Procedure-driven Deployment Support for the Microservice Era

Amit Sheoran
asheoran@alumni.purdue.edu

Sonia Fahmy
Purdue University
fahmy@purdue.edu

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

Navin Modi
modin@alumni.purdue.edu

ABSTRACT

In this work, we examine the challenges that service providers encounter in managing complex service function graphs, while controlling service delivery latency. Based on the lessons we learn, we outline the design of a new system, *Invenio*, that empowers providers to effectively place microservices without prior knowledge of service functionality. *Invenio* correlates user actions with the messages they trigger seen in network traces, and computes *procedural affinity* for communication among microservices for each user action. The procedural affinity values can then be used to make placement decisions to meet latency constraints of individual user actions. Preliminary experiments with the Clearwater IP Multimedia Subsystem demonstrate that even a single high-latency link can result in significant performance degradation, and placement with *Invenio* can increase user quality of experience.

CCS CONCEPTS

•Networks → Network management; *Data center networks; Mobile networks;*

KEYWORDS

Cloud-native architectures; Microservices; Cellular Networks

ACM Reference format:
Amit Sheoran, Sonia Fahmy, Puneet Sharma, and Navin Modi. 2021. Procedure-driven Deployment Support for the Microservice Era. In *Proceedings of 22nd International Middleware Conference Industrial Track, Virtual Event, Canada, December 6–10, 2021 (Middleware '21 Industrial Track),* 7 pages.
DOI: 10.1145/3491084.3491429

1 INTRODUCTION

The recent trends of increasing *virtualization* and *cloudification* have reduced the capital and operational costs for service providers. Network Functions Virtualization (NFV) enables the deployment of virtualized instances of Network Functions (NFs) on demand [14]. New service offerings can be created by adding one or more NFs to a function graph, also known as Service Function Chain (SFC), *i.e.*, a graph of NFs. Software architectures have evolved to support the pace of service deployment, and disaggregated fine-grained microservice designs are replacing monolithic designs [11, 32].

Complexity introduced by composition and cloudification. Rapidly adding and deleting new services incurs a cost. Function graphs are becoming more complex, and the effort associated with service deployment is growing [15, 30, 37, 38]. To reduce costs, service providers are increasingly using private or public clouds to deploy services that had traditionally been confined to a single data center and had used carefully-designed proprietary hardware. This *cloudification* trend poses unique challenges to orchestration frameworks, particularly in instantiating and placing Virtualized Network Functions (VNFs) in a function graph with strict Service Level Agreements (SLAs) [38].

Network functions in systems such as the cellular Evolved Packet Core (EPC) and IP Multimedia Subsystems (IMS) react poorly to unpredictable latency variation [26, 28, 29, 34]. Fortunately, service providers can use their domain knowledge of NF functionality, and meticulously define function graphs [1, 2] to bound the latency. Virtualization platforms such as Openstack [23], Kubernetes [18], and Docker [10] allow administrators to configure "affinity policies" in placement. The affinity policies specify which NFs should be colocated to meet SLA requirements. However, the increasing use of non-standard interfaces and the ongoing integration of 5G core (5GC) [3] into existing 4G network deployments is necessitating extensive manual re-analysis of communication patterns. The diversity of NFs in modern networks and the new 5GC interfaces make determining the function graphs involved in service delivery (and the communication affinities between their constituent NFs) a time-consuming and error-prone task.

Complexity introduced by microservices. Microservices are fine-grained, independent components that can be deployed as autonomous entities communicating via REST-based proprietary interfaces [7, 24]. This results in disaggregation and decomposition of a VNF into multiple smaller VNF Components (VNFCs), and more complex function graphs [11, 38]. The lack of standardization in microservice architectures yields VNFCs that play roles that do not accurately map to an NF defined by standards. This ambiguity in the role of VNFCs/microservices implies that placement using domain knowledge is insufficient, and we need automated tools to infer communication patterns between components to aid service providers.

