DRENCH: A Semi-Distributed Resource Management Framework for NFV based Service Function Chaining

Argyrios G. Tasiopoulos[‡*], Sameer G Kulkarni[§*], Mayutan Arumaithurai[§], Ioannis Psaras[‡],
K.K. Ramakrishnan[†], Xiaoming Fu[§], George Pavlou[‡]
[‡]University College London, [§]University of Göttingen, Germany, [†]University of California, Riverside.

***Abstract*—As networks grow in scale and complexity, the use of Network Function Virtualization (NFV) and the ability to dynamically instantiate network function instances (NFIs) allow us to scale out the network's capabilities in response to demand. At the same time, an increasing number of computing resources, deployed *closer* to users, as well as network equipment are now capable of performing general-purpose computation for NFV. However, NFV management in the presence of Service Function Chaining (SFC) for arbitrary topologies is a challenging task.**

In this work we argue for the necessity of an *algorithmic resource management framework* that captures the involved trade-offs of NFIs minimum workload, load balancing, and flow path stretch. We introduce DRENCH as a low complexity NFV and flow steering management framework. In DRENCH an NFV market is considered where a centralised SDN controller acts as market orchestrator of NFV nodes. Through competition, NFV nodes make flow steering and NFI instantiation/consolidation decisions. DRENCH design enables third party NFV nodes participation while it can coexist with other NFV management solutions. DRENCH orchestrator parameterisation strikes the right balance between path stretch and NFI load balancing, resulting in significantly lower Flow Completion Times, up to $10\times$ less, in some cases.

I. Introduction

Middleboxes (MBs) are becoming ubiquitous in today's networks. MBs have typically been constructed from purpose-built hardware, customized to perform specific tasks. Once setup, a network of MBs cannot alter its structure (*e.g.,* topology) or (service) functionality (*e.g.,* morph from one service to another). Network Function Virtualisation (NFV) [1] has been proposed to increase flexibility in the network, evolving MB architectures to virtual, or software-based services on top of commercial off-the-shelf (COTS) hardware. NFV promises to increase flexibility and achieve efficiency in using network resources, since both the structure and the (service) functionality of NFV nodes can be adjusted dynamically in response to service demand. Finally, Service Function Chaining (SFC), which determines the chain of NFV-based MB services a flow has to pass through, is gaining momentum as a necessary network process.

In order to exploit the full potential of virtualised Network Function Instances (NFIs), we argue that: *i)* NFIs have to be dynamically placed, replicated, instantiated and terminated (or consolidated), *ii)* new incoming flows have to be dynamically steered to the least-expensive NFI (in terms of current network and computation load), and *iii)* active, existing flows have to be redirected (if and when needed) to other instances of the same service in order to balance the load between the NFIs of this service. Following this line of argument, we introduce and make the case for *a resource management framework that dynamically handles NFIs and flow traffic*, in order to load-balance *i)* network load in links; and *ii)* computation load in NFV boxes. In designing such a resource management framework, we consider all potential options, that is, from centralized, software-based control plane approaches to decentralized, hardware-oriented data plane approaches. We find that a centralized controller can become the bottleneck when assigned with the task of making *real-time decisions* on service placement, instantiation, termination and flow steering/redirection. On the other hand, a purely distributed approach suffers from increased latency and overhead in order to exchange information between decision-making nodes, a process that also raises stability issues.

We argue for the need of a hybrid solution, where the *(logically) centralised SDN controller* performs lightweight tasks, related to the NFV environment coordination and flow setting-up of state; while the *distributed network of NFIs* makes more frequent decisions, that impact flow latency. Despite the multiple SDN-NFV architectures proposed in recent years (*e.g.,* E2 [2], Stratos [3], Slick [4], SDNFV [5]), the problem of resource allocation and management in such environments has not received as much attention.

In this paper, we propose a semi-DistRibutEd resource management framework for NFV based service function CHaining (**DRENCH**). *DRENCH incorporates a traffic load-balancing algorithm that utilises dynamic estimation of NFI loads with each NFV node independently directing flows to an appropriate least-loaded service instance; DRENCH utilises a real-time service instantiation capability and redirects existing flows as necessary; DRENCH is applicable for an NFI based Service Function Chaining environment by using a centralised SDN controller to disseminate information among the NFV nodes in the network.* DRENCH as a resource management framework can fit into most of the existing architectures, albeit with some modifications.

We realise DRENCH in the context of SDN-NFV architectures by defining an NFV-node market environment. In particular, an SDN controller acts as the market orchestrator/regulator, assigning *prices* to NFIs which are indicative of their workload. At the same time, NFV nodes target the increase of their 'income'. This means that NFV nodes aim to host popular NFIs that result in higher prices. In more

[*] Joint first authors

ISBN 978-3-901882-94-4

detail, when the demand for a service increases (declines) the price of the service NFIs rises (decreases) accordingly, which in turn may drive NFV nodes to instantiate (consolidate the) NFIs of the corresponding service. In addition to NFIs instantiation/consolidation, each NFV node is also responsible for taking flow steering and redirection decisions.

In DRENCH the market orchestrator is setting the control parameters of *i*) *minimum NFI price* and *ii*) *off path penalty factor*. The minimum NFI price defines a threshold for consolidating NFIs whose prices are below the minimum one. Since NFIs' prices are representative of their workload, the minimum NFI price indicates the threshold below which an NFI is considered being *under-utilised*, thereby controlling the number of active NFIs. On the other hand, the off path penalty factor controls the path-stretch of a flow in the context of SFC, thereby penalising the choice of NFIs that force the flow to deviate from its shortest path towards the destination. Considering Flow Completion Time (FCT) as an index of flow performance, DRENCH minimum NFI price (off path penalty factor) defines the tradeoff between under-utilised instances (flow path stretch) and FCT.

The main technical contributions of this paper involve:

- *A feasible NFV management approach:* in DRENCH resource management decisions are taken locally by NFV nodes while the market orchestrator solves lightweight problems, addressing a complex problem in a computationally feasible way with respect to *i) path-stretch, ii) number of active NFIs per service, iii) load on each NFI and iv) flow completion time*.
- *A decoupled NFV resource management framework:* in DRENCH, NFV nodes do not have to be owned by the same entity, which thereby contributes to the incremental adaptation of NFV in arbitrary network topologies.
- *Implementation and large scale evaluations:* We prototyped DRENCH in Cloudlab testbed and Mininet to demonstrate that DRENCH is immediately deployable in an SDN environment. With Mininet, we compare DRENCH to a centralized approach Slick [4] on a 4K-Fat tree topology. Additionally, we compared DRENCH in a simulation environment consisting of a Rocketfuel topology (87 switches) to a custom centralized approach: SIMPLE [6] on top of a E2 SDN framework [2]. Our results show that DRENCH is robust to: *i)* asymmetries caused by dynamics of arrival/departure of *elastic* flows with different service needs, and, *ii)* the instantiation/removal/failure of service instances (see Section VI).

