorporate additional functionality in the network to protect, manage and improve service performance. Software Defined Networking (SDN) seeks to manage the network with the help of a (logically) centralized control plane. We observe that current SDN solutions pre-translate policy (what) into forwarding rules at specific switches (where). We argue that this choice limits the dynamicity, flexibility and reliability that a software based network could provide. Information Centric Networking (ICN) shifts the focus of networks away from being predominantly location oriented communication environments. We believe ICN can significantly improve the flexibility for network management. In this paper, we focus on one of the problems of network management – service chaining – the steering of flows through the different network functions needed, before it is delivered to the destination. We propose Function-Centric Service Chaining (FCSC), a solution that exploits ICN to provide flexibility in managing networks that utilize virtualization to dynamically place functions in the network as required. We use a real-world topology to compare the performance of FCSC and a more "traditional" SDN solution. We show that FCSC reacts to failures with fewer packet drops, adapts to new middleboxes more quickly, and maintains less state in the network.

Categories and Subject Descriptors

C.2.3 [**Network Operations**]: Network Management

General Terms

Design; Management

Keywords

ICN; Service Chaining; Network Management; SDN; Network Function Virtualizaion; Middlebox

1. INTRODUCTION

Service provider networks (and networks in general) are becoming increasingly complex. Both network operators and users require various additional functionalities in the network for management and processing of data flows. Software Defined Networking (SDN) aims to manage the network and the functions provided by separating the control plane from the data plane. The SDN controller(s) possess a global view of the network and can therefore simplify the network management as compared to the traditional distributed architectures typical of the Internet. However, even in an SDN environment, management logic ("what") is intricately coupled with the node location ("where"). With the use of virtualization and the prevalence of mobility the location of a particular function in the network may no longer be fixed. We envision that the performance of SDN would be further improved by incorporating the ideas of information-centricity that decouple the location of a particular network function instance from the identity of the function it provides. In this work, we make a first attempt by incorporating Information Centric capabilities into a common and important problem of network management – Service Chaining.

The need to perform additional processing of packets of a data flow in the network before it is delivered to the destination has become an integral element of providing Internet services. These functions include the modification of the packet header (*e.g.*, NAT, proxy), discard packets (*e.g.*, firewall), collection of statistical information (*e.g.*, Deep Packet Inspection (DPI)) or even the modification of the payload (*e.g.*, optimization and compression). They are provided in the form of *Middleboxes* [9, 19, 40] for policy control, security and performance optimization. The middleboxes have to be resident on the path of a flow, which implies that the traffic has to be deviated from its "natural" IP shortest path and forced through the middleboxes. We use the term *Service Chaining* to describe the action of steering packets through these middleboxes. For example, a network operator might require flows that access dynamic web pages such as Facebook, Twitter, FourSquare, Google Instant, or MyYahoo to go through middleboxes like Content Delivery Network (CDN), Dynamic Site Accelerator (DSA [1]), TCP optimization over tunnel, *etc.*, in order to improve the perceived user experience [2].

The limited presence of middleboxes at specific locations in the network often results in sub-optimal routing and lower performance (*e.g.*, increased latency, lower throughput, *etc.*). This is especially true in environments like cellular networks [15, 16] where middlebox functions are restricted to be in the

"Network Data Center" and thus have a significant impact on latency. The recent introduction of Network Function Virtualization (NFV) [14, 20] promises to make it easier to dynamically and flexibly deploy middleboxes. NFV allows for middlebox functions to be virtualized and therefore be present in greater number and positioned on-demand. We envisage network service providers will increasingly adopt NFV to provide network resident functionality, not only for reducing CAPEX but also for offering more flexibility to customers who would like customized processing of their packets. However, managing such a network of dynamically placed functions can be much more complex. Current routing protocols deployed in IP networks constrain how packets can be deviated from well-defined path (*e.g.*, shortest path) and thus cannot take full advantage of the great flexibility offered by NFV.

Recently proposed solutions for Service Chaining in Software Defined Networking (SDN) [22, 33, 45] attempt to perform Network Management by making use of a (logically) centralized controller that has the capability to setup flow-based forwarding rules on the switches [18, 27] of the desired path. Such solutions provide greater control over the network in order to steer packets of a flow more flexibly, without being constrained by traditional routing such as OSPF and BGP. But the controller has to keep track of the status of the middleboxes and the network.

We argue that the existing approaches have a common issue of unnecessarily coupling the routing with the policy. *I.e.*, when an SDN controller decides the *functions* a flow needs, it also decides the *path* the flow has to go through and setup *state* on the intermediate switches. These solutions have limitations in *scalability*, *dynamicity* and *flexibility* and therefore have difficulty in adapting to the requirements of a large scale, dynamically changing middlebox set supported by NFV (see §3.3 for detailed descriptions).

Information-Centric Network (ICN [4,21,44]) is a new networking paradigm that introduces ContentNames to decouple the user interests from data location. Following this line of thinking, we present *Function-Centric Service Chaining* (FCSC), a novel approach that decouples the *functions* a flow needs from the *location* of network function instances (and thus routing) via a naming layer (see Fig. 1). Such a decoupling facilitates the dynamic modification of the functions needed by a flow on the controller or the middleboxes (*e.g.*, DPI, load balancer). This also enables switches to dynamically detect the load (popularity) of a certain function and accordingly instantiate/dispose of network function instances (co-resident with the switch or on some other node). The enroute function-based routing allows more dynamic use of the newly created instances and faster recovery from node/link failures. FCSC intrinsically supports the presence of multiple instances for the same functionality and can perform network-layer load-balancing among these nodes at any time. By placing the flow state in the packet header, FCSC helps to reduce the amount of state stored in the network and results in much better scalability compared to the per-flow state solutions like SDN. FCSC is therefore able to provide a highly dynamic and adaptive Service Chaining capability and effectively exploit the promise of NFV in the software-based network of the future.

The key contributions of this work are:

- We exploit the combination of ICN with SDN to meet the dynamic requirements of service chaining. We propose FCSC, a scalable and flexible architecture, that clearly separates the policy (required functions) from the routing by introducing a light-weight (function) naming layer.
- With the help of varying number of flows and dynamic creation/deletion of virtual service instances on a synthetic and a real-world Rocketfuel topology, we show how FCSC compliments the current SDN solution in terms of network state amount, packet drop rate on node failure and overall latency.

