Amalgam: Distributed Network Control With Scalable Service Chaining

Subhrendu Chattopadhyay
IIT Guwahati
Guwahati, India 781039
subhrendu@iitg.ac.in

Sukumar Nandi
IIT Guwahati
Guwahati, India 781039
sukumar@iitg.ac.in

Sandip Chakraborty
IIT Kharagpur
Kharagpur, India 721302
sandipc@cse.iitkgp.ernet.in

Abhinandan S. Prasad
NIE Mysuru
Karnataka, India 570008
abhinandansp@nie.ac.in

***Abstract*—Management of virtual network function (VNF) service chaining for a large scale network spanning across multiple administrative domains is difficult due to the decentralized nature of the underlying system. Existing session-based and software-defined networking (SDN) oriented approaches to manage service function chains (SFCs) fall short to cater to the plug-and-play nature of the constituent devices of a large scale eco-system such as the Internet of Things (IoT). In this paper, we propose *Amalgam*, a composition of a distributed SDN control plane along with a distributed SFC manager, that is capable of managing SFCs dynamically by exploiting the in-network processing platform composed of plug-and-play devices. To ensure the distributed placement of VNFs in the in-network processing platform, we propose a greedy heuristic. Further, to test the performance, we develop a complete container driven emulation framework *MiniDockNet* on top of standard *Mininet* APIs. Our experiments on a large scale realistic topology reveal that *Amalgam* significantly reduces flow-setup time and exhibits better performance in terms of end-to-end delay for short flows.**

***Index Terms*—Service function chaining, Virtual network function, In-network processing, Programmable network, software defined network**

I. Introduction and Related Works

Due to the rapid deployments of connected environments, large-scale Internet of Things (IoT) networks [1] have become prevalent in recent years. Management of such large-scale heterogeneous ecosystems requires various network services such as network address translator (NAT), firewall, proxy, and local domain name server (DNS); these network services are called network function (NF)s. Generally, the network functions are deployed using virtual machines (VM)(s) to provide service isolation and reduce CapEx and OpEx; therefore, they are termed as virtualized network function (VNF) [1]. VNFs execution require computation platform to host the VM and execute the NF within the VM. Depending on network management policies, the application messages require steering through an ordered set of VNFs known as service function chaining (SFC) [2].

Among various existing architectures to execute VNFs over a network infrastructure [3]–[5] relies on software-defined network (SDN) [6] to steer flows from one VNF to another. On the other hand, [7], [8] takes a session-based approach where the end hosts control the SFC. Session-based approaches achieve lower host-based state management of VNFs, where SDN-based approaches achieve fine-grained quality of service (QoS). However, for a large-scale network spanning across multiple administrative domains, both of the SFC management (SCM) approaches fall short in several aspects as follows.

(a) Lack of scalability: Existing SCMs [6], [9] use a central controller that monitors the resource usage of the devices and use as the basis for the VNFs deployment. The use of a central controller for VNF deployment becomes challenging, especially when the network spans across multiple autonomous administrative domains that interconnected through different network service providers. On the other hand, the VNF placement is $\mathcal{NP}$-hard [10]. Existing distributed heuristics for VNF placement [11] require multiple rounds to deploy VNFs, which increases flow initiation delay leading to reduced IoT application performance since the majority of the IoT flows are short-lived [12].

(b) Dynamic service chaining: Usually, VNFss modifying the headers are common in a large-scale network. Consequently, the participating VNFs can change the SFCs during the lifetime of a flow based on the flow characteristics. For instance, a classifier VNF can add a load balancer based on the arrival rate of the packets in a flow. Existing scalable distributed VNF placement methods [11] and IP based traffic steering proposals [13] are not suitable for dynamic service chaining. On the other hand, [7] ascertains dynamic service chaining by adding an agent in each device, including hosts. Installation of agents on a large scale IoT becomes infeasible, where the devices with plug-and-play capability can dynamically enter and exit the ecosystem.