II. Related Work

In the case of purely centralized solutions, the complexity involved in the decision making for service instantiation and flow redirection (NP-hard in many cases) results in a much coarser granularity of decision making. Solutions can only be based on heuristic-based approaches [7], or act on a per-flow basis [8]. SIMPLE [6] relies on an offline Integer Linear Programming (ILP) solver to optimise the number of flow rules on the switches and an online Linear Programming (LP) solver for load balancing. However, SIMPLE [6] assumes that the NFIs and middleboxes are statically placed and any dynamic instantiation of NFIs requires the re-run of the expensive offline ILP solver. Slick [4] provides a programming model abstraction for service chaining that supports heuristic based function placement and flow steering schemes. Slick [4] also supports dynamic scaling of NFIs. However, it does not redirect the existing flows from the overloaded instances, but only steers the new flows to the scaled-out ones.

By and large, related works in this area have targeted *only newly-arriving flows*, while works such as Split/Merge [9], that address flow-redirection, need to pause the existing flows for transferring NFIs and forwarding states. On the other hand, purely distributed approaches for load balancing (*e.g.,* TeXCP [10], CONGA [11]) face a different set of issues, such as: *i) high overhead and latency:* a large amount of information has to be exchanged among decision making nodes to synchronise their view of network state, with the consequent latency as the network scales; *ii) complexity:* achieving a stable synchronised view at every decision making node is complex; and *iii) inefficiency and stability:* since each node makes decisions based on its local view of the network state, packet loss, frequent rerouting, and transient loops are likely. In [12], authors employ the concept of shadow-prices to trade-off performance, QoS, and complexity, but they limit their scope only to the flow steering problem in SFC.

OpenNF [13] presents a control plane for managing both network forwarding state and internal NFI state for migrating flows, where migration events are generated by NFIs and buffered at the controller for the interval of state migration. Stratos [3] proposes an orchestration framework that employs a rack-aware NFIs placement strategy with horizontal scaling and migration of NFIs. The load/flow distributions are computed by an ILP formulation where only new flows are steered to the new instances while the existing ones continue to get processed at the same NFI, without alleviating the load on the already congested instances. Lastly, E2 [2] is a NFV scheduling framework that supports affinity based placement and dynamic scaling of NFIs. The framework tries to minimise the traffic across switches while balancing the traffic across NFIs, but it avoids flow redirections across hardware switches.

To the best of our knowledge, *DRENCH is the first work that tackles both NFI placement and flow steering problems in arbitrary topologies*. DRENCH provides a computationally feasible algorithmic resource management framework inspired by the principles of market competition.

III. Design Overview

A. Desired Properties

DRENCH is an in-network, congestion-aware, load balancing algorithmic framework that handles SFCs and dynamic NFIs in arbitrary network topologies. In designing DRENCH, we focus on providing the following key properties:

P1 **Efficiency:** As an NFI placement mechanism, DRENCH should neither under-utilise nor over-utilise the resources available in NFV nodes.

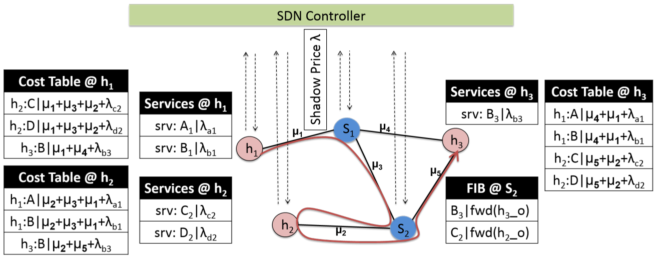


Fig. 1: DRENCH High-Level Operation

P2 **Cost awareness:** DRENCH should instantiate the minimum NFIs to meet the requirements of the SFCs for the flows through the network at any point in time, and balance the utilisation across the active NFIs.

P3 **Fine-grained flow handling:** DRENCH must meet each flows' SFC functional requirements, in terms of end-to-end latency and minimum throughput.

P4 **Responsiveness:** DRENCH should react to SFC traffic demand fluctuations, especially when traffic is volatile and bursty [14], [15], for arbitrary network topologies.

P5 **Incremental deployability:** DRENCH should require the minimum possible modifications in terms of protocols and network infrastructure. It should also be applicable to any of the existing SDN architectures [2], [13], [5] with minimal changes. Furthermore, it should be possible to directly apply DRENCH to a subset of available switches and of incoming traffic when necessary.

B. DRENCH Solution Overview

Our framework is designed to leverage the benefits of the centralised as well as the distributed networking paradigms. We use the **centralized approach**, *i.e.,* an SDN controller, to perform tasks with less computation load, but those that need to be carried out in a coordinated fashion across multiple nodes. These tasks include: *i)* gather, compute and disseminate NFI load information periodically to all the decision making entities; and *ii)* set up paths towards instances and egress nodes in case they do not already exist. Additionally, the SDN controller is used to decide which services are applicable to a flow (based on policies and/or flow characteristics). This design choice reduces the complexity of the controller and also overcomes the issues faced by a purely distributed approach, where the decision making entities might not have up to date information, thereby impacting performance. On the other hand, a **distributed approach** is used for decision making at individual NFV nodes. Based on the information provided by the controller, each node independently decides to *i) steer flows* towards the next required service; *ii) redirect flows* to the least loaded instance; and *iii) instantiate/terminate* NFIs in order to adapt to demand. The high-level operation of the proposed mechanism is shown in Fig. 1.

