Contain-ed: An NFV Micro-Service System for Containing e2e Latency

Amit Sheoran*
Purdue University
asheoran@purdue.edu

Puneet Sharma
Hewlett Packard Labs
puneet.sharma@hpe.com

Sonia Fahmy
Purdue University
fahmy@purdue.edu

Vinay Saxena
Hewlett Packard Enterprise
vinay.saxena@hpe.com

ABSTRACT

Network Functions Virtualization (NFV) has enabled operators to dynamically place and allocate resources for network services to match workload requirements. However, unbounded end-to-end (e2e) latency of Service Function Chains (SFCs) resulting from distributed Virtualized Network Function (VNF) deployments can severely degrade performance. In particular, SFC instantiations with inter-data center links can incur high e2e latencies and Service Level Agreement (SLA) violations. These latencies can trigger timeouts and protocol errors with latency-sensitive operations.

Traditional solutions to reduce e2e latency involve physical deployment of service elements in close proximity. These solutions are, however, no longer viable in the NFV era. In this paper, we present our solution that bounds the e2e latency in SFCs and inter-VNF control message exchanges by creating *micro-service aggregates* based on the affinity between VNFs. Our system, *C*ontain-ed, dynamically creates and manages affinity aggregates using light-weight virtualization technologies like containers, allowing them to be placed in close proximity and hence bounding the e2e latency. We have applied Contain-ed to the Clearwater [1] IP Multimedia Subsystem and built a proof-of-concept. Our results demonstrate that, by utilizing application and protocol specific knowledge, affinity aggregates can effectively bound SFC delays and significantly reduce protocol errors and service disruptions.

CCS Concepts

•**Networks** → **Data center networks;** *Mobile networks;* Network protocol design;

Keywords

Network Functions Virtualization; Containers

1. INTRODUCTION

In traditional deployments of large carrier-grade systems, network service elements (functions) execute on hardware with dedicated CPU, memory and storage resources. The hardware boxes are connected via high speed links in operator data centers (DCs). Since the network is purpose-built to handle predefined network elements (NEs) and workload, the deployment is optimized to meet service requirements [18]. This includes allocating adequate resources and carefully placing NEs to meet latency requirements. NEs that constitute a Service Function Chain (SFC) or, more generally, a forwarding graph, are deployed in the same data center, and are carefully configured to meet Service Level Agreements (SLAs) or Quality of Service (QoS) requirements.

Network Functions Virtualization (NFV) leverages Commercial off-the-shelf (COTS) hardware to dynamically deploy network services. New network service instances are created by adding NEs to existing SFCs using virtualization and programmable networking technologies such as Software Defined Networking (SDN). NFV orchestration frameworks can instantiate these Virtualized Network Functions (VNFs) on-demand. The eventual placement of these VNFs is a balancing act by the orchestrator to meet both the QoS requirements of the deployed service and the need for cloud providers to maximize the utilization of the underlying infrastructure. Owing to the operational polices of the orchestrator and the physical locations of the data centers that it manages, new NE instances may be located on different racks or even different data centers. This, coupled with the unpredictable latency variations due to the sharing of the underlying physical infrastructure among services, can cause violations of end-to-end (e2e) latency requirements of SFCs [14]. Distributed instantiations of SFCs and latency variations can cause significant performance degradation since current applications and network protocol stacks are designed for traditional deployments and therefore react poorly to such "unbounded" latencies.

In systems like Evolved Packet Core (EPC) and IP Multimedia Subsystems (IMS) where multiple NEs participate in service delivery, congestion on any interconnecting link triggers message drops or retransmissions. Constituent NEs often aggressively retransmit latency-sensitive messages to ensure timely execution of the protocol call flows [9]. Such re-transmissions aggravate network conditions, leading to

*Work funded by Hewlett Packard Labs and done during Amit Sheoran's internship. This work has been presented at the ACM SIGCOMM 2017 1st International Workshop on Hot Topics in Container Networking and Networked Systems (HotConNet).

further QoS deterioration. Since a single event/action can produce multiple message exchanges among the constituent elements in an SFC, an orchestrator must consider, when placing the SFC elements, the type and frequency of message exchanges among the SFC elements – a factor which is not considered by current orchestration frameworks.

A natural solution to control latency with NFV is to instantiate service elements within an SFC onto physical machines that are in close proximity. This ensures that congestion in other parts of the DC, as well as latency due to inter-DC communication, can be avoided. However, such a placement policy will force cloud provides to preallocate VNF resources in designated sections of the DC, ultimately undermining their ability to maximize infrastructure resource utilization. Even if such a policy can be implemented, the footprints of current VM-based VNFs are far too large to guarantee close proximity allocation. VNFs also support mobile users: even if a user is assigned to an NE where all SFC elements meet latency demands, user mobility makes it impossible to sustain such assignments. The mobile user can move across geographic regions, and this generally entails handover of the user session to NEs physically closer to the user location, which will inevitability result in user traffic traversing multiple data centers.