Our approach. We use information exposed by NFs or microservices to optimize placement and meet SLAs [15, 30, 37]. However, merely co-locating NFs based on the number of messages they exchange [30] can yield unexpected results due to the diversity of workloads. Instead, we propose grouping messages triggered by a user action into *procedures*, and computing *procedural affinity* between NFs. A provider can then make placement decisions based on procedural affinity values, together with procedure type distribution and policies. For example, a VNFC used during voice calls, but not for SMS (text-msg), in a cellular network can be placed based on the currently dominant workload type and its requirements.

Contributions. This paper describes the *Invenio* system for supporting NF deployment. *Invenio* maps user activity at the network edge to traffic in the network core, computes procedural affinity, and aids in making placement decisions. *Invenio* includes two subsystems that are executed after upgrades or policy and service changes: one in which a snapshot of traffic is analyzed to compute affinity values, and another in which an orchestrator uses computed procedural affinity values, in conjunction with current procedure type distribution and policy rules, to make placement decisions. This paper primarily focuses on first subsystem, *i.e.*, automatically computing procedural affinity values. *Invenio* empowers providers to optimize placement to meet SLA objectives, even with upgrades in services and microservices and changing user demands. In summary,

(1) We identify the challenges for a service provider to meet SLAs (§3).
(2) We describe *Invenio*, a system for service providers to automatically compute procedural affinity (§4).
(3) We demonstrate the benefits of placement based on affinity by studying the performance of voice-call and text-msg workloads (§5).

We believe that the principles underlying *Invenio* are applicable to the service-based architecture of the 5GC and other microservice-based deployments.

2 DEPLOYING MICROSERVICES

A network function can be instantiated on bare metal (as a Physical Network Function (PNF)) or on virtualized hardware (as a VNF). A VNF can be deployed as a collection of VNFCs or microservices. In the rest of this paper, we use the term NF to refer to all three types of instantiations (PNF/VNF/VNFC).

The increasing use of private or public clouds to reduce operational costs has yielded scenarios where NFs in a function graph are deployed across multiple physical machines in one or more data centers. Consider Fig. 1 which shows an example microservice-based cellular network for Voice over LTE (VoLTE) that includes wireless access, session management, voice-call signaling, policy control (QoS), and billing. Latency-sensitive NFs (such as signaling and policy) may be connected by high and unpredictable latency links.

2.1 Impact of Service Type

An orchestrator that cannot instantiate the entire function graph in Fig. 1 on a single machine or rack can identify the NFs exchanging a large number of messages and place them in close proximity. Modern networks offer many services, however, and NFs exchange different types and numbers of messages to support each service.

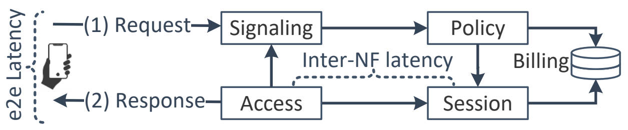


Figure 1: Latencies in a cellular network

Not all services have equal impact on user-perceived latency and quality of experience (QoE). For example, interactive services such as voice calls impact user QoE more than non-interactive services such as text-msg or presence services [33]. Orchestrators must reduce the end-to-end latency of interactive services by minimizing the inter-NF latency for NFs handling these services. Simple techniques such as counting total messages exchanged between NFs [30] are not always effective in making placement decisions as they do not explicitly consider the impact of inter-NF latency on user QoE.

Table 1 shows the percentage of traffic exchanged by NF pairs for two different procedure type distributions of voice-call (V), text-msg (T), and presence (P) services in a VoLTE

Table 1: Messages exchanged between NF pairs in a microservice-based VoLTE implementation

NF Pair	% of Traffic with Service Type Distribution	
	V(5%),T(5%),P(90%)	V(48%),T(52%)
AF, SUB	27.4%	0%
SUB, memcached	27.4%	0%
GX-App, memcached	9%	20%
AF, PCRF-Base	9%	20%
PCEF, PCRF-Base	9%	20%
PCRF-Base, GX-App	9%	20%
PCRF-Base, RX-App	6%	13.5%
RX-App, SDP	3%	6.4%

system we implemented. The left column uses proportions from typical busy-hour IMS traffic [4] in which presence is triggered ~9x more frequently than voice. Clearly, exchanged messages depend on the incoming procedure type distribution and therefore merely using the number of messages for placement [30] may optimize non-interactive services such as presence and degrade user QoE.