IV. DRENCH Components

DRENCH consists of the following components:

$\mathcal{G}$	Network topology
$\mathcal{V}$	Set of switches
$\mathcal{E}$	Set of links
$\mathcal{H}$	Set of NFV Nodes
$\mathcal{S}$	Set of services
$\mathcal{H}_s$	Set of NFV nodes executing service s
$\mathcal{F}$	Set of flows
x_f, x_f^*	Rate and optimal rate of flow f
$U_f(x_f)$	Utility function of flow f
b_e	Bandwidth capacity of link e
$a_{e,f}$	Coefficient = 1, if flow f traverses link e
b_s^h	Computational resources of NFI s at h
d_s	Computational power Required by a NFI of s for processing a single bit of traffic
$d_{s,f}^h$	Coefficient = d_s if f is processed by NFI s at h
w_f	Weight of flow f
λ_s^h	Service cost of NFI s at NFV node h
$\underline{\lambda}, \bar{\lambda}$	Minimum and Maximum shadow prices, defining the efficiency of an instance
p_{v_i,v_j}	Shortest path from switch v_i to switch v_j
μ_{v_i,v_j}	Communication cost from switch v_i to switch v_j
$\mathcal{C}_{v_i,h}(s)$	Communication and service cost from switch v_i to a NFI executing service s at NFV node h
$\|p_{v_i,v_j}\|$	Number of Hops from switch v_i to switch v_j
$\Delta p_{v_i,h}^f$	Shortest path deviation overhead
ρ	Off Path penalty factor
$\mathcal{C}_{v_i,h}^f(s)$	Estimated $\mathcal{C}_{v_i,h}(s)$ cost of flow f including ρ
θ_{rid}	Redirection threshold
P_h	Profit of NFV h in terms of shadow prices
$\widetilde{\lambda}_{on}, \widetilde{\lambda}_{off}, \widetilde{\lambda}$	On-/Off- path and expected competitive price

TABLE I: DRENCH Notation Description

- **Market Orchestrator:** It associates every NFI and link resource to a *shadow price* (*i.e.,* cost) produced by utilising global information available at the controller. The orchestrator regulates the market by allowing the existence of instances above a certain minimum price.
- **Flow Steering and Redirection:** This component steers each flow through a valid sequence of NFIs (according to its SFC) determined by the SDN controller. Steering and redirection takes into account latency, NFI and link costs.
- **NFI Instantiation/Consolidation:** This component instantiates and consolidates NFIs in a distributed way through the market competition between NFV nodes.

Below, we describe each of these components in detail.

A. Market Orchestrator

DRENCH, as any market-based approach, requires the association of each network resource (commodity), in terms of NFIs and link bandwidth, to an offered price, which is imposed on a given set of incoming flows (demand) that utilise this resource. In particular, when the quantity of demanded resources equals the quantity supplied for a set of prices, we refer to them as *market-clearing prices*. DRENCH market-clearing prices should *A1*) be representative of each NFI's workload, *A2*) be derived in the minimum possible time, *A3*) not require additional in-network signalling given the existence of an SDN controller [16], [11]. Every price derivation violating requirement (*A2*) and (*A3*) would be in stark contrast with DRENCH desired properties *wrt* responsiveness (***P4***) and incremental deployability (***P5***), respectively.

DRENCH deploys a *Market Orchestrator/Regulator* component, which by simply exploiting flow path information, already available at the SDN controller, efficiently derives the market-clearing prices; complying to requirements (*A1*)-(*A3*). Inspired by [17], where the authors formulate a Network Utility Maximisation problem (NUM) based on market principles to allocate bandwidth resources to a set of flows, we extend their model to include NFI computational resources. We achieve this by solving the Extended Network Utility Maximisation problem (ENUM) at the Market Orchestrator as we describe next.

We denote the network topology by $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, of $\mathcal{V}$ switches and $\mathcal{E}$ links, where a set of NFV nodes, $\mathcal{H}$, is placed at a subset of switches $\mathcal{H} \subseteq \mathcal{V}$. Then, given a set of NFIs executing a set of $\mathcal{S}$ services and a set of $\mathcal{F}$ flows, we associate each link, $\forall e \in \mathcal{E}$, with a bandwidth capacity, b_e, and each NFI s at NFV node h with b_s^h computational resources, in order to form the ENUM problem that maximises the total utility of the system. Similar to NUM, we associate each flow rate, $x_f \geq 0$, with a utility that is a weighted logarithmic function, $U_f(x_f) = w_f \log(x_f)$, of weight w_f, capturing a decreasing marginal gain as the flow rate increases (*i.e.,* rate changes at low rate flows have a greater impact on their utility). In turn, we maximise the total system utility, $\sum_{f \in \mathcal{F}} U_f(x_f)$, subject to link capacity constraints, $\sum_{f \in \mathcal{F}} a_{e,f} x_f \leq b_e$, $\forall e \in \mathcal{E}$, and computational resource constraints, $\sum_{f \in \mathcal{F}} d_{s,f}^h x_f \leq b_s^h$, $\forall s \in \mathcal{S}, \forall h \in \mathcal{H}$; where $a_{e,f}$ is a coefficient equal to 1 if flow f traverses link e and 0 otherwise, while $d_{s,f}^h$ equals the computational power required by service s for processing a single bit of traffic, d_s, if f is executed at NFI s of NFV node h, and 0 otherwise. Parameters $a_{e,f}$ and $d_{s,f}^h$ describe the path of each flow and therefore they are known to the SDN controller which provides them to the Market Orchestrator.

Since the objective function is differentiable and strictly concave, while the feasible region of the constraints is compact, the optimal rates $x_f^* \; \forall f \in \mathcal{F}$ exist, are unique, and can be found efficiently by Lagrangian methods. Based on [17], it can be shown that the dual problem of the ENUM is:

$$\begin{aligned} \text{maximise} \sum_{f \in \mathcal{F}} w_f \log\big(\sum_{e \in \mathcal{E}} \mu_e a_{e,f} + \sum_{h \in \mathcal{H}} \sum_{s \in \mathcal{S}} \lambda_s^h d_{s,f}^h\big) \\ - \sum_{e \in \mathcal{E}} \mu_e b_e - \sum_{h \in \mathcal{H}} \sum_{s \in \mathcal{S}} \lambda_s^h b_s^h \end{aligned} \tag{1}$$

subject to

$$\mu_e \geq 0, \; \forall e \in \mathcal{E},$$
$$\lambda_s^h \geq 0, \; \forall s \in \mathcal{S}, \forall h \in \mathcal{H},$$

where μ_e and λ_s^h are the Lagrange multipliers of link e and service instance s at NFV node h respectively. The Lagrange multipliers are also known as *shadow prices*, due to their association to the optimal rates of each flow:

$$x_f^* = \frac{w_f}{\sum\limits_{e \in \mathcal{E}} \mu_e a_{e,f} + \sum\limits_{h \in \mathcal{H}} \sum\limits_{s \in \mathcal{S}} \lambda_s^h d_{s,f}^h} \tag{2}$$

where weight w_f is perceived as the budget that flow f is willing to pay for its rate, while the denominator is the cost imposed to the flow in order to use the resources along its path. In that sense, *each Lagrange multiplier can be considered as the price of a particular resource*, leading us to the following definition about communication and service cost.