(c) Issues of flow monitoring over multi-administrative platforms: To steer the traffic through proper service chains while ensuring QoS, requires fine-grained flow monitoring. Existing flow identification methods using packet header fields are insufficient in the presence of a header modifying VNF in the SFC (such as NAT, load balancer, and proxy). Existing SDN-based flow monitoring schemes like FlowTags [9], Stratos [14] utilize *"vlan/mpls"* tagging which does not work through multiple administrative domains. On the other hand, the use of packet encapsulation in session-based approaches

This work was supported by TCS India Pvt. Ltd.
[1]https://www.sigfoxcanada.com/

Annex to ISBN 978-3-903176-28-7 © 2020 IFIP

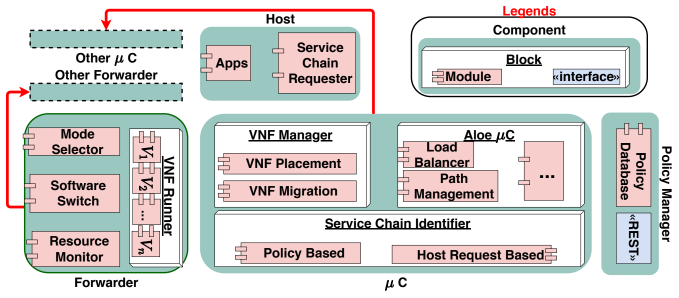

Fig. 1: Component diagram of Amalgam

[8] also fails to ensure QoS.

Therefore, in this paper, we propose *Amalgam* to provide a scalable SFC orchestration platform to provide fine-grained QoS over multiple administrative domains and dynamic SFC. The SCM *Amalgam* provides significant performance improvement in terms of flow initiation delay. The step by step contributions is as follows.

- *Amalgam* achieves distributed state management while ensuring fine-grained QoS by coupling distributed SDN and distributed SCM.
- We develop a distributed heuristic for the distributed deployment of VNFs, which provides performance improvement in terms of flow initiation delay.
- *Amalgam* design exploits the "micro-service"(μS) architecture of the in-network processing platform [15] to support dynamic service chaining and "`zero-touch deployment`" to support the plug and play nature of the devices.
- For performance evaluation, we develop an emulation framework *MiniDockNet* for VNF deployment using "`docker`" [2] over in-network processing, as the existing network name-space oriented mininet [3] emulator is not sufficient for in-network processing.
- Based on the experimental results over a realistic large-scale system (with 70 devices and 6 different service chain scenarios), we found that *Amalgam* can ensure fine-grained QoS without significant increase of resource utilization of the devices. After comparing the performance of *Amalgam* with two state-of-the-art distributed SFC platform *"Dysco"* [7] and *"WGT"* [11] we found that, *Amalgam* is capable of a significant reduction in the flow initiation delay and improves the performances of short-duration flows in terms of end-to-end delay.

This paper is organized as follows. Section II describes the architecture of proposed *Amalgam*. Section III describes the design choices taken during the development of *Amalgam*. We describe the experimental setup details, results and our observations in Section IV before concluding in Section V.

II. Architecture

The proposed *Amalgam*[4] is constructed on the top of *Aloe* [16] framework. *Aloe* provides an orchestration framework for a fault-tolerant self-stabilizing distributed control plane on top of the in-network processing platform using μCs (microcontrollers) instead of the standard SDN controller. Like *Aloe*, *Amalgam* supports 3 operating modes for each device; (a) **Host/(default) mode**: if the device executes only the client/server application, (b) **Forwarding mode**: If the device has multiple active network interfaces, (c) μ**C mode**: based on the topology, if *Aloe* selects the device as a μC. At any instant, each device is in "`atleast`" any one of the above modes and each mode is a component of *Amalgam* framework as shown in Fig. 1. A mode selector module in *"forwarder"* component identifies the mode of operation and activates the respective components. Apart from the mode functional components, we add a seperate *"policy manager"* component in the *Amalgam* framework. Policy manager is a distributed database with a *"REST"* driven interface which contains (i) the flow identifier (i.e., "`OpenFlow`" match field) and (ii) ordered list of the types of VNF (service chain) through which the flow should be steered. This component is consulted by the mode functional components at time of determination of SFC.