The rest of the paper is organized as follows: §2 discusses previous work on service chaining and information-centric network; §3 provides detailed description on the service chaining scenario and the problems with the existing solutions. §4 describes the design rational of FCSC and §5 details upon the solution. §6 illustrates simulation results and conclusion is inferred in §7.

2. RELATED WORK

In this section, we briefly present existing work on service chaining, then present an overview of ICN and finally present existing work that involves both SDN and ICN.

2.1 Existing Solutions for Service Chaining

Existing Service Chaining solutions can be broadly classified into 3 classes: indirection-based, policy-based and SDN-based.

2.1.1 Indirection-based Service Chaining:

Several proposals for service chaining regard *indirection* as an indispensable element for achieving high flexibility to support various scenarios including node mobility, caching and anycast. Works in [6, 40] propose an architectural modification to TCP/IP networks in order to allow further indirection than what is supported by DNS, allowing simple integration of middleboxes into the TCP/IP architecture. [6] highlights the problem of having an IP address – location-dependent element – as the identifier of end hosts, and proposes the introduction of several levels of indirection. Other similar works include [10, 17, 30, 31, 37, 39]. Unfortunately, these solutions rely on predetermined nodes that provide the service, thus becoming inflexible to react to node failure as well as new instances of middlebox functionality.

2.1.2 Policy-Based Routing (PBR):

Cisco's policy-based approach [13] allows the administrator to specify adjunctive rules for routing, that are selectively applied depending on the traffic characteristics (*e.g.*, IP 5-tuple, rate, *etc.*). Since the rules must be manually configured on each PBR router, the solution scales poorly and cannot dynamically react to network condition changes.

2.1.3 SDN-based Service Chaining:

Several solutions have been presented that leverage SDN [22, 33, 45]. The general idea is to have a logically central controller that has a comprehensive view of the administered network portion and of the networking elements present. This controller can determine the best route for each flow that traverses the network and can take into consideration the potential need for this flow to go through one or more middleboxes. To make its decision effective, the controller must add forwarding rules to the involved switches, instructing them on the new next hop for each flow that deviate from its standard IP path.

2.2 Information Centric Networking

Information Centric Networking (ICN) has been actively studied in recent years [4, 5, 11, 12, 21, 24]. ICN shifts the focus of the network from node location (IP, MAC, *etc.*) to data names. Such design enables name-based routing which forwards the requests of a specific name towards a best source of the data in terms of latency, available bandwidth, source load and *etc.* Named-Data Networking NDN [4, 21] is one of the popular ICN solutions. NDN uses human-readable, hierarchical names such as `/thisroom/projector` or `/icn/papers/FCSC. pdf`. The forwarding engines perform the longest-prefix matching in the FIB to find the next-hop router closer to the data provider. FCSC adopts the idea of ICN since the naming layer focuses more on the name of the functions rather than the location of the function instances. It is implemented on a model similar to NDN (naming and routing) but with some changes (no Pending Interest Table or reverse-path forwarding).

2.3 Works that Combine ICN and SDN

There are several works that try to explore the potential for combining ICN and SDN [29, 34, 38]. But most of these works use SDN technology to enable incremental deployment of ICN. To the best of our knowledge, this is the first work that tries to improve the performance of SDN via information-centric concept (ICN).

3. SCENARIO DESCRIPTION AND PROBLEM STATEMENT

In this section, we describe the scenarios we envision of how network resident functionality of middleboxes could be utilized and point out the shortcomings of the state-of-art SDN solutions. We will use these as the basis to demonstrate the benefits of our proposed approach.

3.1 Service Chaining Scenario

An Autonomous System (AS), for example an IP network, data center or an information centric network, is typically composed of many edge routers and a set of core routers/switches. Packets from users enter this AS from one of the edge routers (Ingress). These packets categorized into flows (either by 5-tuple in IP or “Interest” prefix in ICN) need to go through a specified set of functions in the core in *a particular order*, as required by policy. The functions may include Deep Packet Inspection (DPI), policy, QoS, Network Address Translation (NAT), Dynamic Site Accelator (DSA), proxying, transparent caching, accounting and logging *etc.* It is also possible that a subset of these functions may in fact be provided by third parties, and possibly in a cloud-resident platform [35].

3.2 Detailed Requirements

With the growth of the middleboxes and the network traffic, we envision that an efficient service chaining network should meet the following requirements:

3.2.1 Flexibility:

The outcome of packet processing by a middlebox may change the set of function(s) to be applied on subsequent packets of the flow. *E.g.*, after a packet goes through DPI, the policy or algorithm may determine the need for additional network resident functions like intrusion detection, logging, *etc.*, to be applied on the flow. It is also possible that functions can reduce/replace the functions a flow needs to go through. *E.g.*, after observing a set of packets in a flow, the DPI can decide to remove the virus scan and even DPI itself from the function list. Therefore, even if a set of apriori service functions were specified, they might be changed during the lifetime of the flow. An efficient service chaining network should support such changes in a flexible way – the middleboxes should be able to determine the functions of a flow *themselves* and the changes should take effect *immediately.*

3.2.2 Dynamicity:

The advent of NFV allows for network resident middleboxes to dynamically incorporate (additional) functionality by spinning up additional virtual machines on demand. *E.g.*, if there are many more flows that require firewall functionality but fewer flows require DPI functionality, the network manager should be able to instantiate more firewall nodes and reduce the number of DPI nodes. Since more functions are running on virtualized platforms, these functions can potentially be placed anywhere in the network instead of on only a selected set of predefined nodes. This requires the network to be able to apply these changes as soon as possible while keeping the communication cost low. For the functions having multiple instances, the network should also be able to balance the load on these instances to optimize performance.

3.2.3 Scalability:

The scalability requirement comes in three dimensions: the number of functions, the number of flows and the size of the network. With more customized services provided to network users, it is envisioned that there would be an increase in the number of network functions available. For the networks that adopt NFV, the number of instances of network functions can also grow to be large. A scalable service chaining solution should not limit the number of users/flows, the number of functions a flow should traverse, or the size of the network due to the response latency or the number of states stored in the network.

3.2.4 Reliability:

A productive service chaining solution should also take reliability into consideration. The solution should be able to dynamically react to the node (middleboxes, switches or controllers) failures and the link failures within a threshold. As suggested by [32], the recovery time of a failure should be within 10s of milliseconds.