Definition 1: The *communication cost* between two switches, $v_i, v_j \in \mathcal{V}$, is the sum of on-path link shadow prices $\mu_{v_i,v_j} = \sum_{e \in p_{v_i,v_j}} \mu_e$, where p_{v_i,v_j} is the shortest path between switches v_i and v_j; while the *service cost* of an instance s at NFV node h is the shadow price λ_s^h.

Note that the service and communication costs are kept in the forwarding tables of the NFIs, *i.e.,* the decision making nodes, and are updated periodically by the SDN controller after being estimated by the Market Orchestrator (see Fig. 1).

Shadow prices are indicative of the workload at a particular resource, complying with (*A1*). In fact, from (2), we derive that the value of a shadow price, λ, defines the maximum possible rate that flows using that resource can achieve, w_f/λ.[1] Based on the maximum achievable flow rate we can define the efficiency of a NFI as a range of shadow prices.

Definition 2: The Market Orchestrator determines the load of a NFI by a shadow price range $[\underline{\lambda}, \bar{\lambda}]$, where if a service cost, λ_s^h, is less/more than $\underline{\lambda}/\bar{\lambda}$ the NFI is considered under-/over-utilised, respectively.

Given the shadow price range $[\underline{\lambda}, \bar{\lambda}]$ the Market Orchestrator tries to maintain the minimum required number of instances per service type (***P2***) by: *i)* terminating instances that are underutilised and *ii)* allowing for more instances for services whose existing instances are over-utilised (see Section IV-C).

B. Flow Steering and Redirection

1) Flow Steering: Given a placement of NFIs and their respective shadow-prices, as determined by the Market Orchestrator, DRENCH's flow steering component is responsible for steering each new incoming flow towards the chain of required services. The flow steering component tries to route the flow through the chain that imposes the lowest possible cost to the flow. Determining the optimal end-to-end path of a flow through the SFC is a NP-complete problem [18]. DRENCH works on a hop-by-hop heuristic basis, picking each time the best next-hop NFI choice, in an effort to achieve instantaneous and adaptive steering decisions (***P4***).

We illustrate DRENCH's flow steering component through the following example. Assume that flow f arrives at the network requiring the execution of service s (or service chain $s_1/s_2/.../s_m$) before being delivered to destination v_f. Let v_i be the switch that has to make a steering decision about f and $\mathcal{H}_s \subseteq \mathcal{H}$ be the set of NFIs of service s. Then the combined communication and service s execution cost at $h \in \mathcal{H}_s$ is $\mathcal{C}_{v_i,h}(s) = \mu_{v_i,h} + \lambda_s^h$. The flow steering component initially estimates the shortest path deviation

[1] It follows that the shadow prices are positive when a resource is totally utilised and 0 otherwise. To introduce a minimum workload to the resources that are not saturated, we add a set of *dummy flows* into $\mathcal{F}$ when solving (1).

overhead applied by steering flow f to instance h in terms of hops, *i.e.,* $\Delta p^f_{v_i,h} = |p_{v_i,h}| + |p_{h,v_f}| - |p_{v_i,v_f}|$, weighted by an *off-path penalty factor* ρ. Therefore, the estimated cost applied to the flow for executing service s at NFI of h is $\mathcal{C}^f_{v_i,h}(s) = \mathcal{C}_{v_i,h}(s) + \rho\Delta p^f_{v_i,h}$. Then, v_i selects the next service instance s of f that minimises $\mathcal{C}^f_{v_i,h}(s)$:

$$h^* = \underset{h\in\mathcal{H}_s}{\operatorname{argmin}}\, \mathcal{C}^f_{v_i,h}(s) \qquad (3)$$

In Eq. (3) the off-path penalty factor, ρ, dis-incentivises node v_i from sending flow f away from its shortest path towards v_f. Eq. (3) applies on a hop-by-hop basis, that is, it is calculated at each NFV node responsible for forwarding flow f towards the next instance in its chain.

Lastly, upon making a steering decision, switch v_i informs the SDN controller that flow f is forwarded towards h^* to execute service s. At the same time, the SDN controller is setting up paths towards NFIs and/or egress nodes as necessary.

2) Flow Redirection of Stateless and Stateful flows: The cost of a service instance might change dramatically throughout the duration of a flow, rendering previous flow steering decisions outdated. Therefore, *redirection of existing flows* is necessary in order to keep the expenditure of existing flows at low levels (***P3-P4***) and avoid routing through overutilised instances (***P1***). We realise flow redirection as follows: if the cost difference between two instances of s at h and h', as seen by switch v_i, is bigger than a redirection threshold, θ_{rid}, $\mathcal{C}_{v_i,h}(s) - \mathcal{C}_{v_i,h'}(s) > \theta_{rid}$, switch v_i repeats the flow steering process for a portion of flows that v_i currently forwards to h. The redirection threshold is set to $\theta_{rid} = \bar{\lambda} - \underline{\lambda}$.

Rerouting of stateful flows to dynamically instantiated services for improving load balancing is usually complex and costly. For instance, solutions such as Split/Merge [9], pause ongoing flows in order to transfer internal NF and forwarding states. In DRENCH, we leverage the approach in Split/Merge [9], to pause ongoing flows and transfer the internal state of the involved network functions. To identify service instances, we make use of the Information Centric Networking (ICN) construct which is proven to be beneficial in terms of providing flexible routing, and reducing the routing states at the switches [19].

C. Instantiation

In DRENCH, NFV nodes autonomously provide services in an effort to maximise their profit, in terms of *shadow prices* (*i.e.,* the cost to execute some service). In particular, let S_h be the set of NFIs at some NFV node h, then the profit of h in terms of *shadow prices*, λ^h_s, can be estimated, as:

$$P_h = \sum_{s\in\mathcal{S}_h} \lambda^h_s \qquad (4)$$

DRENCH NFI instantiation/consolidation scheme defines how service demand and NFI *shadow prices* affect the individual NFV node decisions to manage the number of service instances. Through competitiveness, NFV nodes achieve responsiveness to NF demand changes, while the market orchestrator ensures market efficiency, as we explain next.

1) NFV Node Competitiveness: Let the *shadow price* of an NFI s' at NFV node h' be λ'. We are interested in estimating the competitive price of a potential NFI s at an NFV node h, $h \neq h'$, with respect to λ'.

Definition 3: The *shadow price* of NFI s is *competitive* in the price of NFI s' when the flow steering component has a preference, or is indifferent, of steering new flows at s.