In this paper, we explore the design of a small-footprint, stateless and portable VNF solution based on aggregating micro-services. Our solution, Contain-ed, meets latency demands while simultaneously supporting user mobility and elastic resource allocation. Contain-ed aims to:

1. Bound e2e service latency by creating collocated aggregates of NEs.

2. Develop a service-aware, latency-sensitive orchestration and deployment framework at a low cost to the provider.

More information, including our scripts, can be found at http://www.cs.purdue.edu/homes/fahmy/contained/

2. CONTAIN-ED ARCHITECTURE

Our design is guided by two key observations: **(1) VNF Affinity:** The number of messages among VNFs depends on the standards being used and the SFC structure. For example, in a virtualized EPC system [19], 41% of the signaling messages that are incident on the Mobility Management Entity (MME) are propagated to the Serving Gateway (SGW), but only 18% of the MME signaling load is propagated to the Packet Data Network Gateway (PGW). Protocol message exchanges and/or SFC dependencies enable us to identify affinity between VNFs or VNF components (VNFCs). **(2) Transactional Atomicity:** Transactions are sequences of messages that are exchanged among VNFs/VNFCs to handle a network event. Table 1 enumerates common network events in the IMS and EPC systems. These network events are often a result of user actions and each independent action (e.g., REGISTER) can trigger a sequence of messages. VNFs involved in processing a user's messages generally allocate/update state information, and this state information is used to process future messages of this user. The state information is either stored locally (in traditional network designs) or in shared storage (in NFV based designs). We observe that, due to state dependencies, user messages that are part of a specific transaction are, in general, processed by a specific VNF instance. However, once the transaction is completed, this state information can be shared with other VNF instances to handle future transactions.

Contain-ed leverages these observations as follows. Affinity dictates that certain VNFs in an SFC or VNFCs in a complex VNF be placed in close proximity to meet e2e latency requirements. The smaller resource footprint of virtualization technologies like containers enables micro-service bundles of VNFs with high affinity to be placed near each other. Contain-ed creates network micro-service bundles called *Affinity Aggregates (AAs)*. AAs are bundles of network services comprising VNFs that have message exchange affinity towards each other. AAs are instantiated as a single logical entity of micro-services using lightweight virtualization technologies (containers). Each AA is configured to handle a predetermined transaction type and only consists of VNFs/VNFCs involved in processing this transaction type. Contain-ed includes components for managing and orchestrating AA instances, with the goal of distributing load across active AA instances and resource flexing according to workload variations. Figure 1 illustrates the Contain-ed architecture, whose components we now describe.

Affinity Analytics Engine (AAE): The AAE is an offline module that analyzes SFC dependencies and message exchange sequences to determine the required AA types. The AAE uses the VNF affinity information derived from analyzing the VNF messages exchanges to decide which VNFs should be bundled together as an AA. The AAE also determines transactional boundaries so that the same AA instance is used to handle a transaction's message exchange sequence in an atomic manner. Additionally, the AAE determines what to store in the shared state store across all AA instances.

A single transaction can generate significant amounts of intermediate state information, based on the structure of the SFC and the protocols involved. While it is possible to publish all intermediate state information generated by an AA to the shared state store, such a design would lead to significant performance degradation due to increased message exchanges between the AAs and the state store. Furthermore, all intermediate state information is not required by the VNFs/VNFCs to handle independent transactions. As an example, the MME processes 10 of the 18 messages generated during the EPC Attach procedure [19], and each of these message exchanges is capable of generating intermediate state information. However, if the AA instance that handles the Attach request does not change during Attach procedure, there is little merit in publishing intermediate state information to the state store. The AAE therefore leverages transactional boundaries to determine the minimum state information that must be shared across AA instances. Only state information that persists across transactions is published in the state store.