3.3 Limitations of Existing SDN Solutions

Current SDN solutions [22,33,45] perform better than policy based routing (PBR) and indirection based solutions. However, they are still not able to meet the requirements mentioned above, because:

Flexibility: When a middlebox like DPI needs to change the functions a flow requires, it has to rely on the controller to build a new path that goes through a certain instance of each of these functions. This results in extra control overhead in both communication and latency for every flow whenever the set of functions are changed. This is not desirable since the controller in SDN design is supposed to *generate* the rules but not be involved in the real-time handling of packets [43].

Dynamicity: It is difficult for SDN controllers to perform real-time decisions on the path of a flow to balance the load in the network and on the function instances. The problem will become more severe when the number of flows grows and NFV enables more dynamic instantiation/disposition of function instances.

Scalability: SDN solutions place rules for every flow on the switches. The number of rules stored in the network is proportional to the number of flows, the functions the flows require and the size of the network. It is very difficult to scale when the network has larger number of flows or the network itself grows larger.

Reliability: When a middlebox or a link fails, the switches in the existing SDN solutions have to rely on the central controller to build a new path for the flow. This increases the convergence time while dealing with such failures and might violate the typical 30-50ms convergence time target requirement typical in a large provider networks. Alternatively, the controller has to setup backup paths proportional to the number of hops for every flow to ensure quick convergence time. But this exacerbates the scalability problem.

4. FCSC OVERVIEW

In this section, we start by reasoning the design choices we have made and then describe the whole architecture based on the design choices. Design details will be provided in §5.

4.1 Design Rationale

To achieve the requirements of flexibility, dynamicity and reliability as described above, we propose to add a naming layer (similar to ICN) to the current SDN architecture. Moreover, to improve the scalability of the system, we choose to put flow-state in the packet header rather than in the switches. But our solution is still backwards compatible with existing SDN-based service chaining solutions.

4.1.1 Naming Layer

ICN provides flexibility to users – they only need to request the network with *what* they want rather than *where* that data might reside. Such a shift in the focus of the network also provides better dynamicity and reliability – a request can go to *any* of a set of possible data providers/caches in the network.

We find a strong similarity between the fundamental needs that drive service chaining and the capabilities offered by ICN. Middleboxes that need to change the function list of flow (*e.g.*, DPI) require flexibility – they only need to care about the *functions* the flow requires rather than asking the SDN controller to build a path to *where* the flow should go through. While forwarding a packet, the network can forward the packet to *any* of the instances that provides the same function. This allows the dynamic adoption of new function instances and can also help fast recovery when a function instance/link fails.

To achieve high performance (line-speed forwarding in the network), we primarily incorporate the hierarchical naming capabilities of an ICN environment like NDN, to represent the function list and the longest-prefix matching in the FIB to forward packets. The reverse-path forwarding (PIT) and caching (Content Store) capabilities that are used in an ICN for information distribution is not key to this solution. According to [36, 42], line-speed (hashed) name forwarding is achievable with existing hardware.

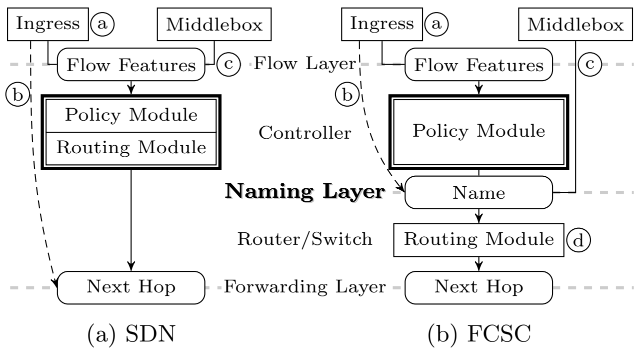


Figure 1: Architectural design of FCSC *vs.* SDN.

4.1.2 Flow State in the Packet Header

We note that the solutions of existing SDN-based service chaining incorporate flow state in the switches – the switches maintain state on how to forward a specific packet based on the 5-tuple (or other header features) of that packet. Such a design results in the number of rules in the network to be proportional to the number of flows and the number of functions these flows require. It does not scale well with the growth of clients (flows) and the adoption of new functions in the network.

Therefore, we choose to put the flow state – the functions the packet still need to traverse through – in the packet header. The function list of a packet (in the form of an ICN name) is tagged to the packet header when it enters the network. After traversing through a middlebox, the name of the applied function is removed from the header and the network will forward the packet to the next function it requires.

In our solution, the network only needs to maintain forwarding information on a per-function basis rather than a per-flow basis. The amount of state stored is therefore proportional to the functions in the network but not the flows, and thus our solution can scale much better than existing solutions.

4.1.3 Compatibility with existing SDN solutions

Network management, as exemplified by service chaining, needs SDN for flexible placement of functions and more powerful routing, and achieves this because it has a (logically) centralized view of the whole network. The purpose of our solution is not to replace these ideas in SDN solutions or to remove the existence of the (logically) central controllers, but to make them more flexible by the modifications proposed in our paper: namely the naming layer and the use of flow state in the packet header. Our solution can also be backwards compatible with the existing SDN solutions by naming all the intermediate switches (in the form of IP or MAC addresses) and setup a separate forwarding table on every switch in a hop-by hop manner. But that will result in the loss of the benefits mentioned above.

4.2 Architecture Description

Fig. 1 illustrates the logical separation of the architecture of FCSC compared to existing SDN solutions. As described above, we add a *Naming Layer* in the architecture that separates the *policy* module (the module that manages *what* functions should be applied to a flow) from the *routing* module (the module that manages *where* the function instances reside). The representation of the naming layer (the function list a packet should go through) resides in the packet header to scale the network better. To help understand the figure, we describe the differences between the two solutions in the following 4 scenarios (following the marking on the figure):

a. Flow initiation:

In SDN, on seeing a new flow, the ingress sends the flow feature (*e.g.*, 5-tuple) to the controller. In this case, the controller has the *function* module and *routing* module coupled. The controller determines the result, a set of forwarding rules (*e.g.*, 5-tuple$\mapsto$NextHop) that are then incorporated on different switches on the path.