Then, let μ be the communication cost, between NFV node h' and h that are Δp hops away, and f be a new flow that is about to get steered at NFI s'. Then according to DRENCH's flow steering component, the minimum competitive price at s for flow f, would be equal to $\widetilde{\lambda}_{off} = [\lambda' - \mu - \rho\Delta p]^+$, where ρ is the off-path penalty factor.[2] The off-path penalty factor is taken into account as in the worst case that flow f will have to traverse Δp additional hops to reach s from s'. This acts as a disincentive for a node to forward traffic to nodes that are far off from the flow's shortest path. On the other hand, in the best case, that flow f is forwarded to NFV node h' by NFV node h, meaning that s is already *on the path* of flow f and additional hops are not required;[3] the minimum competitive price at s for flow f is $\widetilde{\lambda}_{on} = \lambda' + \mu$.

The expected competitive price of NFI s with respect to NFI s' price λ' will be a value between $(\widetilde{\lambda}_{off}, \widetilde{\lambda}_{on})$. Let y be the total amount of traffic with competitive price $\widetilde{\lambda}_{on}$. Then y can be considered as the local information of NFI s' demand at NFV h that accounts for the utilization percentage $d_{s'}y/b^{h'}_{s'}$. Here, $d_{s'}$ is the computational power required by the service of NFI s' for processing a single bit of traffic and $b^{h'}_{s'}$ is the fixed computational resources that are allocated to NFI s'. Then the expected competitive price is estimated as:

$$\widetilde{\lambda} = (d_{s'}y/b^{h'}_{s'})\widetilde{\lambda}_{on} + (1 - d_{s'}y/b^{h'}_{s'})\widetilde{\lambda}_{off} \qquad (5)$$

2) NFI Instantiation: As long as a NFI shadow price, λ, executing a service at NFV node h, is lower than the maximum target price, $\lambda < \bar{\lambda}$, this service is not considered over-utilised and an instantiation of an additional NFI of the same service at h is prohibited (see also *Definition 2*). On the other hand, if $\lambda > \bar{\lambda}$, the Market Orchestrator limits the number of NFI that can be created by competing the NFI with service cost λ to $\lfloor \lambda/\bar{\lambda} \rfloor$. Therefore, given the set of allowed services for instantiation at each NFV node h, h estimates the expected competitive prices of every NFI. Then moving from the highest to the lowest competitive price, the NFV node instantiates the service associated with the price $\widetilde{\lambda}$ as long as *i*) it is expected that the instance will not be under-utilised, $\widetilde{\lambda} > \underline{\lambda}$, and *ii*) the Market Orchestrator maximum number of instances allows it, respecting properties (***P1***), (***P2***), and (***P4***).

3) NFI Consolidation: If the price of an instance is below the minimum target shadow price, $\underline{\lambda}$, the NFV node consolidates this instance ***(P2)***. When there is a service availability requirement, the market orchestrator can hinder the consolidation of the last instance of that service.

[2] $[\cdot]^+$ denotes the projection onto nonnegative orthant.

[3] In practice, it is not the NFV node that is aware of the forwarded traffic but the switch that the NFV node is attached to.

V. IMPLEMENTATION

DRENCH's prototype involves the implementation of an SDN controller/orchestrator, Open vSwitches, and a set of custom NFIs. The overall implementation, including the controller extensions for supporting NFIs, is $\sim$ *1800 LOC*.

A. Control Plane: DRENCH Controller

We extended Pox[4] to serve as DRENCH's controller, where policy specifications are provided as input.

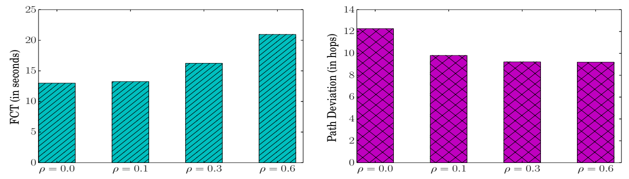
Flow Classifier and Policy Enforcement: DRENCH controller performs fine-grained flow classification, based on the standard IP 5-tuple. As a flow is mapped into a policy specification, the Controller applies the sequence of the involved service functions. We dedicate the combination of *IP-ToS* field and *L4: destination port* to represent the service chain ID and the sequence of functions in the service chain respectively.

Flow Steering: In order to be more flexible, and readily deployable with NFIs, we rely on the switch-based service chain/network function ID mapping that can be supported by openflow switches, without the need for any modifications to the NFI. The controller sets the path from the ingress/NFV node to the next NFV node/destination in the chain. For finer granularity at each ingress or intermediate NFV node, the flow rules are installed based on the match obtained from the flow-classification - this readily enables the capability to match and correlate the packets that enter/exit the NFIs, even when NFIs modify the packet headers, while the rest of the intermediate switches rules are installed for forwarding the traffic via "tunnels" to the next NFV node or egress switch. For tunnelling, we enhance the named instance source routing scheme [19], wherein the tag comprising of the VNFs service type (*SVC-ID*), the switch (*SW-ID*), and the pinned compute core (*CoreID*) uniquely represent the NFI. It is defined as: $NFI\text{-}ID = CoreID|SW\text{-}ID|SVC\text{-}ID$.

This enables the paths to be installed proactively on all the intermediate switches as soon as an NFI is discovered in the network. Our proposal of ID-based tunnelling can be realised by using either the Multi-Protocol Label Switching (MPLS) or VLAN tags, underlay (unused IP/TCP header fields like DS/option fields) or with Network Service Header (NSH) [20]. The choice of Layer-2.5 (MPLS) or Layer-2 (VLAN) tag makes it convenient to define the match in Openflow switches, *i.e.,* by being agnostic to L3/L4 fields allow any TCP/UDP flow to be matched with a same rule. This helps to significantly reduce the number of switch rules. As most MBs readily support VLAN as opposed to MPLS [21], we make use of Layer-2 tags to realise the NFI-based tunnels.

B. Data Plane: Openflow Switches and Network Functions

One of the key challenges we faced in the implementation of DRENCH prototype was the unavailability of a MB/NFV-capable switch in the Mininet environment. Our solution was to realise NFIs as hosts connected to the switches. However, this resulted in additional challenges since OpenFlow is designed with a Southbound API control channel between the controller and the network switches, but not the hosts. Therefore, any information (*e.g.,* estimated cost of other instances/shadow-prices) that had to be exchanged between the controller and the NFV-hosts had to be performed either via the switch (*e.g.,* we used LLDP packets to make the controller aware of the presence of NFV-hosts) or via a separate channel. In a real-world deployment, we believe that this would not be the case since NFV nodes will have a Southbound-API based channel from the controller through an in-built switch. Once the communication channels were established, the controller was able to both obtain and disseminate `cost` related information periodically. Moreover, to demonstrate that the NFI capability can be realised on Openflow switches, we implemented on each host a host-local Open vSwitch and controller. The host interface is setup as a port of the local vSwitch. This way, we leveraged the local Open vSwitch and Pox controller to implement the VNF specific functionality, *i.e.,* to monitor and disseminate NFI-specific load information and communicate it to the global controller.