Among the mode functional components, the host is the simplest. This component is responsible for traffic generation through the *"App"* module, which represents the client/server applications. Additionally, this module can request the nearest μC to change the SFC for the flows generated by the host. The *"mode selector"* module elevates the mode of the device with multiple active interfaces to *"forwarder"*. The forwarder component consists of *"software switch"*, *"VNF runner"*, and a *"resource monitor"*. The software switch module is responsible for forwarding data from one interface to another. On the other hand, the *"VNF runner"* block is reserved for execution of VNFs (e.g., V_1, V_2 etc. as shown in Fig. 1). This VNF runner block from each device constructs the in-network processing framework. During the execution of the VNFs *"resource monitor"* module periodically monitors the available resources in the device. The resource monitor module forwards the collected resource utilization statistics to the μC component.

The μC component is composed of three functional blocks namely Service chain identifier (SCI), "*"VNF manager"*", and "*Aloe* μC". The tasks of these blocks are as follows.

1) Service Chain Identifier: At the startup phase of the μC, SCI caches the policy in a local cache. The local cache is updated whenever the policy manager database is updated. SCI module is consulted when an "`OpenFlow`" "`packet in`" event is initiated at the μC. From the list of VNFs in the service chain, SCI chooses the first VNF, and it's execution status in the local domain. If the VNF is executing inside a forwarder connected to the μC, the SCI consults the path management module to establish the data flow path by installing flow table entries via standard "`OpenFlow`" protocol. Otherwise, it sends a search query to the other μCs to identify the target VNF address. If the address of the VNF is not found, then SCI consults the VNF manager module (Section II-2) to start the execution of the VNF. This procedure is iterated for all the VNFs in the service chain.

[2]https://www.docker.com/
[3]http://mininet.org/
[4]https://github.com/subhrendu1987/NFV_MiniDockNet

2) VNF Manager: The VNF manager module (VMM) works in a distributed fashion and communicates with the neighbor μCs. VMM tries to answers the following two questions; (a) should the VNF be placed in any of the forwarders associated with the μC? (b) which forwarder should take care of the VNF?. The detailed protocol to find an answer to these questions is described in next section. Additionally, VMM also takes care of the dynamic addition or removal of the VNFs to an ongoing flow.

3) Aloe μC: The *Aloe μC* module is the containerized SDN μ controller module as described in [16]. *"Aloe"* provides a distributed fault-tolerant controller module suitable for in-network processing frameworks that are responsible for path management. Use of *Aloe* ensures quick flow initiation along with fault and partition tolerance in *Amalgam*. However, during the design of *Amalgam*, we face several challenges exclusive to SFC deployment of IoT

III. Challenges and Design Choices

The goal of *Amalgam* is to provide a highly dynamic in-network processing platform. In this section, we describe the implementation challenges and the proposed solutions to overcome the scalability issues without affecting the dynamic behavior of the platform.

(a) Plug-and-Play Capability:A typical IoT platform is composed of plug-and-play devices where "`zero-touch deployment`"[5] is highly desired. It is necessary to configure a new device as soon as it enters the eco-system. To avoid individually configuring the devices, we design each component of *Amalgam* (except the host component) as Docker containers. Once a device enters the eco-system, it assumes the host mode of operation. Since the host mode does not require anything more than the IoT applications (clients and servers), they can work smoothly. Whenever the device wishes to change its mode, it can pull the container image of the *Amalgam* component from the nearest forwarder.