In FCSC, the controller determines which functions the flow needs and return the result only to the ingress. The ingress then tags the packets of the flow with the function list. On seeing the functions carried in the packet header, switches in the network will look into their FIB (in the form of Function$\mapsto$NextHop) and decide the outgoing "face" of the packet. The FIB of the switches are controlled by the *Routing Module*. The *Routing Module* can either be distributed (*e.g.*, OSPFN [41], IS-IS [8]) or logically centralized (*e.g.*, SDN).

b. Proactive rules:

The controller can also setup wildcard proactive rules on every ingress. An SDN controller essentially has to build a path for every flow from each ingress. This increases complexity since almost every edge router can be an ingress and there might exist $O(N^2)$ *src-dst* paths where N is the number of edge routers even without considering different paths for a same *src-dst* pair.

In FCSC, the controller only needs to flood the wild card rules (FlowFeature$\mapsto$FunctionList) to all the ingress. The core of the network does not have to keep any state on a per flow basis any more.

c. Policy change by middleboxes:

When certain middleboxes need to change the policy (function list) of a flow, in SDN the middleboxes have to request the controller to build a new path for the flow. This might result in even more state in the network and also higher latency, just like what we would experience at the beginning of a flow.

FCSC allows middleboxes to determine the new policy, without having to request the controller to change forwarding rules at specific switches. These middleboxes change the function list in the packet header and the network will forward it towards the next middlebox automatically. Additionally, they may notify the ingress to change the function list for the future packets of the flow. This solution therefore only requires a change in the state of the packet header and the ingress as opposed to every switch on the old and new path. Therefore, unlike existing solutions, it does not require additional set up time while a middlebox like DPI tries to modify the policy. It is thus able to quickly enforce the policy on the newly arriving packets.

d. Dynamic routing:

With existing SDN solutions, the functions a flow requires is represented by the path it follows. Whenever a middlebox

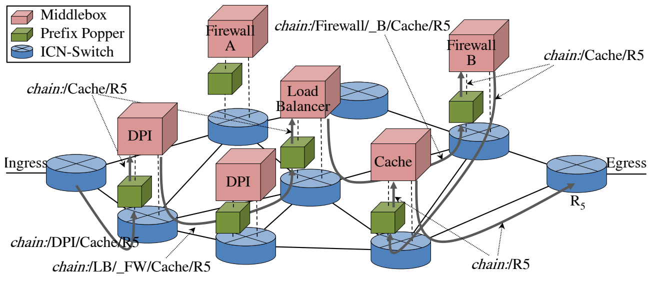


Figure 2: **Example of the name changing of a packet in FCSC.**

fails, the failure notification has to be reported to the controller before the new path that includes another instance of the function is built for the flow. Approach in [32] pre-computes backup paths to shorten the recovery time on such failures, but this results in exponential complexity due to the permutations and function combinations. When a new instance of a function is adopted, it is also difficult to use it for existing flows for purposes of load balancing.

FCSC separates the routing of flows from the policy. The switches can decide (an alternate) outgoing face based on its own FIB. This shortens the response time for node/link failures and can make use of new instances of functions on the fly (as long as the FIB is updated based on the *Routing module*).

5. DESIGN DETAILS

In this section, we describe in detail how we design the architecture to ensure a highly efficient and scalable service delivery network.

5.1 Naming Strategy

We extend the ICN principle of *naming content* to *naming function*. Every instance that provides the same function is referred to by the same name, *e.g.*, `/DPI`, `/Firewall`, *etc.*

When the network policy requires a flow to go through a sequence of functions, the policy executor (the controller or the ingress) will encapsulate each packet of the flow with a header containing a name that represents the sequence of functions to be executed, in a FCFS manner. *E.g.*, a packet header with name *chain:*`/DPI/Cache/R5` implies that DPI and cache function must be applied to that packet before it exits the network from the egress R_5. Here, we use the scheme identifier (as per URI Generic Syntax [7]) "*chain*" to represent the packets for service chaining. We can use other identifies like "*monitor*", "*ctrl*", *etc.*, to represent packets meant for other purposes (*e.g.* monitoring and controlling, *etc.*)

The switch fronting a middlebox (SxFM) will pop the first part of name (prefix) in the packet header before it forwards the packet to a middlebox function associated with it. Some policy nodes can also change the name to redirect the packet towards other functions.

Prefix popping is a simple and stateless task. It can be separated from the switching and the middlebox functions. If necessary, we can include a designated hardware component for acceleration,or instantiate a virtual prefix popping function, on the SxFM (although we believe it is a simple task). Since it is a stateless task, the SxFM can also have multiple of these components (either hardware-based or software-based) that run in parallel to ensure line-speed forwarding is achieved on the SxFM.

Fig. 2 illustrates the lifetime of a packet in our network. The ingress encapsulates the incoming packet with a header *chain:*`/DPI/Cache/R5` as desired by policy. The network forwards the packet to the SxFM of the "best" DPI function (in terms of relative location, latency or other criteria). The prefix `/DPI` is removed when passing through the prefix popping function (represented by a green box) before entering the DPI box. The DPI function decides the packet should also go through a firewall and since there is a load balancer for the two firewalls in the network, the DPI adds a prefix `/LB/_FW` to the header. The prefix is replaced with `/Firewall/_B` by the load balancer since it decides that the flow should go through Firewall B. The remaining prefixes are popped one by one when going through the Firewall B and Cache. On reaching R_5, the egress sees its own name and therefore it decapsulates the packet and forwards the original packet out of the network.

5.2 Routing Strategy

Middleboxes need to advertise their existence before they can be used by the flows. A middlebox offering a certain service (*e.g.*, Firewall) advertises the name of the service as prefix (*e.g.*, `/Firewall`). Packets whose names have the prefix `/Firewall` can be routed towards this middlebox as a consequence of the normal name based routing. A packet in FCSC is only forwarded to one middlebox even when multiple middleboxes exist for the same function (prefix). The intermediate switches can monitor the popularity of a function based on these prefixes, and they can create additional instances where needed with NFV support.

The decision of *which* exact instance of the function a packet should traverse is determined by the routing module. FCSC does not limit the routing module that can be adopted. To better support different routing strategies, we provide a simple standard interface for the routing module to control the forwarding decision. We add a "cost" field in the FIB and thus the data structure looks like `Function`$\mapsto$`{NextHop, Cost}`. If multiple "NextHop"s exist for the same function, the switch will always forward the packet to the next hop with the lowest cost. The routing module can have different interpretations of the "cost", *e.g.*, link latency, policy, energy/work-load considerations, *etc.*

The choice of the routing scheme can affect the dynamicity and reliability of the whole solution. We discuss the benefits and issues of some possible routing solutions – generally categorized into centralized and distributed schemes. But, it should be noted that regardless of which routing scheme is used, FCSC is able to achieve better scalability since we only maintain function state.