(a) Flow Completion (b) Path Deviation

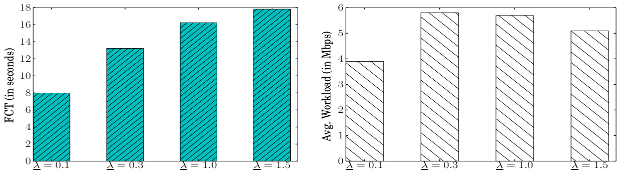
Fig. 2: Off-path penalty (x-axis)

VI. EVALUATION

Our goal is three-fold: *i)* study the effect of different DRENCH parameters, the resulting trade-offs, and the impact on performance in order to fine-tune DRENCH; *ii)* highlight the benefits of DRENCH on a controlled topology (we perform these evaluations on a CloudLab[5] test-bed); and *iii)* compare the performance of DRENCH with other approaches in large scale scenarios (both data-center and Rocketfuel topologies). We make use of the DRENCH prototype on a data-center topology in a Mininet Cluster[6] to study the benefits of DRENCH in terms of Flow Completion Time (FCT), delay, number of active NFIs, NFI utilisation, and the impact of redirection. We compare DRENCH against a centralised approach (*Slick* [4]) and *DRENCH without redirection (DwoR).*

Moreover, we build a custom-based simulator to study the benefits of DRENCH in terms of path deviation, average throughput and FCT in comparison to a custom centralised approach, *i.e., SIMPLE* [6] *on top of E2 SDN framework* [2]. Unlike similar works that focus only on the latency requirements of service chains, we also emphasize on FCT - arguably the most important metric for the user [22].

[4] https://openflow.stanford.edu/display/ONL/POX+Wiki



(a) Flow Completion (b) Work Load (Throughput)

Fig. 3: Shadow Price threshold (x-axis)

A. *DRENCH Parameter design and study of tradeoffs*

We implemented DRENCH on a python-based discrete event simulator using *SimPy*[7], in order to be able to flexibly fine-tune DRENCH's parameters and to perform large scale evaluation. For these experiments, we perform simulations on a Rocketfuel topology with 27 hosts, that send/receive flows, and 57 nodes that are capable of hosting NFIs. Unless stated otherwise, the off-path penalty, ρ, is 0.3 while we consider SFCs of length equal to 2. We then apply the results of the simulation study to setup DRENCH's prototype.

1) Off-path penalty, ρ: Traditionally, flows follow the shortest path towards their destination, deviating only for traffic engineering reasons and/or in response to link/node failures (*e.g.,* fast-reroute in MPLS). When it comes to service chaining, flows deviate from their shortest path in order to be served by NFIs. In DRENCH the off-path penalty factor, ρ, controls the tradeoff between the shortest path deviation and FCT by *trading* path deviation overhead for less congested NFIs, as Fig. 2 indicates. In more detail, the FCT increases function to off-path penalty, ρ, since flows prefer to get served by a more congested NFI (*e.g.,* with a higher service cost) than deviating from their shortest path, Fig. 2a. Hence, for our experimental purposes we choose a value of ρ in the range of 0.3-0.5. The exact setting of the ρ factor is up to the network operator. During low-demand periods (*e.g.,* during nighttime), where links are generally less utilised, operators might choose a lower value to improve FCT (*i.e.,* given low link utilisation, extra path deviation should not cause problems). On the other hand, in high demand periods, path deviation should be kept to lower levels, even if this increases the individual flows' FCT, in order to avoid extensive path stretch.

2) Shadow price Threshold: Fig. 3 shows the results of varying the minimum shadow-price threshold, $\underline{\lambda}$, at which new instances are spawned. Fig. 3a shows that FCT increases function to threshold values. A low $\underline{\lambda}$ of 0.1 results in lower FCT compared to values in the range of 0.3 to 1.5, but at the same time leads to a poor average utilisation of NFIs, *i.e.,* 4 Mbps as seen in Fig. 3b. In particular, utilisation is considered as the amount of traffic (in terms of throughput - see Fig. 3b) that an NFI serves on average. The exact setting of $\underline{\lambda}$ is up to the network operator. If the demand is low, then more instances could be allowed to reduce the average FCTs. In contrast, during high demand periods, the operator might have to compromise on individual flow FCT, in order to make full utilisation of the existing NFIs and eventually serve more flows overall.

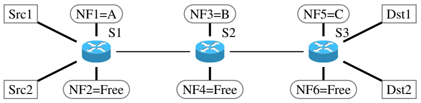
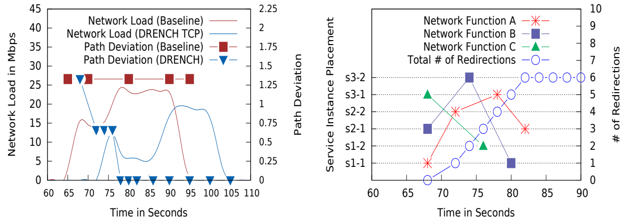


Fig. 4: Simple Topology with initial placement of NFIs.



(a) Network Utilization (b) NFI Relocation

Fig. 5: TCP flow with service chain of *C—B—A*

TABLE II: Average Bitrate and Delay

	DRENCH	Baseline
Avg Bitrate (Mbps)	4.033	3.814
Std. Dev of Pkt Delay (ms)	129.151	143.216

B. *Testbed: Simple controlled experiments*

We perform controlled experiments on a small topology, as shown in Fig. 4, in our CloudLab setup to illustrate the benefits of DRENCH components. Consider flows *Src1-Dst1*, requiring the service chain of $C/B/A$, which also comprises the ideal placement.

We study the performance and behavior of the system in the worst case scenario, where the services are initially placed in the reverse order as depicted in Fig. 4.[8] To prevent an increase in the flow's path length (by going back and forth in this topology), it is desirable to relocate the NFIs (from $A-B-C$ to $C-B-A$ in nodes S1-S2-S3, respectively) to minimize path stretch and FCT.

In Fig. 5a, we observe that, initially, due to the service instances being located in the wrong order (*i.e.,* $A-B-C$, instead of $C-B-A$) the flow suffers higher path-stretch, resulting in additional delay and higher network load. Later, as the switches gradually adapt towards the ideal placement, path stretch declines and network throughput increases (see Table II) compared to a non-reactive (Baseline) approach.

