Amalgam: Distributed Network Control With Scalable Service Chaining

Subhrendu Chattopadhyay  
IIT Guwahati  
Guwahati, India 781039  
suhrendu@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 centralized 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 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, VNFs 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.

1https://www.sigfoxcanada.com/

Annex to ISBN 978-3-903176-28-7 © 2020 IFIP
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., V1, V2 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) **VNF Manager:** The VNF manager module (VMM) works in a distributed fashion and communicates with the neighbor µCs. VMM tries to answer 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” 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 placement 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’.

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

```plaintext
Function GreedyPlace (Path: P, Service Chain: C, µC: l):
1. Find ordered set of unplaced VNFs from C.
2. \( I \leftarrow \{ i : i \in P, \phi_i = l \} \).
3. Place as many VNFs as possible among I.
4. return number of VNFs placed;
5. max GreedyPlace (P, C, µC).

Function Main (Flow: f, µC: l):
1. /* Find VNF placement profile for \( f \) in \( \phi_l \).*/
2. Find set of paths (P) from \( s_j \) to \( d_j \) by querying “Path Management” module of l;
3. maximize GreedyPlace (P, C, µC).
4. if \( \exists c_{j,k} \ldots c_{j,k} \not\in \varphi_l \) then not placed then
5. /* All devices under I*/
6. Obtain the list of adjacent \( \mu C \) of l and store it in \( N_{p_i} \) for each
7. \( \phi_i \in N_{p_i} \) do
8. Main (f, l');
9. return;
```

Each \( \mu C \) in the end-to-end path (P) executes the proposed heuristic for each flow \( f \) (i.e., \( \phi_l \) in the flow) from source \( (s_j) \) to destination \( (d_j) \). We denote SFC of \( f \) with \( C \). Certain \( \mu C \) with ID \( l \) maintains the topology information as the list of devices \( (D_l) \) and list of links \( (E_l) \) where each link \( e_{i,j} \in E_l \) represents the physical connection between two devices \( i \) and \( j \). For the sake of simplicity, we denote the \( \mu C \) associated with \( \phi_l \). The proposed heuristic identifies a path \( P \) between \( s_j \) to \( d_j \) from the set of \( P \) such that, most of the VNFs of \( C \) are placed near \( s_j \) in a distributed fashion. This way, one \( \mu C \) 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 \( \mu C \) nearest to the source node of the flow decides the VNF readjustment after receiving the piggybacked resource utilization of the devices. On the other hand, the addition and removal of devices trigger “topology_change” event, and the local \( \mu C \) initiates the decision about the VNF deployment. In both cases, the decision \( \mu C \) 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 \( \mu C \) 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 \( \mu C \) via “REST” interface. The local \( \mu C \) can deploy the DPI if it is not available and sends the “OFPT_FLOW_MOD” events to

---

3https://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 $f^{n}$, 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 interworking platform emulation. Therefore, we develop docker based MiniDockNet which mimics real-life VNFs using the “Dock-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. 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 “cc” 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 cassandra” to implement the policy server module. Rest of the Amalgam modules targeted for μCs are implemented on top of “Ryu”, 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.

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 Wilcoxon 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).

---

https://criu.org/Live_migration
https://tiny.cc/m70mnz

---

7https://p4.org/
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 provide 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 Wilcoxon 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