Centralized/SDN solutions (*e.g.*, [33]) have better control over the node state including what is maintained at switches and middleboxes. They provide more flexible control in determining where middleboxes should be placed and monitoring node state. Routing based on names of function instances may offer better real-time load balancing capability, faster failure recovery and utilization of new function instances.

Distributed routing solutions (*e.g.*, [41]) allow every switch to have the intelligence to make routing decisions on its own. This would enable dynamicity in routing to a newly created instance or avoiding a failed link/middlebox. But these solutions might incur higher control overhead for synchronizing the network state on every switch, especially when automatic load balancing is required for different instances of the same function.

We realize that both the centralized and the distributed routing methods face the difficulty of achieving real-time load balancing similar to [3]. A compromise is for the network to incorporate a load balancer to dynamically distribute the load on the servers/middleboxes that have the same functionality. Our architecture can fully support such load balancers (see the example in §5.1).

5.3 Stateful Middleboxes

There might be some functions in the network that need to maintain state. In such a case, all the packets of a flow should go through the same instance, even though they may not care which actual instance they might use. This implies that the different instances for the same function cannot be treated equivalently. The two firewalls in Fig. 2 could be an example of this kind. FCSC adopts the hierarchical name in ICN to meet this requirement. Instead of using the same name, the multiple instances share a common prefix (function name), but they have function-level unique ID. *E.g.*, the firewalls in Fig. 2 are called `/Firewall/_A` and `/Firewall/_B` respectively.

While advertising the prefix, the middleboxes advertise the whole name instead of the function name itself. If a packet can go through any of the instances for a function, it just puts the function name in the header (*e.g.*, *chain:* `/Firewall/Cache`). Otherwise, it will use the full name (*e.g.*, *chain:*`/Firewall/_A/Cache`). This can be determined by the policy executor or on the fly. ICN switches perform the longest-prefix match, and therefore the packet can be forwarded to the required function instance, if specified. While popping the name from the packet header, we can also perform "longest-prefix popping" of the full instance name from the name list. To avoid ambiguity, this solution requires that the instance ID space should not overlap the function name space. *E.g.*, Firewall B pops both the `/Firewall` and `/_B` prefixes from the packet header in Fig. 2.

For the functions that require visibility of the bidirectional packets of a flow, the policy module can also specify the function instance via its full name and create a function (instance) list in the reverse order. *E.g.*, if say the firewall function (only) requires packets from both directions, the policy layer can create name *chain:*`/DPI/Firewall/_A/Cache` for one direction and *chain:*`/Cache/Firewall/_A/DPI` for the other.

5.4 Security

Security is another big concern in service chaining. The users of the network should not have any chance to infiltrate the network and steer the packets through paths that are not allowed by the policy.

FCSC encapsulates each packet at the point they enter the network and decapsulates them on egress. Therefore, the service provider network is essentially transparent to users. We do not provide any user interfaces to the clients and therefore there should be no way a user can interact with the encapsulation/decapsulation function or the other function modules in the network. No client of FCSC can violate this policy by altering the packet in some way.

6. EVALUATION

In this section, we evaluate on a custom simulator (widely used in previous works [12] [11]) the dynamicity, reliability and scalability of FCSC with a distributed routing module and compare it to a relatively simple (physically) centralized SDN solution that is conceptually similar to [33, 45]. These approaches use the basic OpenFlow protocol constructs that is a controller interacting with network forwarding elements [28].

We recognize that there are approaches for decentralized SDN solutions like [23, 43], but the results from [25] show that the inconsistent SDN control state can *significantly* degrade performance of logically centralized control applications that are agnostic of the underlying state distribution. In addition, the communication overhead for keeping all the controllers synchronized has to be addressed. Moreover, even if there exists multiple controllers, it is still fair to assume that each of these controllers is in charge of a set of routers (a portion of the network). Therefore, the centralized solution we use here can also be viewed as such a portion.

We first demonstrate the benefits of FCSC on a small synthetic topology (shown in Fig. 3a). Subsequently, we run a large-scale simulation on a real world topology to evaluate the scalability and the efficiency of our solution in a more realistic environment.

6.1 Study of FCSC Behavior

Fig. 3a shows a simple topology with multiple middleboxes. R_1-R_6 are FCSC capable switches. N_1-N_4 and DPI provide functions A and B and DPI as noted in the figure. Src and Dst are the source and destination of a flow of interest. $Ctrl$ is the central controller in an SDN solution. The link latency between switches is $2ms$ and the latency between switches and the end-systems (middlebox, src, dst, control) is $10ms$. The bandwidth on the link is $100Mbps$ (large enough to support the flow). The processing latency on all the middleboxes (including $Ctrl$) is $1ms$, or $1000pps$ (packets per second). The sending rate at src is also $1000pps$.

We use several scenarios to compare the behavior of FCSC with the simple SDN solution:

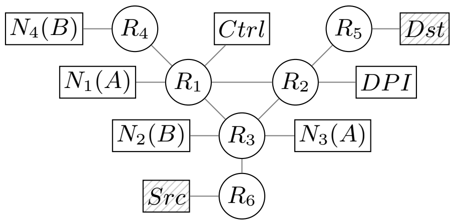
6.1.1 Proactive rules for flow initiation:

We first compare the initiation phase for both solutions. We compare FCSC with an SDN solution without proactive rules set up in the switches.

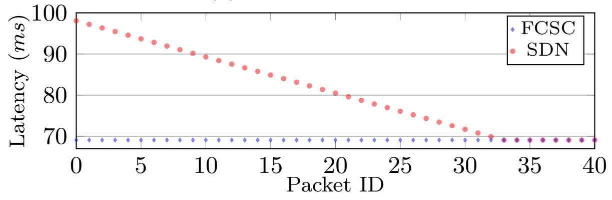
Fig. 3b shows the overall latency (the amount of time spent from Src to Dst in the network) of every packet in the initiation phase of a flow that requires DPI and B function. Because FCSC does not make a request to the central controller, it can achieve significantly lower latency for the first 30 packets of a flow compared to the SDN solution. This reduction may be critical for small flows that require timely processing of middlebox functions (*e.g.*, games).