When all the instances individually seek to be competitive and maximise their utilisation, the switches with NFV capacity near the ingress switches end up hosting most of the services. From Fig. 5b, we see that DRENCH converges to this ideal case by instantiating and placing the services in the chain along the path towards the destination.

C. *Large scale Evaluation: Data-Center Topology*

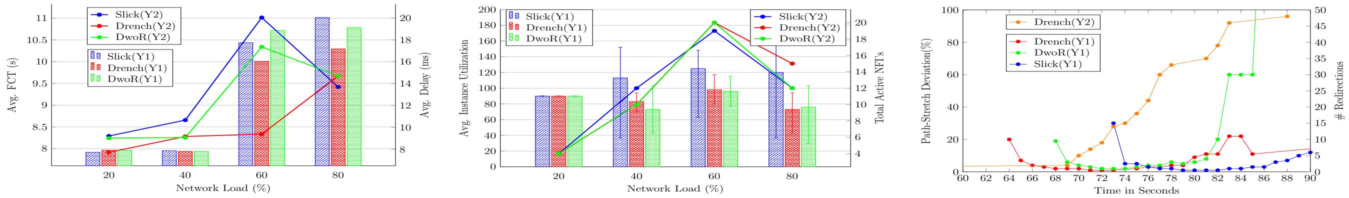
We setup a Mininet Cluster on Cloudlab to study the performance of DRENCH in a large network topology.

[5] https://cloudlab.us

[6] http://mininet.org

[7] http://simpy.readthedocs.io/en/latest/

[8] We set the NF capacity and the off-path penalty factors to just exceed the threshold required in order to allow for service instantiation.



(a) Overall Avg. FCT and Delays (b) Avg. # of NFIs and an NFI's Utilization (c) Impact of redirection

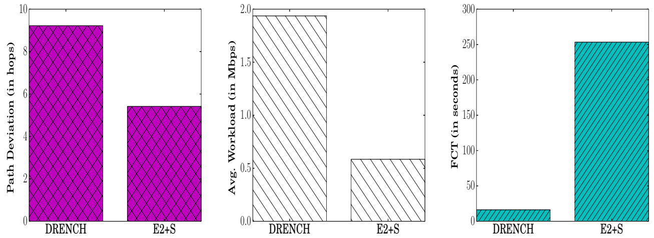
Fig. 6: Study on a Data-Center Topology (Y1: Left Y axis, Y2: Right Y axis)

Topology: We use a 4K Fat-Tree topology to evaluate DRENCH in a data-center environment and compare it to Slick (used as an example of a centralized approach.) We consider that only aggregation-layer switches in the fat-tree topology have NFV capability and dedicate 2 cores per aggregate switch for instantiating the NFIs.

Workload: We model the traffic based on the available data center workload characteristics similar to the ones used in [11], [14]. The workload constitutes a mix of elephant flows (20% with flow size $\geqslant$ 10MB) and mice flows (80% with flow size $<$ 2MB). Thus, elephant flows account for more than 80% of the traffic bytes. We use iperf[9] and D-ITG [23] to generate traffic with varying network loads. Flows originate from one of the servers connected to a leaf switch and terminate at another server connected to another leaf switch (in either the same or different pod). We use TCP flows in a client/server model with random flow arrival times. Based on the information on the service chaining policies in [24] and the details presented in earlier work (*e.g.,* [6]), we setup a service function chain of 3 distinct Network Functions (NFs). We assume that in total there are 6 NFs available in the network and that each flow requires exactly three of them. The service functions are chosen based on a zipf-distribution with exponent set to 0.3.

Parameters: Based on our findings from Section VI-A2, we set the `off-path penalty factor` eqaul to 0.5, and the instance `shadow-price` that influences service consolidation and service instantiation decisions as follows. An instance is consolidated when the `shadow-price` reflects a NFI utilisation of $<$ 30% and the flows served by that instance are less than 5. This is done in-order to mitigate the number of flow re-directions and the packet re-ordering impact of flow re-directions. A new NFI is instantiated when the `shadow-price` reflects NFI utilisation of $>$ 85%.

Comparison: In order to perform a fair comparison, we implemented a greedy heuristic-based flow steering approach as in Slick [4] - as a representative of a fully centralised state-of-the-art load balancing scheme.[10] We also compare against DwoR to study the effects of redirections and its impact on DRENCH. Our main goal is to evaluate the behaviour and performance of DRENCH *wrt* NFI placement, flow steering, and load balancing in terms of its efficiency in NF resource utilisation and FCT.



(a) Path Deviation (b) NFI Work Load (c) FCT

Fig. 7: Comparison of Drench vs. E2+SIMPLE

Figure 6a shows the average FCT and packet delays for different schemes. We observe that at lower loads (20-40%) all the schemes have similar FCT, but DRENCH and DwoR incur relatively lower packet delays. However, as network load increases, DRENCH performs better than Slick and DwoR in terms of FCT roughly by 10$\sim$20% . Furthermore, we also observe that DwoR provides better FCT than the centralised approach, while DRENCH outperforms both DwoR and Slick. Finally, we also see that the average delay in the case of DRENCH remains low in most cases and close to the other solutions when the network load is extremely high (80%).

In Fig. 6b we see that the average number of NFIs for all the schemes is almost the same. However, DRENCH balances the load more efficiently since the variation in the load among the NFIs and the NFI utilisation is maintained at low levels.

In Figure 6c, we present the specific case for network load of 80%, where we observe that DRENCH is able to get close to Slick, which is optimal in terms of path-stretch deviation. We can also see the benefits of DRENCH redirections in order to correct path-deviation during traffic bursts (see 70-80secs) and also when a large number of flows terminate. In both cases we can see that DRENCH is more effective than DwoR, in terms of keeping path-stretch at a minimum, providing better load-balancing across NFIs and achieving better FCT.
Note: In DRENCH re-directions enable to reroute flows through lightly loaded links and NFIs, thus aid to lower packet delays. However, the FCT might still get affected due to interim packet-reordering, which result in false congestion signals. We believe that a modified TCP stack as in [25] could help mitigate the packet-reordering issue.

D. *Large scale Evaluation: ISP Topology*

To further examine the capability of DRENCH to efficiently use NFIs in a typical WAN ISP environment, we performed simulations with *SimPy*.

[9] https://iperf.fr/

[10] We implemented the shortest weighted path-based flow routing scheme and not the entire Slick runtime [4]

Experimental Setup: We performed simulations on the Rocketfuel AS-1755 (Ebone in Europe) topology.[11] In order to perform a fair comparison, we implemented a greedy heuristic for flow steering which we call `E2+SIMPLE` that uses a combination of E2 [2] for both service instantiation and service-chain path definition, and SIMPLE [6] for flow steering. Note that in terms of performance, the combination of E2 and SIMPLE performs much better than any of the E2 or SIMPLE alone and therefore is the best choice for comparison.