(b)Distributed VNF Placement: In a short-lived flow heavy system, minimization of the flow initiation delay is critical. The flow initiation delay consists of following components namely (a) Controller consultation delay (b) SFC deployment delay, and (c) path setup delay. The proposed VNF placement reduces the SFC deployment delay. A SFC for a particular flow is composed of multiple VNFs, which requires resource consumption. Each device of a IoT in-network processing platform has residual resources that can be used for deployment of these VNFs's. The proposed VNF placement identifies the set of devices where the VNFs of the SFCs can be placed for a given network and flow profile while satisfying the capacity constraints of the devices. Maintaining capacity constraints in a multi-domain system is non-trivial since the residual capacity of a device residing in a different administrative domain is difficult to collect. Therefore, we propose the greedy heuristic as given in the Algorithm 1.

Algorithm 1: Distributed Placement of VNF

```
1  Function GreedyPlace(Path: $P^a$, Service Chain: $C^j$, $\mu$C: $l$):
2      Find ordered set of unplaced VNFs from $C^j$;
3      $I \leftarrow \{i : i \in P^a, \varphi_i = l\}$;
4      Place as many VNFs as possible among $I$;
5      return number of VNFs placed;
6  Function Main(Flow: $f^j$,$\mu$C: $l$):
       /* Find VNF placement profile for $f^j$ in $\varphi_i$ */
7      Find set of paths ($P$) from $s_j$ to $d_j$ by querying "Path Management"
         module of $l$;
8      $\underset{P^a \in P}{maximize}$ GreedyPlace($P^a$, $C^j$, $l$);
9      if $\exists c_{j,k} \cdots c_{j,j_{max}}$ not placed then
           /* All devices under $l$ */
10         Obtain the list of adjacent $\mu$C of $l$ and store it in $N\mu$foreach
             $l' \in N\mu$ do
11             Main($f^j$,$l'$);
12     return;
```

Each μC in the end-to-end path (P) executes the proposed heuristic for each flow (f^j represents jth flow) from source (s_j) to destination (d_j). We denote SFC of f^j with C^j. Certain μC with ID l maintains the topology information as the list of devices (D_l) and list of links (E_l) where each link $e_{i,i'} \in E_l$ represents the physical connection between two devices (i and i'). For the sake of simplicity, we denote the μC associated with $i$as φ_i. The proposed heuristic identifies a path P^a between s_j to d_j from the set of P such that, most of the VNFs of C^j are placed near s_j in a distributed fashion. This way, one μC does not need the resource utilization of devices from other administrative domains. Once the flow is established, the resource utilization of devices in the path (`info`) is piggybacked with the data packets. The VNF manager can re-solve the Algorithm 1 and find a new allocation of VNF with updated utilization.

(c)Migrations of the VNFs: A VNF may be relocated during (a) VNF readjustment due to prior sub-optimal placement and (b) addition or removal of the device. The μC nearest to the source node of the flow decides the VNF readjustment after it receives the piggybacked resource utilization of the devices. On the other hand, the addition and removal of devices trigger "`topology_change`" event, and the local μC initiates the decision about the VNF deployment. In both cases, the decider μC starts the migration process at the source device. Initially, the source device saves a snapshot of the executing container, and the snapshot is transferred and restored in the target device. Finally, the μC updates the existing flow table entries accordingly.

(d)Dynamic management of service chains: *Amalgam* provides support for dynamic service chaining. Dynamic service chaining enables VNFs to meet changing service requirements. For instance, consider a flow passes through a firewall VNF. Based on the signature of the flow, the firewall conditionally decides to steer the flow through an additional deep packet inspector (DPI) without interrupting the flow. To implement this, *Amalgam* allows the VNFs to interact with the local μC via "`REST`" interface. The local μC can deploy the DPI if it is not available and sends the "`OFPT_FLOW_MOD`" events to

[5]https://www.etsi.org/technologies/zero-touch-network-service-management

the forwarder component to enable the flow steering without terminating the flow.