We recognize that with proactive rules set up in the switches, SDN solutions can achieve lower latencies, as good as FCSC. However, we believe it still requires a lot more effort to precalculate the paths for different permutations as compared to our solution.

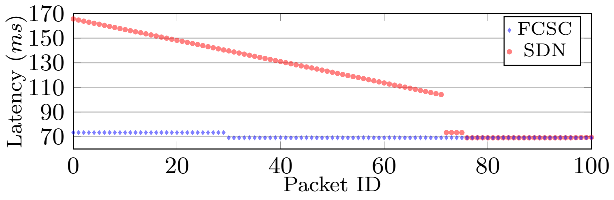
6.1.2 Dynamic Policy change on Middleboxes:



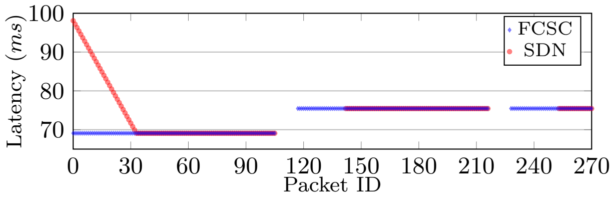
(a) Demo Topology



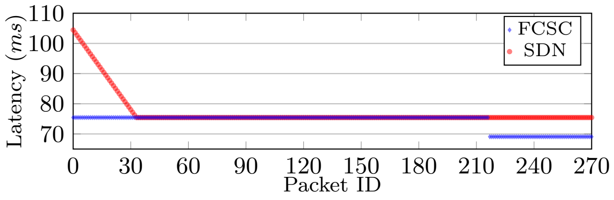
(b) Flow initiation using proactive rules



(c) Dynamic function modification by DPI



(d) Dynamic failure recovery



(e) Dynamic adaptation to new instances

Figure 3: Behavior of FCSC: Topology and Results.

In the second experiment, the default policy requires a flow to go through DPI and B functions (represented as a *chain:*`/DPI/B`). But the DPI then decides to add function A and also removes itself from the function list (the name then becomes *chain:*`/A/B`) after it examines the first packet of the flow (this is a typical dynamic function processing required in service provider networks for actions such as mobile video processing etc.). In the SDN solution, DPI requests $Ctrl$ to create a new path for the flow and block the packets from being forwarded before the new path is built. In contrast, with FCSC, DPI directly renames the packets that continue to arrive and notifies the ingress (R_6) to change the policy. There is no need to block the packets at the DPI.

Fig. 3c shows the latency of the first 90 packets in the flow. In FCSC, only the first 29 packets go through DPI with less than $75ms$ overall latency . However, in SDN, 73 packets flow into DPI before $Ctrl$ can setup a new rule for the later packets. The overall latency of the first packet grows up to $165ms$. Another 4 packets experienced a loop since the rules are not setup atomically.

From the result, we see that FCSC responds faster to the dynamic policy changes, this results in lower packet latency and also lower DPI load (process & modify 29 headers *vs.* process & buffer 73 packets in this example).

6.1.3 Dynamic failure recovery:

In the third trace, the flow is required to go through functions A and B (represented as *chain:*`/A/B`). The initial shortest path routing in both SDN and FCSC choose to go through N_3 for A and N_2 for B. We dispose N_3 at $150s$ and N_2 at $240s$ and see the packet loss and recovery time in FCSC and SDN.

Fig. 3d shows the overall latency of the packets that reach Dst. The packets in the outgoing buffer of SxFM and on the link to a failed middlebox ($\sim$ 10 pkts) have to be dropped in both solutions. Since the intermediate switches can redirect the packets without going to the central controller, FCSC can have around 25 more successful deliveries every time a node fails. This value can increase when the network is becoming more complex or the controller is farther away from the failure.

6.1.4 Dynamic adaption to new instances:

The last trace has a flow that goes through functions A and B. At the beginning of the trace, only N_1 and N_4 are instantiated. N_3 is created at $150s$ and N_2 is created at $240s$.

Fig. 3e shows the overall latency of the packets in the flow. The SDN solution does not modify the path of the ongoing flows due to the complexity (the problem is similar to a warehouse location problem and is NP-complete). Therefore, the latency does not change even when the new instances are created for the functions. FCSC enables the middleboxes to advertise their function prefix to the network and the switches can redirect flows based on that information and therefore this solution can adapt to the new instances and the latency is lowered. Note that when N_3 is instantiated at $150s$, the flow is redirected to N_3 for the shorter distance from the ingress, but the overall latency is not changed because the same number of hops are traversed. However, adding N_2 reduces the latency in FCSC as the packets do not have to flow to N_4.

6.2 Large Scale Evaluation

We adopt a slightly modified Rocketfuel topology [26] (Exodus, AS-3967, see Fig. 5) to evaluate FCSC in a real world environment. The 18 cities present in this topology is used as the core network. The latency between every pair of these core switches is determined by the average of the latency on the links between the two cities. We eventually get 30 links with latency ranging from $2ms$ to $21ms$ and a mean value of $6.6ms$. The latency between the core switches and the end-hosts, middleboxes and the controller is set to $6ms$. Since the original topology only contains latency information, we assign $100Mbps$ bandwidth to all the links.

We assume that there exist 11 different functions in the network. One of them is unique in that the DPI-like function rewrites the function list as needed. The flows belong to one of the 100 different applications. Every application requires

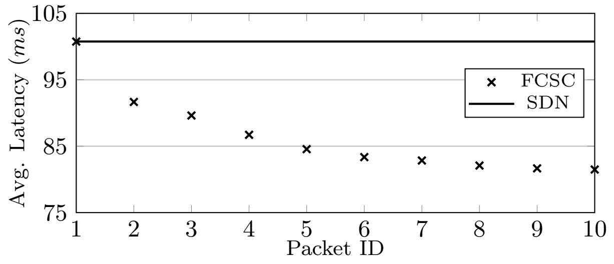


Figure 4: Benefit of increasing # of instances per function.

a range of different network functions varying in number from 1 to 4. DPI can dynamically change the functions a flow that needs.

6.2.1 Varying number of function instances

We first study the benefits of adopting FCSC in a network that dynamically creates virtual instances at random switches. We study 100 long-lasting ($5min$) flows starting at $0s$ with different sending rate (ranging from $120kbps$ to $1.05Mbps$).