To meet SLAs for latency-sensitive services, service providers may (a) create dedicated NFs to optimize specific functionality [28], or (b) decompose existing monolithic applications into lightweight microservice components, that are then aggregated by functionality to create NF bundles, and placed together with a higher probability [34]. Manually identifying and configuring bundles can be difficult and error-prone, however.

2.2 Key Insight

Our goal is to empower service providers to easily and automatically react to upgrades and changing user QoE demands. From prior research [26, 28, 29, 34], we observe that: (a) NFs typically exchange several messages to complete a seemingly simple user action such as turning on User Equipment (UE) or making a voice call, and (b) Network endpoints only perceive latency in the actions they trigger (*i.e.*, end-to-end latency in Fig. 1), and are oblivious to message exchanges and inter-NF latency within a function graph. User QoE therefore only depends on user action/network response pairs, such as initiating a voice call (action) and hearing a dial tone (network response), or turning on an Internet connection (action) and being connected to a packet access network such as LTE (network response). Based on this insight, we aim to leverage readily available knowledge of *endpoint actions* to improve NF placement.

3 GROUPING PROCEDURE MESSAGES

We need to group events or messages triggered due to a single user action into *procedures.* We then use this procedure information to compute *procedural affinity* between NFs for each procedure type. The procedural affinity information is then used for NF placement.

Since NFs in modern networks exchange numerous messages, manually determining control messages that are triggered due to a specific endpoint (or associated user or subscriber) action can be tedious and error-prone. We propose to (a) automatically isolate control messages related to a user, and (b) map each message to an action invoked by that user. We describe the challenges in accomplishing these tasks in the remainder of this section.

3.1 Scale and Complexity

It is necessary to understand the protocols and message formats exchanged by each NF. For example, consider a cellular network EPC (including NFs to inter-work with previous generation networks (2G, 3G) and WiFi). Such an EPC deployment can involve 60+ NFs communicating via 15+ protocols over 150+ interfaces using 500+ message types [2, 9]. While many of these NFs are logical, the sheer number of NFs, supported protocols, and message types makes isolating and understanding control-plane traffic a difficult task.

3.2 User and Session Identification

Networks such as cellular networks check user (subscriber) identifiers located in control messages to determine the user associated with a device or a network endpoint, and NFs use these identifiers to enforce policies and bill users. A user is identified by: (a) **Subscriber-ID:** the key used by the network to authenticate a device, identify packets associated with it, and bill the user, and (b) **Session-ID:** the key allocated by an NF to group together messages triggered by a device. Unlike the subscriber-ID, the value of session-ID is not pre-allocated, *i.e.,* NFs allocate a value at run-time. Different protocols and interfaces use different terms to refer to the subscriber-ID and session-ID carrying headers.

Since the session-ID is dynamically allocated, the relation between session-ID and subscriber-ID may vary based on the NFs involved in message processing. When a single device creates multiple connections at the same time (such as in EPC), multiple session-IDs may be allocated to the same subscriber-ID. Additionally, an EPC/IMS may create a mapping between the session-ID and the subscriber-ID, and then use the two values interchangeably.

As in the 4G core, 5GC NFs [3] use headers such as Subscription Permanent Identifier (SUPI) and Subscription Concealed Identifier (SUCI) to identify, authorize, and bill traffic. A user can create multiple sessions with 5GC data networks and therefore the 5GC NFs use session headers such as the pudSessionId in conjunction with the user ID to uniquely identify user sessions. When 4G EPC and the 5GC coexist, the complexity of manual NF placement further increases.