Contain-ed transactions are specific to an SFC, and the messages that constitute a transaction are driven by protocol bindings within the SFC. Example message exchanges for IMS and their transaction boundaries are shown in Figure 3. Table 1 lists the AA types from our analysis of IMS and EPC protocol message exchanges and latency requirements. When AAs have the same VNFs, the same AA type can be used to handle different kinds of transactions/network events. For services such as Home Subscriber Server (HSS), Policy and Charging Rules Function (PCRF) and Online

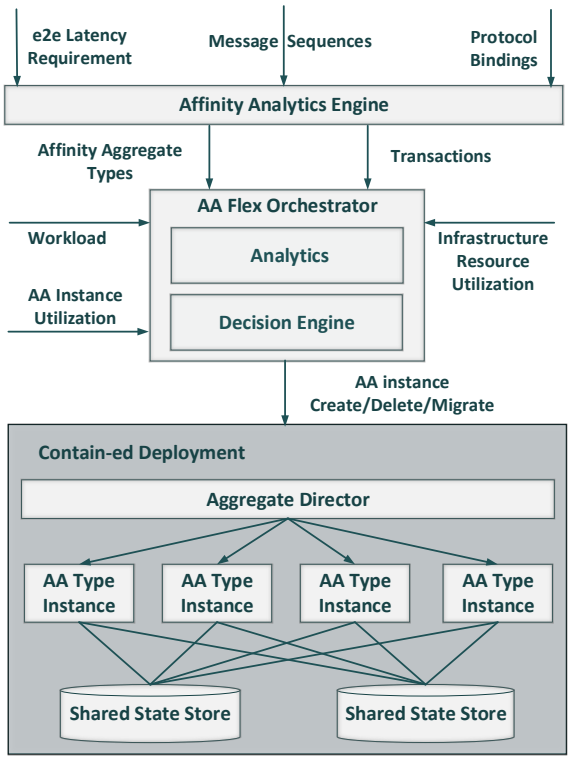

Figure 1: Contain-ed Architecture

Charging System (OCS) that need database lookups, the Front End (FE) component [8] can be instantiated with the AA. The REGISTER AA can also contain the Application Server (AS) if specified in the Initial Filter Criteria (iFC) [6, 7].

AA Flex Orchestrator (AFO): The AFO manages the life-cycle of AA instances. It continuously monitors their resource usage and workload. If the latency requirements of a specific request type are not being met, the AFO deploys new AA instances with appropriate resources and at appropriate locations to meet the latency requirements. Conversely, if the workload decreases, the AFO removes unneeded AA instances after migrating active user sessions to other active AAs. Contain-ed transactions are short-lived compared to user sessions, which enables the AFO to elastically manage the resource allocation for the incoming workload. AA instance information is communicated to the AA Director for forwarding transactions. For example, all VNFs that participate in user registration can be bundled into a REGISTER AA type. Depending on the allocated resources (hence capacity of the AA type) and expected peak load, the AFO determines the number of instances of this AA type to deploy and how/when to add/remove instances to match workload dynamics.

AA Director (AD): The AD is an online module that directs incoming traffic to different active AA instances based on transaction types. The AD maintains a list of all active AA instances and their capabilities, and directs traffic (along transaction boundaries) accordingly. Multiple instances of a particular AA type can coexist with different resource allocations. When a new AA instance is spawned, the AFO updates the AD with its AA type and resource allocation.

Table 1: IMS and EPC Affinity Aggregates (AAs)

Network Event	VNFs in AA	AA Type
IP Multimedia Subsystem (IMS)		
REGISTER	P/I/S-CSCF HSS	REGISTER
INVITE	P/I/S-CSCF AS, OCS	INVITE
NOTIFY SUBSCRIBE	P/I/S-CSCF	SUBSCRIBE
Evolved Packet Core (EPC)		
ATTACH	MME, HSS, SGW PGW, PCRF	ATTACH-DETACH
DETACH	MME, HSS, SGW PGW, PCRF	ATTACH-DETACH
HANDOVER	MME, SGW	HANDOVER-SR
BEARER SETUP	MME, SGW PGW, PCRF	BEARER-CRT
SERVICE REQUEST (SR)	MME, SGW	HANDOVER-SR

This enables the AD to intelligently load-balance the incoming workload on available AA instances. The AD analyzes each incoming packet to classify it according to the AA types and transaction boundaries. All messages associated with a particular transaction (*e.g.*, messages that are part of a single user registration request) that were handled by a specific AA instance will continue to be directed to the same instance until the transaction is completed. This implies that AAs can only be deleted when there are no active transactions pending. When the AFO decides to scale-in an AA instance, the AD removes it from the active list and stops sending new transactions to it.

A single AD instance is capable of handling incoming traffic for multiple AA types. In cases where these AAs are part of different systems, such as the EPC and the IMS, which use different signaling protocols, the AD has to support multiple protocols. The AD is not, however, required to understand all the protocols that are used within the SFCs. For example, an analysis of the AAs in Table 1 reveals that all *inbound messages* for the IMS AAs use the Session Initiation Protocol (SIP) [3]. Similarly, AAs in the EPC system use the GPRS Tunneling Protocol (GTP-C) [11] for all *inbound messages*. Therefore, an AD that handles both EPC and IMS traffic using the AAs described in Table 1 is only required to support the SIP and GTP-C protocols.