(e)Flow Tags for Monitoring: Once the VNFs are placed, the path management module of μCs set-up the flow table entries of the participating forwarders via `OpenFlow` protocol. One issue regarding path management through service chains is to identify an end-to-end flow that arises in the presence of the "5-tuple" changing VNFs (e.g., Load balancer, web proxy cache, and NATs). Since such VNFs may alter the packets in unpredictable ways, fine-grained management and monitoring of the flows passing through becomes difficult. To avoid this issue, *Amalgam* attaches a "`VLAN`" tag to the packets before it enters the VNF. The nearest μC of the VNF maintains a table for flows like f^a, which keeps track of the original match field of the flow and the modified field (alias).

(f)Providing QoS: *Amalgam* is developed on top of the SDN decentralized control plane, which enables us to ensure flow specific QoS guarantees. On the other hand, since the VNF deployment is done using containers, using "`cgroups`"can ensure the VNF specific QoS like reservation of CPU, Memory, etc. The policy server module contains the "`cgroups`" parameters for each VNFs of a service chain, which is used to ensure VNF specific QoS.

IV. Prototype and Experimental Results

The existing namespace oriented emulation frameworks (e.g. Mininet) is not suitable for VNF migration and in-networking platform emulation. Therefore, we develop docker based MiniDockNet which mimics real-life VNFs using the *"Docker-in-Docker"* configuration. This feature ensures rapid deployment from MiniDockNet emulation to real in-network processing environment. To implement the VNF migration, we use standard live container migration using CRIU[6]. For the emulation of the links between any two nodes, we use *"l2tp"*

A. Experimental Setup

For experimental purpose, we use "`rocketfuel topology`"[7]. Each link is configured to emulate $3ms$ of delay and $10Mbps$ of bandwidth using linux "`tc`" utility. We use "`iperf`" to generate long flows; for shorter flows we use "`ping`". The clients and server applications are hosted on the *diameter* of the topology. For background traffic we use python based "`HTTP`" client and server.

We use "`Apache cassendra`" to implement the policy server module. Rest of the *Amalgam* modules targeted for μC are implemented on top of "`Ryu`"[8], a python based SDN controller framework. For experiments, we use 3 different VNFs (NAT (N), Load Balancer (L) and Web Proxy(W)) to create 6 different combination of service chain as given in Fig. 5. In order to ensure the confidence on the results, each experiment is repeated atleast 30 times.

[6]https://criu.org/Live_migration

[7]http://tiny.cc/nv70mz

[8]https://ryu.readthedocs.io/en/latest/

B. Results

We compare the performance of *Amalgam* with the existing *"P4"*[9] based distributed session-oriented service function chaining framework called Dysco [7]. Since, Dysco ensures session related performance and does not provide any VNF placement strategy, the performance evaluation of the proposed distributed VNF placement algorithm is done with another existing work WGT [11] which proposes a distributed heuristic for VNF placement for the multi-domain network.

1) Session Related Performances: Fig. 2 shows the comparison between Dysco and *Amalgam* in terms of flow initialization delay. We found that *Amalgam* is capable of quicker flow initialization than Dysco. This reduction in flow initialization delay comes from the parallel deployment of VNFs as opposed to the hop by hop deployment of VNFs in Dysco. The advantage of flow initialization delay becomes much evident in the case of longer service chains like C^6 than the smaller service chain like C^1.

Since *Amalgam* uses containers to deploy the VNFs as opposed to the P4 applications used in Dysco, the deployment of VNFs using *Amalgam* incurs greater latency, as shown in Fig. 3. The increase in VNF deployment time for *Amalgam* depends on the VNF container size. Therefore, the deployment latency is higher for C^6 in compared to C^1. However, in a large scale network, VNF deployment events are far rare than a flow generation event. On the other hand, the use of containers provide greater flexibility as the creation of new middlebox application using container requires less programming overhead than the creation of a new P4 application. As a result, state management during the migration of VNFs from one node to another becomes easy when they are running inside a container as compared to the P4 applications of Dysco. However, these management benefits of containers come at the cost of resource utilization.