3.3 Proprietary Microservices

Microservice architectures use fine-grained autonomous components, fragmenting traditional control-plane NFs into multiple VNFCs [15, 37, 38]. The VNFCs are independently instantiated, and communicate using proprietary message formats. This lack of standardization implies that the roles and functionalities of VNFCs are not well-understood and can change with new versions, altering their affinity. Consequently, service providers must (re)analyze affinity whenever NFs are upgraded or a service is added/removed. Microservices also result in more complex function graphs, reducing the latency allowed for each VNFC [11, 38].

While the lack of standardization can complicate mapping a given message to a user action, microservices often reuse the subscriber-ID/session-ID in traditional signaling protocols [20, 25] to facilitate logging and reduce performance overhead. For example, the timer service (Chronos) in Clearwater [25], a popular microservice-based IMS implementation, uses the "Call-ID" header in Session Initiation Protocol (SIP) messages to manage timers. This behavior can be exploited to trace VNFC-generated messages to user actions.

3.4 Lessons Learned

The above discussion highlights three consequences for *Invenio*. First, *Invenio* should automate message and event processing, which should be transformed into a protocol-agnostic format before further processing. Second, *Invenio* should understand the relation between different identifiers used by NFs to correlate messages related to the same user. This involves understanding the user-identifying headers used by standard protocols, and correlation of identifiers in proprietary message payloads. Third, *Invenio* should understand user actions and their corresponding responses, and map each message to a specific user action. Since internal implementations of microservice-based systems change frequently, *Invenio* should only use endpoint messages which follow well-known protocols (such as messages (1) and (2) in Fig. 1) to map messages to user actions.

4 SYSTEM PROTOTYPE

Fig. 2 shows the architecture of our prototype. *Invenio* consists of two components: an affinity engine, executed after upgrades, and a placement engine, executed when a new NF is to be instantiated or after major changes in policies or procedure type distribution.

Invenio uses messages exchanged between NFs in a function graph to identify procedures and their associated messages. *Invenio* utilizes output generated by open-source packet analyzer software such as Wireshark [36] to decode

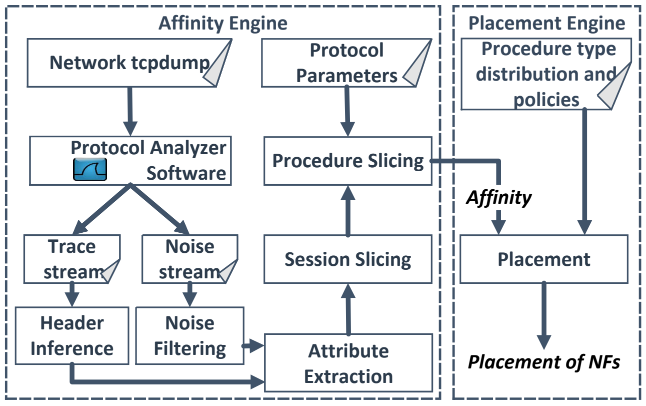


Figure 2: *Invenio* architecture

raw messages. We use Wireshark to export Packet Description Markup Language (PDML) and Portal Structure Markup Language (PSML) files, and use these files as inputs. *Invenio* also uses domain knowledge of the service provider, specified in a configuration file containing the following information in xml format: (a) Procedure start/end messages, and (b) Subscriber-ID and session-ID header names. Finally, *Invenio* uses procedure type distribution and provider policy information to decide NF placement.

The first step in computing affinity is to find user-identifying headers. A header inference module analyzes messages in the trace stream to identify possible headers that carry the subscriber/session-ID values. Then, the noise filtering module eliminates messages not generated due to user actions. Since protocols and interfaces use different encoding formats (binary or text) to exchange information, we convert the input message stream to a protocol-agnostic intermediate format that we refer to as *event objects*. The session slicing module operates on the event objects generated by the attribute extraction module to identify all messages associated with a single user. The procedure slicing module determines the NF set and message set associated with each procedure type. Finally, the placement engine uses affinity information generated by the affinity engine, together with input procedure type distribution and provider policies, to make the final NF placement decisions.