Shared State Store (SSS): The shared state store is used by AA instances to store persistent state information across transaction boundaries. This allows incident workload to be distributed across multiple AA instances. The SSS is implemented as a key-value store and is agnostic to the actual structure/definition of state elements as specified by VNFs. The AAs use a simple Representational State Transfer (REST) based interface to store/fetch the sate information. Several VNFs (including Clearwater which we use in our evaluation) already support persistent state information management for horizontal scaling of individual components. The SSS can be deployed as a geographically redundant cluster when a single instance cannot handle the workload. The AFO can create multiple instances of the SSS in case the data store/fetch latency exceeds a predetermined threshold.

Contain-ed determines VNFs/VNFCs that handle messages of a specific transaction type and deploys these VNFs

or VNFCs as a single AA. For example, Contain-ed can create an AA that handles only REGISTER messages (REGISTER-AA) and another that only handles messages of type INVITE (INVITE-AA) as described in Table 1. Such a decomposition of the functionality of an SFC into AAs offers the following key advantages: **(1)** Since not all elements of an SFC are involved in processing all transaction types, the resource requirements of an AA can be significantly lower than that of the original SFC. AAs with lower resource footprints are more likely to be instantiated in close proximity as compared to the entire SFC. **(2)** AAs enable granular resource allocation by coupling resource allocation with traffic composition. The AFO can scale-out AAs handling a specific transaction type as the percentage of messages of that transaction type increases. This allows Contain-ed to react, in real time, to incoming traffic composition.

3. CONTAIN-ED IN ACTION

In this section, we illustrate how Contain-ed can be leveraged for deploying an IMS instance to increase the utilization of the underlying NFV infrastructure.

3.1 Project Clearwater: IMS in the Cloud

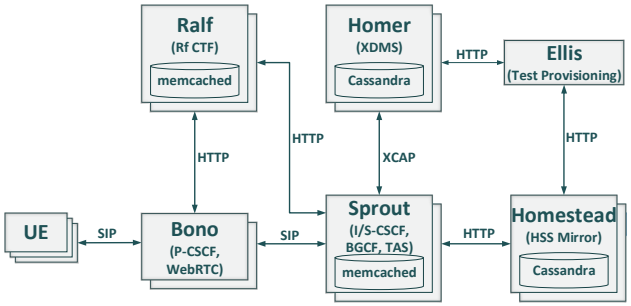

Figure 2: Clearwater Architecture

We choose Clearwater, an open-source IMS implementation. The availability of a containerized implementation of Clearwater enabled us to better compare performance of different IMS deployment options. While Clearwater provides a horizontally scalable clustered IMS implementation, the VNF components in Clearwater do not strictly match standard IMS functional elements. Clearwater utilizes web-optimized technologies like Cassandra and memcached to store long-lived state, provide redundancy, and eliminate the need for state replication during scale-in and scale-out.

The architecture of Clearwater is illustrated in Figure 2 (adapted from [1]). For brevity, only the components used in our experiments are depicted. We briefly explain the Clearwater components that can be deployed individually and horizontally scaled. **Bono** is the edge proxy component that implements the P-CSCF (Proxy Call Session Control Function) in the 3GPP IMS architecture [7]. SIP clients communicate with Bono over UDP/TCP connections and are anchored at a Bono instance for the lifetime of the registration. **Sprout** implements the Registrar, I/S-CSCF (Interrogating/Serving CSCF) and Application Server components. Sprout nodes store the client registration data and other session and event state in a memcached cluster. There are no long-lived associations between a user session and a Sprout instance. **Homestead** provides a REST interface