The placement of VNFs requires resource occupancy in the deployed devices, which is an important aspect of resource constraint IoT devices. In Fig. 6, we compare the performance of *Amalgam* with Dysco in terms of CPU utilization of devices due to the placement of VNFs. In order to normalize the additional resource consumption of *Amalgam* due to the use of containers, we also compare the resource utilization of *Amalgam* without using docker. Similarly, we provide a comparison of memory utilization for *Amalgam* and Dysco in Fig. 7. Based on these two experiments, we observe that Dysco incurs less utilization of resources than the proposed *Amalgam* with the container. However, based on the *"Wilcoxon Rank Sum test"* we find that, the difference of resource utilization of *Amalgam* without Docker and Dysco is statistically insignificant (i.e. $p-value >$ 0.05) for C^4, C^5 and C^6. Fig. 8 shows the comparison of throughput between *Amalgam* and Dysco. The Wilcoxson rank sum test reveals that the throughput between *Amalgam* and Dysco are statistically indistinguishable (Here our alternate hypothesis H_a is *Amalgam* provides less throughput than Dysco).

[9]https://p4.org/

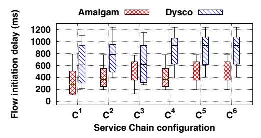

Fig. 2: Flow Initialization Delay

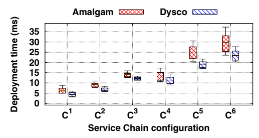

Fig. 3: Latency of Deployment

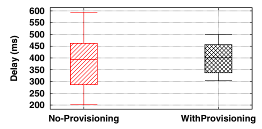

Fig. 4: Effect of QoS

Name	SFC		Name	SFC
C^1	(N)		C^2	(L)
C^3	(W)		C^4	(N→L)
C^5	(L→W)		C^6	(N→L→W)

Fig. 5: List of Service Chains

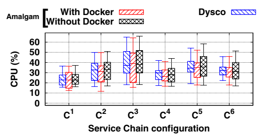

Fig. 6: CPU Utilization

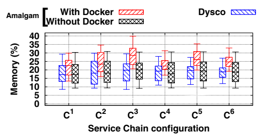

Fig. 7: Memory Utilization

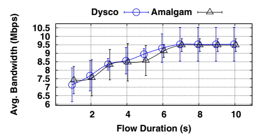

Fig. 8: Average Throughput

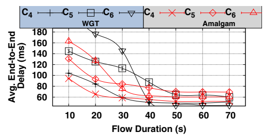

Fig. 9: Delay for C^4, C^5 and C^6

2) Performance of Distributed VNF Placement: To measure the performance of distributed VNF placement heuristic used in Amalgam, as mentioned earlier, we deploy WGT [11] on top of Dysco. However, it is difficult to deploy a centralized controller for a large scale multi-domain system. Therefore, we place the WGT in the micro-Controller (μC) nearest to the source device. We measure and compare the effect of delay for all the service chains when the flow duration increases. Based on the experimental results, we found that the effect of delay for single VNF does not change since *Amalgam* and WGT provides the same results for VNF placement. Hence, we omit the plots for C^1, C^2, C^3. For multiple VNF oriented service chains like C^4, C^5 and C^6, we provide the average end-to-end delay in Fig. 9. Based on the results, we can observe that *Amalgam* can perform significantly well for shorter flows as the iterative WGT requires a significant amount of feedback rounds to find the proper placements of VNFs.

C. QoS Provisioning

Amalgam is capable of showing QoS provisioning by reserving resources limiting CPU, memory, bandwidth, and link delay. We perform two experiments for each resource type, one with no provisioning and another with resource reservation limit set as the mean value found in the previous experiment. Based on the Wilcoxson rank-sum test on these results we found that, except the memory utilization (P-value $= 0.42$) rest of the resource reservation works significantly well (with P-value< 0.05). We also find that the resource reservation can reduce the jitter of the flow, as shown in Fig. 4.