Our *Invenio* prototype includes ~2600 lines of Python code.

5 PRELIMINARY RESULTS

In this section, we seek to answer two questions:

(1) How effective is *Invenio* in computing affinity with multiple protocols?
(2) What is the impact of inter-NF latency on performance under different workloads?

Table 2: Testbed configuration

Server	CPU	Cores	RAM	NFs Deployed
R430	2x Intel Xeon E5-2620 v4	16	64 GB	Clearwater
DL120	1x Intel Xeon X3430	4	8 GB	Swarm Workers, Load-Generator

5.1 Clearwater Architecture

We use Clearwater [25], an open-source platform for a containerized, microservice-based, implementation of an IMS. Clearwater uses REST-based communication to retrieve authentication vectors, manage timers and handle state synchronization, which makes it ideal for a case study. The architecture is illustrated in Fig. 3 (adapted from [25]). Only the components used in our experiments are depicted. We use Clearwater version 1.0 (clearwater-docker release-120). **Bono** an edge proxy that implements the P-CSCF (Proxy Call Session Control Function (CSCF)) in the 3GPP IMS architecture [1]. SIP clients communicate with Bono over UDP/TCP connections. **Sprout** implements the Registrar, I/S-CSCF (Interrogating/Serving CSCF) and Application Server components. **Homestead** provides a REST interface to Sprout for retrieving authentication vectors and user profiles. **Chronos** is a distributed, redundant, reliable timer service. Bono and Sprout report chargeable events to the Charging Trigger Function **Ralf**.

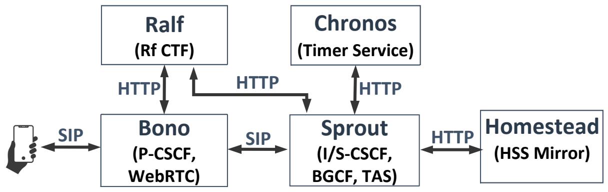


Figure 3: Clearwater architecture

5.2 Experimental Testbed

Our testbed includes one Dell PowerEdge R430 and 5 HP ProLiant DL120 G6 (Table 2) connected by a Gigabit Dell N2024 Switch. We use Docker [10] version 17.03.0-ce and Docker-compose (v1.11.2) to deploy NFs for Clearwater (Fig. 3). Each NF runs within a container and all containers are deployed on the same physical host.

5.3 Experimental Methodology

We use two primary network services: (a) **Voice-call**: This service involves two procedure types (INVITE and BYE). (b) **Short Message Service (text-msg)**: This service utilizes a single procedure of type MESSAGE. We also use SUBSCRIBE (which supports the Presence service) to illustrate the impact of message-count based placement on system performance. However, SUBSCRIBE messages are not generated during performance evaluation and system performance is only evaluated for interactive workloads (voice-call and text-msg).

SIPp [35] is used to generate four types of messages: REGISTER, INVITE, BYE and MESSAGE. Each SIPp instance runs on a dedicated physical machine and saturates available system resources. We measure failures by the observing the result code in the SIP response messages. We record the total number of successful calls or messages for each workload type. For the voice-call workload, where multiple procedure types are required to complete a call, we only count the number of calls that were successfully completed; *i.e.*, partially completed calls are ignored. Each experiment runs for 30 seconds. Each experiment is repeated at least 5 times and results are shown with 95% confidence intervals.