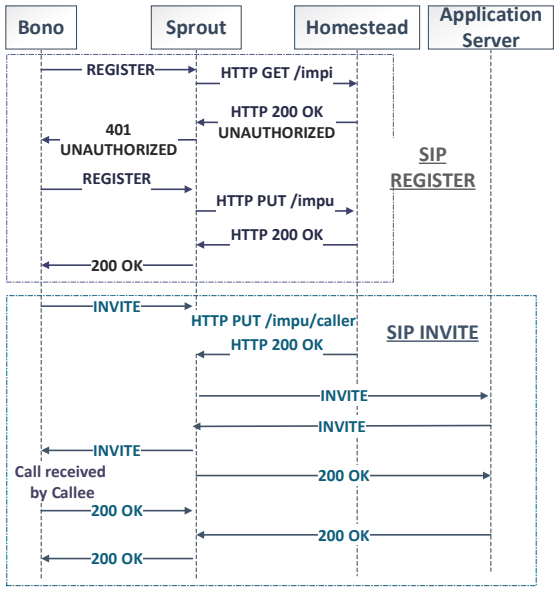

Figure 3: Clearwater IMS Call Flow

to Sprout for retrieving the authentication vectors and user profiles. Homestead can host this data locally or retrieve it from the HSS using the Diameter Cx interface. **Homer** acts an XML Document Management Server that stores the service profiles. **Ralf** implements the Off-line Charging Trigger Function (CTF). Bono and Sprout report chargeable events to Ralf. Figure 3 illustrates this call flow without a third-party REGISTER in the iFC.

3.2 Mapping with Contain-ed

We determine the AAs by applying the principles in Section 2: **(1) VNF Affinity**: Analyzing the 3GPP IMS architecture [7], we find that there is high affinity between the P-CSCF and S-CSCF components. Therefore, we can aggregate the Bono and Sprout nodes in Clearwater to create an AA. **(2) Transactional Atomicity**: We demonstrate the application of this principle by analyzing user registration in IMS, which generates two messages by the user device. Since both messages must be handled by the same instance of Bono and Sprout, we consider user registration as a transactional boundary.

We thus create an AA of type "REGISTER" corresponding to the SIP REGISTER call flow. A similar reasoning allows us to create AAs of types "SUBSCRIBE" and "INVITE" for the IMS user SUBSCRIBE/NOTIFY and INVITE call flows, respectively. The AAs for "REGISTER" and "SUBSCRIBE" consist of an instance of Bono and Sprout, while the AA for "INVITE" additionally contains an instance of Ralf due to the CTF interaction described in Table 1. We do not use an Application Server (AS) in our testing, so it is not included in the AAs.

In Clearwater, Bono and Sprout operate in a transaction-stateful manner. Transactions in the same SIP dialog can be handled by a different Sprout instance since the Sprout instances share long-lived user state using memcached. Clearwater therefore supports the transactional atomicity property of Contain-ed. Contain-ed leverages the Clearwater memcached as the shared state store.

We develop the AD component based on the OpenSIPS [2] dispatcher. In this implementation, the AD anchors all the incoming and outgoing calls from Clearwater and acts a stateless inbound proxy. It uses a hash on the message "Call-ID" to direct incoming request messages. This mechanism ensures that the messages for the same user session are directed to the same AA instance. For outgoing messages, the AD inserts appropriate SIP headers to ensure that messages take appropriate paths.

4. EXPERIMENTAL EVALUATION

We developed a prototype implementation of the Contained deployment component (the dark shaded box in Figure 1). We use the information in Table 1 to bundle the Clearwater components into AA types. The AAs are instantiated at startup to match the workload requirements (the AFO dynamic scaling/instantiation functionality is not yet implemented).

4.1 Experimental Setup

We use Docker version 17.03.0-ce and Docker-compose (v1.11.2) for micro-service container lifecycle management. The Clearwater VNF components run within a container on the same physical host. A private subnet created by Docker is used for communication between these containers, thereby minimizing the communication latency among the Clearwater VNF micro-services. The physical resources of the server are shared by all containers and there are no resource constraints on an individual container. The Contained AD component is deployed on the same physical machine as the Clearwater VNF. The AD runs on the physical machine directly, and therefore shares the resources with the Clearwater VNF components.

Workload generation: We use SIPp [4] as a workload generator. SIPp runs on a dedicated physical machine, and generates two types of requests: REGISTER and SUBSCRIBE. As shown in Figure 3, REGISTER requests are used to register the user device in the network and result in the generation of two messages (initial request and challenge response) from the user device. SUBSCRIBE requests are used to subscribe to the the state of a user already registered with Clearwater. A SUBSCRIBE request from the requesting client is followed by a NOTIFY request from the server to update the client with the user subscription status. We measure the number of failures by the observing the result code in the SIP response message. Per the SIP specification, for register, "200 OK" indicates success and "401 Unauthorized" is used to challenge. All other 3XX and 4XX codes are considered failures. We observe the error codes received by SIPp for each message type and use them to infer the number of failures.

We generate a workload of 300 requests/s to 1800 requests/s in steps of 300 requests/s, and measure the total number of failed calls for each workload type. As described earlier, aggressive retransmission of requests by the client or middleboxes can exacerbate performance problems, so we disable this to increase the overall throughput. In order to circumvent the impact of retransmissions on our experiments, we configure SIPp to not retransmit requests that failed due to timeouts. Each experiment runs for 60 seconds. The results presented below represent the mean of at least 10 samples for each call rate and delay value.