DRENCH vs. E2+SIMPLE: Fig. 7 compares DRENCH and E2+SIMPLE (E2+S) approach. We observe that DRENCH presents higher path deviation (see Fig. 7a) since it deviates from the shortest path in search of non-congested NFIs. In doing so, DRENCH is able to make better use of the available NFIs (see Fig. 7b) and performs instantiation or consolidation when necessary. This way, DRENCH ensures that all the available NFIs are running close to peak utilisation (in terms of throughput served by the NFV nodes) as seen in Fig. 7b. With these design choices, DRENCH achieves significantly lower FCT (see Fig. 7c). In summary, DRENCH provides a staggering $10\times$ improvement in FCT and is able to support roughly $4\times$ higher workloads, while incurring an average path deviation penalty of up to $2\times$ in comparison to the E2+SIMPLE approach. This extra path stretch though is compensated in terms of higher NFI utilisation and in turn, lower FCT.

Summary of Evaluation: To summarize, our evaluation demonstrates that with its hybrid centralised-decentralised decision-making structure, DRENCH can dynamically load-balance traffic and allocate resources according to demand. DRENCH makes informed decisions on the load of NFIs and accordingly instantiates new or consolidates existing NFIs. In turn, traffic is load-balanced (through flow steering and redirections) to the appropriate instance achieving significantly lower FCT.

VII. Conclusion

We have developed a hybrid algorithmic framework for resource management and traffic load-balancing among virtual NFIs that elegantly combines distributed decision-making with centralised control for orchestration and coordination, while performing complex, dynamic service function chaining. DRENCH is designed to dynamically adapt and balance resources utilisation to traffic demand. DRENCH builds on a market-inspired, competition-based `shadow-price` that is used for taking decisions on flow-steering, flow-redirection and service instantiation/consolidation in a distributed manner. A centralised SDN controller performs market orchestration, dissemination of price information and coordination.

Our novel *semi-distributed* approach for dynamic service instantiation and direction of new and existing flows to the least loaded NFV node increases the throughput from each NFV node and in turn reduces FCT significantly. With the help of a prototype implementation on CloudLab and extensive simulations, we illustrate the benefits of using DRENCH, namely that traffic is dynamically load-balanced among instances and the path deviation of flows across the NFIs is kept to a minimum. Resources are efficiently utilised by timely consolidation of NFIs when they are lightly loaded. Overall DRENCH results in almost a $10\times$ reduction in FCT in some of our experiments.

Acknowledgement

This work was supported by EU FP7 Marie Curie CleanSky ITN project Grant No. 607584, the joint EU H2020/NICT ICN2020 Project, Contract No. 723014. and NICT No. 184 and the EPSRC INSP Early Career Fellowship EP/M003787/1.

References

[1] C. Cui and et al., "Network Functions Virtualisation," in *SDN and OpenFlow World Congress*, 2012.

[2] S. Palkar, *et al.*, "E2: A framework for nfv applications," *SOSP 2015*.

[3] A. Gember and et al., "Stratos: A Network-Aware Orchestration Layer for Virtual Middleboxes in Clouds," *Tech. Rep.*, 2013.

[4] B. Anwer *et al.*, "Programming slick network functions," *ACM SOSR 2015*.

[5] T. Wood, *et al.*, "Toward a software-based network: integrating software defined networking and network function virtualization," *IEEE Network 2015*.

[6] Z. A. Qazi *et al.*, "SIMPLE-fying middlebox policy enforcement using SDN," in *SIGCOMM*, 2013.

[7] A. Mohammadkhan and et al., "Virtual function placement and traffic steering in flexible and dynamic software defined networks," in *IEEE LANMAN 2015*.

[8] Z. Cao, M. Kodialam, and T. V. Lakshman, "Traffic steering in software defined networks," *DCC '14*.

[9] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield, "Split/merge: System Support for Elastic Execution in Virtual Middleboxes," in *NSDI 2013*.

[10] S. Kandula, D. Katabi, B. Davie, and A. Charny, "Walking the tightrope: responsive yet stable traffic engineering," *SIGCOMM 2005*.

[11] M. Alizadeh and et al., "CONGA : Distributed Congestion-Aware Load Balancing for Datacenters," *ACM SIGCOMM 2014*.

[12] L. Guo, J. Pang, and A. Walid, "Dynamic service function chaining in sdn-enabled networks with middleboxes," in *IEEE ICNP 2016*.

[13] A. Gember-Jacobson and et al., "Opennf: Enabling innovation in network function control," in *ACM SIGCOMM 2014*.

[14] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: Measurements & analysis," in *IMC 2009*.

[15] H. Jiang and C. Dovrolis, "Why is the internet traffic bursty in short time scales?" in *SIGMETRICS 2005*.

[16] N. Katta *et al.*, "Hula: Scalable load balancing using programmable data-planes," *SOSR 2016*.

[17] F. P. Kelly, a. K. Maulloo, and D. K. H. Tan, "Rate control for communication networks: shadow prices, proportional fairness and stability," *Operational Research Society*, 1998.

[18] R. Cohen and G. Nakibli, "On the computational complexity and effectiveness of "n-hub shortest-path routing"," in *INFOCOM 2004*.

[19] M. Arumaithurai, J. Chen, E. Monticelli, X. Fu, and K. K. Ramakrishnan, "Exploiting icn for flexible management of software-defined networks," in *ACM ICN 2014*.

[20] P. Quinn and U. Elzur, "Network Service Header," Internet-Draft draft-ietf-sfc-nsh-10, 2016, work in Progress.

[21] W. Ding *et al.*, "Openscaas: an open service chain as a service platform toward the integration of sdn and nfv," *IEEE Network 2015*.

[22] N. Dukkipati and N. McKeown, "Why flow-completion time is the right metric for congestion control," *SIGCOMM CCR, 2006*.

[23] A. Botta *et al.*, "A tool for the generation of realistic network workload for emerging networking scenarios," *Computer Networks*, 2012.

[24] S.Kumar *et al.*, "Service Function Chaining Use Cases In Data Centers," *IETF draft*, 2016.

[25] K. He *et al.*, "Presto: Edge-based load balancing for fast datacenter networks," *SIGCOMM CCR 2015*.

[11] http://www.cs.washington.edu/research/projects/networking/www/rocketfuel/interactive/1755eur.html