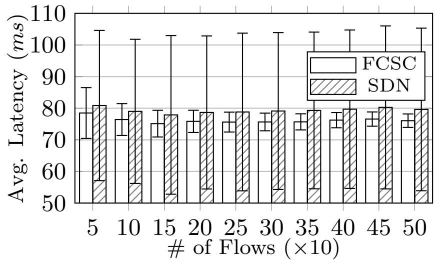
At the beginning of the simulation, we have 1 instance for each function initialized. Then, we start a new randomly located instance for each function every half a minute until the maximum number of instances is reached (the maximum number of instances for each run is shown in the x-axis in Fig. 4). In the first run, only the initial middleboxes are used for the entire 5 minutes; in the second run, in addition to the initial functions, another one instance per available function is randomly placed on a switch in the network at time $30s$ and lasts until the end of the simulation. In the third run, a third instance is put into the network at time $60s$ and so on.

Fig. 4 shows the average latency vs. the number of instances eventually initiated for that simulation run. We see that FCSC can automatically adapt by making use of new instances that are closer, even for the ongoing flows and the average latency drops from $100.75ms$ to $91.66ms$ when adding a second instance. The latency is further reduced to around $85ms$ when we have 5 instances created. This is beneficial for long flows compared to the alternative SDN solution where the ongoing flows are unaffected by dynamic addition of functions in the network, unless the controller resets the rules.

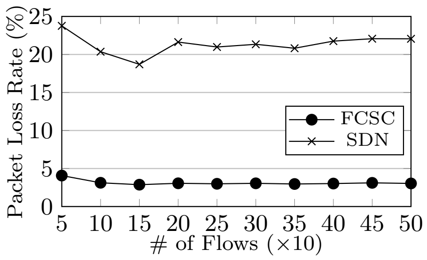
The results illustrate that FCSC is able to seamlessly take advantage of new instances of virtual middleboxes that have the same functionality, even when the network is not overloaded. We can also observe that the higher the number of instances, the lower the incremental benefit. In our scenario consisting of 18 switches, more than 8 instances do not yield additional benefit. Ignoring the absolute numbers, since it is topology dependent, one can nevertheless envision that there is a tradeoff between user-experience and the cost of the deployment. Another way to reduce the latency is to pick an optimal location to instantiate the middlebox. This is an additional optimization that can complement our architectural proposal, and is part of our future work.

6.2.2 Varying number of flows & function instances

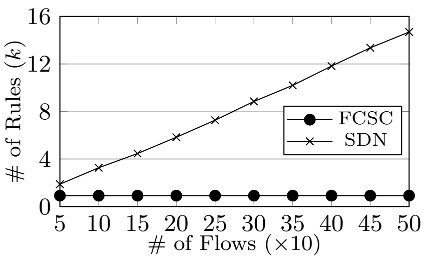
We now load the simulation with a varying number of flows (50, 100, 150, . . . , 500). Each flow has its own arrival time (within the first $5min$), a sending rate in the range



(a) Latency (and 95% CI) vs. # of flows



(b) % of packets lost vs. # of flows



(c) # of rules vs. # of flows

Figure 6: Results for varying-flow simulation.

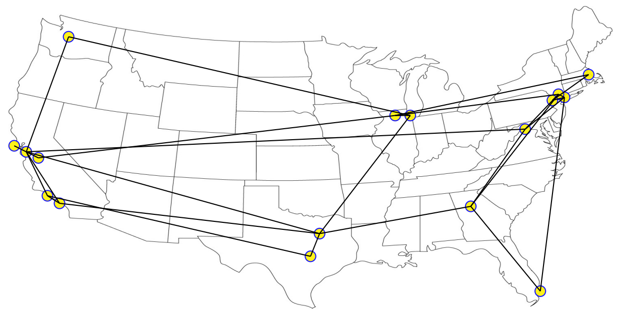


Figure 5: RocketFuel topology (Exodus, AS-3967).

($1.2Mbps$ - $11.9Mbps$) and duration ($0.05s$ to $91.24s$). We also randomly generated $1,151$ middlebox creation/failure events during the simulation period. We run the trace on both the FCSC and SDN solutions and compare the average latency for the flows, packet loss caused by middlebox failure, and the number of rules stored on the switches.

Fig. 6a shows the average latency along with 95% confidence interval (CI) for the different flows. FCSC provides lower average latency and less variability compared to the SDN solution, since the flows are able to take advantage of new instances that are closer.

The overall percent of packet lost in Fig. 6b shows that with the dynamic failure recovery, FCSC helps to deliver more packets to the destination. Lower loss rate usually means lower re-transmission rate and also lower overall network cost.

FCSC capable switches only maintain rules on a per available instance of a function, unlike the SDN solution that keeps rules for each flow type (defined by a n-tuple, potentially with wild-cards). Therefore, the number of rules do not change when we vary the number of flows (as shown in Fig. 6c). However the number of rules in SDN grows with the number of flows. We argue that our solution can be more scalable especially in a large network with millions of concurrent flows.

7. SUMMARY

Existing SDN-based network management solutions pre-translate the policy (what) into the forwarding rules at specific switches (where). Such a design choice limits the benefits that a truly software-based network could provide. By proposing FCSC, we explore the potential of using information-centric concepts within an SDN-based network management environment, especially focusing on service chaining. Using both synthetic and realistic topologies, we show that FCSC is able to provide policy makers simpler interfaces to control a flow (flexibility), is able to react to middlebox failures with fewer packet drops (reliability), is able to more quickly adapt to new instances of middlebox functionality (dynamicity), and requires less state to be maintained in the network (scalability). For our future work, we will explore better routing mechanisms that can fully exploit the benefits provided by FCSC.

8. ACKNOWLEDGMENTS

The research leading to these results has received funding from the EU-JAPAN initiative by the EC Seventh Framework Programme (FP7/2007-2013) Grant Agreement No. 608518 (GreenICN), NICT under Contract No. 167 and the Volkswagen Foundation Project “Simulation Science Center”. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the GreenICN project, the Simulation Science Center project, the European Commission, or NICT.

9. REFERENCES

[1] http://blog.streamingmedia.com/2010/10/how-dynamic-site-acceleration-works-what-akamai-and-cotendo-offer.html.

[2] http://blog.streamingmedia.com/2011/12/its-official-akamai-to-acquire-content-good-for-akamai-bad-for-customers.html.