The performance of a complex VNF like Clearwater is impacted by the control interplay among its functional components. Previous studies [13] have revealed that disproportionate resource utilization by Clearwater components can influence system performance, and the overall throughput depends on the resources allocated to individual components. Clearwater employs token buckets and timeout-based peer blacklisting mechanisms for fault-tolerant overload control. This can also influence the overall throughput. Furthermore, individual components may timeout and discard incoming requests. As an example, Sprout uses a timer to wait for the response messages from Homestead, and if no response is received before a timeout, a failure response (response code timeout 408) is issued to the client. To minimize the impact of disproportionate resource utilization, we do not allocate dedicated resources to any container and all Clearwater components share the available system resources. However, overall performance is limited by the token bucket rate and timeout(s) at individual components, resource utilization notwithstanding.

4.2 Experimental Results

Our experiments are designed to investigate the impact of network latency on Clearwater, and to quantify the performance benefits of Contain-ed. We begin by benchmarking Clearwater in "ideal" conditions on our testbed. In this case, all communicating VNF components are instantiated on the same physical machine. A single instance of Clearwater is created and both REGISTER and SUBSCRIBE messages are handled by this instance. This setup is labeled "ideal" in our plots. We measure the performance of this setup with both REGISTER and SUBSCRIBE workloads.

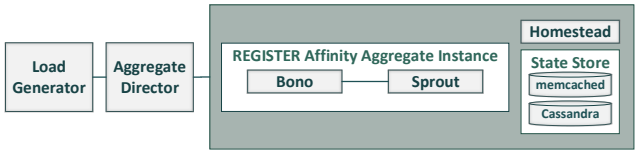

Figure 4: Contain-ed setup with REGISTER AA

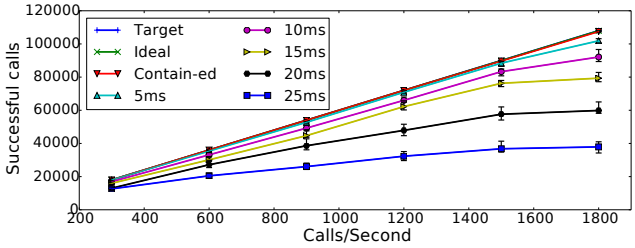

Figure 5: Successful REGISTER calls

We also measure the performance of Clearwater when the VNF components are not located on the same physical machine and therefore the communication latencies are higher than the ideal case. We simulate a scenario where the Sprout node is located in a different DC by adding delays on the Sprout-bound links. As described earlier, the SIP REGISTER request generates two register messages from Bono to Sprout and two database lookup requests from Sprout to Homestead, and therefore Sprout placement is vital to the performance of Clearwater. We use "tc" to introduce delays on the links from Bono to Sprout and Sprout to Homestead. We use delays of 5 ms, 10 ms, 15 ms, 20 ms and 25 ms and

compare the performance of this setup with the "ideal" case. Figures 5 and 7 present the results. In both figures, the error bars represent the minimum and maximum values observed among all samples for a data point. The label "Target" in the figures indicates the maximum number of calls that can be successfully processed at a given call rate.

As seen from Figures 5 and 7, increasing communication latency to a single Clearwater component (Sprout) can result in significant performance degradation. The impact of the introduced latency is not significant at low call rates. However, as the call rate reaches the system capacity, there is significant drop in system throughput. This is a consequence of the timeouts experienced at individual components. As load increases, the number of messages that are waiting for a response at each individual component becomes larger, and higher system capacity is utilized in sending timeout responses at each individual component.

We now describe our experimental setup using Contain-ed. Figure 4 shows an instantiation of Contain-ed to handle REGISTER messages. It consists of the REGISTER AA (Sprout and Bono), the shared state store, Homestead, and the AD. Figure 6 depicts the setup of Contain-ed for handling SUBSCRIBE. This setup consists of two AAs (REGISTER, SUBSCRIBE), since the users must be registered before SUBSCRIBE messages. Both the REGISTER and SUBSCRIBE AAs contain an instance of Bono and Sprout. All other VNF components like Homestead and the shared state store are shared by the AAs. For the SUBSCRIBE setup, all REGISTER messages are handled by the REGISTER AA, and SUBSCRIBE/NOTIFY messages are handled by the SUBSCRIBE AA. A single instance of AD is created in both the cases, which forwards the incoming traffic to the appropriate AA.