V. Conclusion

In this paper, we present *Amalgam*, which integrates the distributed SDN orchestration framework with the distributed service chain management framework. The proposed *Amalgam* is suitable for large scale multi-domain IoT in-networking platforms. We also provide a distributed heuristics for the placement of constituent VNFs of service chains. The lack of an existing emulation platform for container oriented VNF service chain has motivated us to develop "MiniDockNet". Using this emulation platform, we found that *Amalgam* incurs a lesser flow initialization delay than that of a very recent distributed service chain management framework (Dysco). We also show that *Amalgam* is capable of ensuring less end-to-end delay for short flows.

References

[1] F. Duchene, D. Lebrun, and O. Bonaventure, "Srv6pipes: enabling in-network bytestream functions," in *17th IFIP Networking Conference and Workshops*, May 2018, pp. 1–9.
[2] W. Haeffner, J. Napper, M. Stiemerling, D. Lopez, and J. Uttaro, "Service function chaining use cases in mobile networks," *IETF*, 2015.
[3] D. Bhamare, R. Jain, M. Samaka, and A. Erbad, "A survey on service function chaining," *J. Netw. Comput. Appl.*, vol. 75, no. C, pp. 138–155, nov 2016.
[4] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield, "Split/merge: System support for elastic execution in virtual middleboxes," in *10th USENIX Symposium on NSDI*. Lombard, IL: USENIX, 2013, pp. 227–240.
[5] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "OpenNF: Enabling innovation in network function control," *SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 163–174, 2015.
[6] A. Mohammadkhan, S. Ghapani, G. Liu, W. Zhang, K. K. Ramakrishnan, and T. Wood, "Virtual function placement and traffic steering in flexible and dynamic software defined networks," in *The 21st IEEE International Workshop on LAN/MAN*, April 2015, pp. 1–6.
[7] P. Zave, R. A. Ferreira, X. K. Zou, M. Morimoto, and J. Rexford, "Dynamic service chaining with dysco," in *30th Proceedings of the ACM SIGCOMM*. ACM, 2017, pp. 57–70.
[8] IETF, "Rfc 8300 - network service header (NSH)," Sept 2019, [accessed 10. Sept. 2019]. [Online]. Available: https://tools.ietf.org/html/rfc8300
[9] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags," in *11th USENIX Symposium on NSDI*. Seattle, WA: USENIX Association, 2014, pp. 543–546.
[10] M. Ghaznavi, N. Shahriar, S. Kamali, R. Ahmed, and R. Boutaba, "Distributed service function chaining," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2479–2489, Nov 2017.
[11] G. Sun, Y. Li, D. Liao, and V. Chang, "Service function chain orchestration across multiple domains: A full mesh aggregation approach," *IEEE Transactions on Network and Service Management*, vol. 15, no. 3, pp. 1175–1191, Sep. 2018.
[12] M. Z. Shafiq, L. Ji, A. X. Liu, J. Pang, and J. Wang, "Large-scale measurement and characterization of cellular machine-to-machine traffic," *IEEE/ACM Transactions on Networking*, vol. 21, no. 6, pp. 1960–1973, Dec 2013.
[13] A. Wion, M. Bouet, L. Iannone, and V. Conan, "Let there be chaining: How to augment your igp to chain your services," in *18th IFIP Networking Conference*, May 2019, pp. 1–9.
[14] A. Gember, A. Krishnamurthy, S. S. John, R. Grandl, X. Gao, A. Anand, T. Benson, A. Akella, and V. Sekar, "Stratos: A network-aware orchestration layer for middleboxes in the cloud," *arXiv CoRR*, vol. abs/1305.0209, 2013.
[15] M. Charikar, Y. Naamad, J. Rexford, and X. K. Zou, "Multi-commodity flow with in-network processing," in *Algorithmic Aspects of Cloud Computing*. Cham: Springer International Publishing, 2019, pp. 73–101.
[16] S. Chattopadhyay, S. Chatterjee, S. Nandi, and S. Chakraborty, "Aloe: An elastic auto-scaled and self-stabilized orchestration framework for iot applications," in *37th IEEE Proceedings of INFOCOM*. Paris, France: IEEE, 2019, pp. 802–810.