[3] J. Abley and K. E. Lindqvist. Operation of anycast services. In *IETF, RFC*, number 4786, 2006.

[4] B. Ahlgren, M. D’Ambrosio, M. Marchisio, I. Marsh, C. Dannewitz, B. Ohlman, K. Pentikousis, O. Strandberg, R. Rembarz, and V. Vercellone. Design considerations for a network of information. In *Conext*, 2008.

[5] S. Arianfar, P. Nikander, and J. Ott. On Content-centric Router Design and Implications. In *Re-Arch*, 2010.

[6] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish. A layered naming architecture for the internet. In *SIGCOMM*, 2004.

[7] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax . In *IETF, RFC*, number 3986, 2005.

[8] R. Callon. Use of OSI IS-IS for Routing in TCP/IP and Dual Environments. In *IETF, RFC*, number 1195, 1990.

[9] B. E. Carpenter and S. Brim. Middleboxes: Taxonomy and Issues. In *IETF, RFC*, number 3234, 2002.

[10] I. Castineyra and M. Steenstrup. The Nimrod routing architecture. In *IETF, RFC*, 1992.

[11] J. Chen, M. Arumaithurai, X. Fu, and K. K. Ramakrishnan. G-COPSS: A Content Centric Communication Infrastructure for Gaming Applications. In *ICDCS*, 2012.

[12] J. Chen, M. Arumaithurai, L. Jiao, X. Fu, and K. K. Ramakrishnan. COPSS: An Efficient Content Oriented Publish/Subscribe System. In *ANCS*, 2011.

[13] CISCO. Policy-based routing. Technical report.

[14] C. Cui, H. Deng, D. Telekom, U. Michel, H. Damker, I. Guardini, E. Demaria, R. Minerva, and A. Manzalini. Network Functions Virtualisation. In *SDN and OpenFlow World Congress*, 2012.

[15] J. Erman, A. Gerber, K. K. Ramakrishnan, S. Sen, and O. Spatscheck. Over the Top Video: The Gorilla in Cellular Networks. In *IMC*, 2011.

[16] J. Erman and K. Ramakrishnan. Understanding the Super-sized Traffic of the Super Bowl. In *IMC*, 2013.

[17] B. Ford. Unmanaged Internet Protocol: taming the edge network management crisis. *arXiv preprint*, 2006.

[18] B. Heller, R. Sherwood, and N. McKeown. The Controller Placement Problem. *SIGCOMM CCR*, pages 473–478, 2012.

[19] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda. Is it still possible to extend TCP? In *IMC*, 2011.

[20] J. Hwang, K. Ramakrishnan, and T. Wood. NetVM: high performance and flexible networking using virtualization on commodity platforms. In *NSDI*, 2014.

[21] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking Named Content. In *CoNEXT*, 2009.

[22] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. B4: experience with a globally-deployed software defined wan. In *SIGCOMM*, 2013.

[23] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, et al. Onix: A Distributed Control Platform for Large-scale Production Networks. In *OSDI*, 2010.

[24] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A data-oriented (and beyond) network architecture. *SIGCOMM CCR*, pages 181–192, 2007.

[25] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann. Logically centralized?: state distribution trade-offs in software defined networks. In *HotSDN*, 2012.

[26] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. Inferring link weights using end-to-end measurements. In *IMW*, 2002.

[27] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM CCR*, pages 69–74, 2008.

[28] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *SIGCOMM CCR*, pages 69–74, 2008.

[29] N. B. Melazzi, A. Detti, G. Mazza, G. Morabito, S. Salsano, and L. Veltri. An openflow-based testbed for information centric networking. In *FutureNetw*, 2012.

[30] R. Moskowitz and P. Nikander. Host Identity Protocol (HIP) Architecture. In *IETF, RFC*, number 4423, 2006.

[31] A. Myles, D. B. Johnson, and C. Perkins. Mobile host protocol supporting route optimization and authentication. *JSAC*, pages 839–849, 1995.

[32] P. Pan, G. Swallow, A. Atlas, et al. Fast reroute extensions to RSVP-TE for LSP tunnels. In *IETF, RFC*, number 4090, 2005.

[33] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu. SIMPLE-fying middlebox policy enforcement using SDN. In *SIGCOMM*, 2013.

[34] R. Ravindran, X. Liu, A. Chakraborti, X. Zhang, and G. Wang. Towards software defined ICN based edge-cloud services. In *CloudNet*, 2013.

[35] 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 *SIGCOMM*, 2012.

[36] W. So, A. Narayanan, and D. Oran. Named data networking on a router: fast and dos-resistant forwarding with hash tables. In *ANCS*, 2013.

[37] Stoica, Adkins, Ratnasamy, Shenker, Surana, and Zhuang. Internet Indirection Infrastructure. In *SIGCOMM*, 2002.

[38] M. Vahlenkamp, F. Schneider, D. Kutscher, and J. Seedorf. Enabling Information Centric Networking in IP Networks Using SDN. In *SDN4FNS*, 2013.

[39] M. Walfish and H. Balakrishnan. Untangling the Web from DNS. In *NSDI*, 2004.

[40] M. Walfish, J. Stribling, M. N. Krohn, H. Balakrishnan, R. Morris, and S. Shenker. Middleboxes No Longer Considered Harmful. In *OSDI*, 2004.

[41] L. Wang, A. Hoque, C. Yi, A. Alyyan, and B. Zhang. Ospfn: An ospf based routing protocol for named data networking. *Tech. Rep*, 2012.

[42] Y. Wang, Y. Zu, T. Zhang, K. Peng, Q. Dong, B. Liu, W. Meng, H. Dai, X. Tian, Z. Xu, et al. Wire speed name lookup: A gpu-based approach. In *NSDI*, 2013.

[43] M. Yu, J. Rexford, M. J. Freedman, and J. Wang. Scalable Flow-based Networking with DIFANE. In *SIGCOMM*, 2010.

[44] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. D. Thornton, D. K. Smetters, B. Zhang, G. Tsudik, D. Massey, C. Papadopoulos, et al. Named data networking (ndn) project. *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC*, 2010.

[45] Y. Zhang, N. Beheshti, L. Beliveau, G. Lefebvre, R. Manghirmalani, R. Mishra, R. Patney, R. Subrahmaniam, M. Shirazipour, C. Truchan, and M. Tatipamula. StEERING: A Software-Defined Networking for Inline Service Chaining. In *ICNP*, 2013.