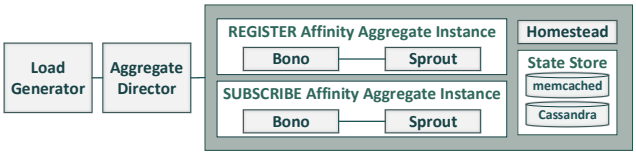

Figure 6: Contain-ed setup with REGISTER/SUBSCRIBE AAs

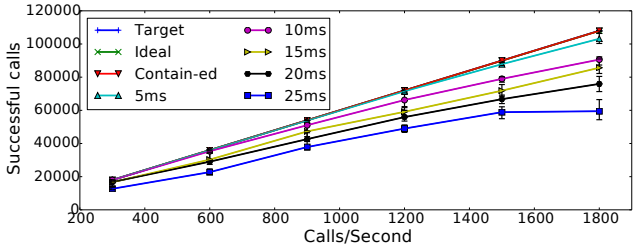

Figure 7: Successful SUBSCRIBE calls

Comparing the results of Contain-ed with "ideal" in Figures 5 and 7, we conclude that the AD does not result in significant call drop compared to the ideal setup, and the overhead due to the AD does not significantly impact overall performance. The DC setup with induced latency increasingly drops higher numbers of messages as the latency increases, but the Contain-ed setup continues to process messages without suffering from significant performance degradation. Even when multiple AA instances of different types are created, the performance impact of the AD and Contain-ed is minimal.

It is important to note that workloads react differently to increasing latency. This is due to the nature of communication between various components within the VNF. User actions that require memory lookup/update (authorization/billing events) will respond poorly to increased latency towards the memcached/cassandra components and workloads that require frequent communication with other components like SUBSCRIBE will respond poorly to increased latency towards state management components within the VNF. With traditional network placement, it is difficult to strike the right balance between workloads and their dependencies. In contrast, the Contain-ed setup can ensure co-location of VNF components for each workload type, and, as seen from the results above, will continue to process various workload types without suffering from significant performance degradation.

5. RELATED WORK

Placement problems and network latency have been widely studied in the context of NFV. A generally accepted direction for scalable cloud-based infrastructure is decoupling user state storage from VNF processing logic. Kablan et al [16] investigate a stateless design that leverages technologies like RAMCloud over InfiniBand to demonstrate how a NAT function can be decomposed into packet processing and data. The focus of their work is, however, on demonstrating how a stateless design improves the elasticity of NFV deployments. They do not consider the impact of stateless design principles on limiting e2e latency. Decoupled control and user plane network functions are also described in [5]. This work aims at minimizing communication latency between the control plane elements while simultaneously bringing the user data processing elements close to the network edge, improving the overall user experience.

Basta et al [12], Hawilo et al [15] and Katsalis et al [17] propose redesign of existing networks to reduce latency. There is, however, little work that uses container-driven backward-compatible solutions. Basta et al [12] explore several implementation models in which v-EPC can be deployed. However, the implementation models proposed can result in extensive refactoring of existing implementations. The scope of their work is limited to the placement of the user and control planes in EPC gateways, and does not include common principles that can be applied to any SFCs. Hawilo et al [15] discuss a scheme for bundling EPC components, guided by the principles of a flat architecture and the decoupling of the control and user planes. While this architecture proposes bundling an NF and provides an analysis of the benefits of the proposed architecture, it does not investigate how these changes can be implemented in current architectures. The work closest to ours was conducted by Katsalis et al [17]. This work analyzes a stateless 5G design pattern. They propose a micro service-driven stateless RAN architecture which uses shared control plane contexts for data storage. The work is limited to the analysis of RAN and does not delve into the application of this design to general SFCs.

6. DISCUSSION AND FUTURE WORK

[**Aggregate Director implementation**] The AD is a protocol-aware online module that classifies each incoming packet, and can consume significant resources. However, there are existing NEs in current telecommunication networks that inspect and classify incoming traffic, and the AD can leverage such gateway elements to minimize resource utilization. In IMSes, the P-CSCF component anchors all inbound signaling traffic. Similarly, the Diameter Routing Agent [10] processes incoming Diameter signaling messages in the EPC. Whenever possible, such existing elements can be enhanced to support the functionality of the AD.

[**State synchronization overhead**] Contain-ed uses persistent state information shared across AA instances to avoid pinning of requests to a particular AA instance. State synchronization overhead can be kept minimal by appropriately determining transaction boundaries and only pushing persistent state to the shared state store.

[**Flow control**] VNF implementers employ throttling based flow control between different VNFs to prevent instability. This is particularly important when capacities of different VNFs are not matched. Clearwater uses token buckets to gracefully manage overload. In Contain-ed, VNFs/VNF components within the same AA can be appropriately provisioned warranting overload detection and gating control only at the AA ingress. This can further improve resource utilization, as overhead can be eliminated from each participating VNF.

[**Legacy VNF implementations**] The Clearwater implementation we evaluated had several design features such as shared state store (SSS) for horizontal scaling. However, not all VNF implementations are amenable to direct application of Contain-ed. Additional stubs may be required for pushing state information to the SSS. We believe that adoption of container technologies and micro-service design principles will reduce the number of VNF implementations requiring significant modification.

[**Future work**] We need to better understand the performance impact of the functional components of Contain-ed, especially the aggregate director and the shared state store modules. Since the AD anchors all incoming traffic, it may become a single point of failure. Future versions of the Contain-ed framework will explore a distributed AD design to deploy a load balancing cluster with multiple instances. Considering the importance of clustering in Contain-ed, it is also vital to understand the performance impact of the SSS. The relative placement of the SSS w.r.t. to individual AAs dictates the number of AAs that can be simultaneously instantiated without violating the SLAs. Therefore, a better understanding of the performance overhead introduced due to the SSS is imperative in deciding the capacity and placement of the AAs. In addition, our future work will explore how to leverage Contain-ed for emerging network architectures such as the 3GPP Machine-Type Communications (MTC) networks and dedicated core networks in the EPC core.

7. REFERENCES

[1] Clearwater. http://www.projectclearwater.org/.

[2] *opensips*. http://www.opensips.org/.

[3] *SIP: Session Initiation Protocol*. https://tools.ietf.org/html/rfc3261.

[4] *SIPp*. http://sipp.sourceforge.net/.

[5] 3GPP. *TR 23.714, Study on control and user plane separation of EPC nodes*. http://www.3gpp.org/DynaReport/23714.htm.

[6] 3GPP. *TS 23.218, IP Multimedia (IM) session handling*. http://www.3gpp.org/DynaReport/23218.htm.

[7] 3GPP. *TS 23.228, IP Multimedia Subsystem (IMS)*. http://www.3gpp.org/DynaReport/23228.htm.

[8] 3GPP. *TS 23.335, User Data Convergence (UDC)*. http://www.3gpp.org/DynaReport/23335.htm.

[9] 3GPP. *TS 23.401, GPRS Enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) Access*. http://www.3gpp.org/ftp/Specs/html-info/23401.htm.

[10] 3GPP. *TS 29.213, PCC signalling flows and Quality of Service (QoS) parameter mapping*. http://www.3gpp.org/DynaReport/29213.htm.

[11] 3GPP. *TS 29.274, Evolved General Packet Radio Service (GPRS) Tunnelling Protocol for Control plane (GTPv2-C)*. http://www.3gpp.org/DynaReport/29274.htm.

[12] Basta, A., Kellerer, W., Hoffmann, M., Morper, H. J., and Hoffmann, K. Applying NFV and SDN to LTE mobile core gateways, the functions placement problem. In *Proceedings of the 4th Workshop on All Things Cellular: Operations, Applications, & Challenges* (2014), pp. 33–38.

[13] Cao, L., Sharma, P., Fahmy, S., and Saxena, V. NFV-VITAL: A framework for characterizing the performance of virtual network functions. In *IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)* (Nov 2015), pp. 93–99.

[14] Han, B., Gopalakrishnan, V., Ji, L., and Lee, S. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine 53*, 2 (Feb 2015), 90–97.

[15] Hawilo, H., Shami, A., Mirahmadi, M., and Asal, R. NFV: state of the art, challenges and implementation in next generation mobile networks (vepc). *CoRR abs/1409.4149* (2014).

[16] Kablan, M., Caldwell, B., Han, R., Jamjoom, H., and Keller, E. Stateless network functions. In *Proceedings of the 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization* (2015), pp. 49–54.

[17] Katsalis, K., Nikaein, N., Schiller, E., Favraud, R., and Braun, T. I. 5G architectural design patterns. In *2016 IEEE International Conference on Communications Workshops (ICC)* (May 2016), pp. 32–37.

[18] Pentikousis, K., Wang, Y., and Hu, W. Mobileflow: Toward software-defined mobile networks. *IEEE Communications Magazine 51*, 7 (July 2013), 44–53.

[19] Rajan, A. S., Gobriel, S., Maciocco, C., Ramia, K. B., Kapury, S., Singhy, A., Ermanz, J., Gopalakrishnanz, V., and Janaz, R. Understanding the bottlenecks in virtualizing cellular core network functions. In *The 21st IEEE International Workshop on Local and Metropolitan Area Networks* (April 2015), pp. 1–6.