Table 3: NF affinity for Clearwater

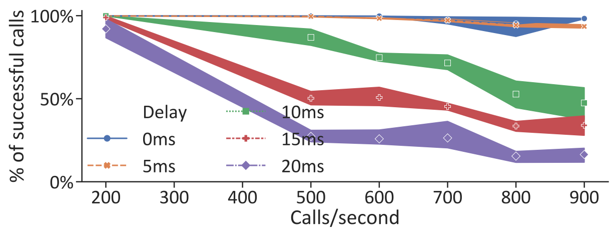
Procedure Type	NF Pair	Affinity
NFs: Bono, Sprout, Ralf		
Voice-call	Bono, Sprout	10
	Bono, Ralf	8
	Sprout, Ralf	4
Text-msg	Bono, Sprout	4
	Bono, Ralf	2
	Sprout, Ralf	0

5.4 Affinity Analysis

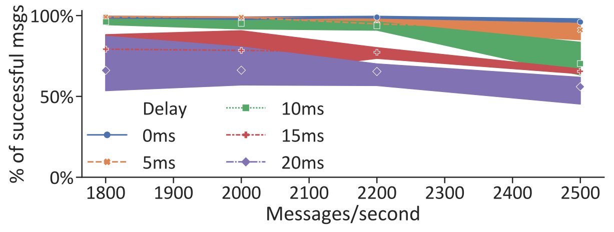
We use *Invenio* to compute affinity between NFs in Clearwater for both voice-call and text-msg workloads. The results are presented in Table 3. We observe that the affinity between Clearwater NFs is different for voice-call and text-msg traffic. For instance, for voice-call traffic, there is high affinity between Bono, Sprout and Ralf, whereas for text-msg traffic, Bono and Ralf only exchange two messages, and no messages are exchanged between Sprout and Ralf. Ralf therefore has a higher affinity with Bono and Sprout for voice-call workload compared to text-msg workload. The placement of Ralf w.r.t. to Bono and Sprout thus has a higher impact on the performance of voice calls compared to the text-msg workload.

5.5 Preliminary Performance Results

We first benchmark the performance of the voice-call and text-msg workloads with negligible delay. These results serve as baselines, and are labeled "ideal" in our plots. We then use `tc` to introduce latency on links connecting two NF pairs (a) Ralf to Sprout and (b) Ralf to Bono, to validate the impact of placement of Ralf on performance. We experiment with delays of 5 ms, 10 ms, 15 ms, and 20 ms and compare to the "ideal" case. Fig. 4a and 4b present the results of



(a) Voice-call performance



(b) Text-msg performance

Figure 4: Impact of latency and affinity on Clearwater

voice-call and text-msg workloads, respectively. We make the following observations from the figures: (a) Even a single high-latency link can result in significant performance degradation for both voice-call and text-msg workloads, and (b) Performance degradation for the voice-call workload (in which Ralf has higher affinity) is more than for the text-msg workload (in which Ralf has lower affinity). These observations underscore the need for careful VNFC placement. While manual analysis shows that Sprout and Bono (which collectively implement the functionality of the CSCF) must always be co-located, analysis of IMS standards does not suffice for proprietary IMS implementations such as Clearwater in which internal implementation determines the affinity values between VNFCs (Bono/Sprout and Ralf).

6 RELATED WORK

We discuss work related to service placement and monitoring, then work that reverse-engineers or infers protocol behavior.

Service placement and monitoring. Functionality-based decomposition has been proposed to reduce latency and increase throughput for cellular network control planes [16, 26, 28, 34]. That body of work uses manual analysis of network architecture and traffic to find the functional elements that can be aggregated. Stratos [12] avoids traversing oversubscribed inter-rack links during function placement, and Selimi et al. [31] explore placement to maximize bandwidth utilization. None of these studies consider workload types and procedures. Other work [15, 30, 37] formulates the placement problem as a graph partitioning problem or an optimization problem. This is orthogonal to our work, as our notion of procedural affinity is a new factor to consider as an input to the placement problem.

Recent studies [7, 21, 22, 24, 38] have highlighted challenges in integrating, deploying, and managing microservice-based applications. For instance, ucheck [24] uses runtime verification and enforcement of invariants to ease managing microservice-based applications. Probius [22] locates performance bottlenecks by correlating VNF, hypervisor, and system metrics. NFVPerf [21] uses network traces collected from NFs to compute per-hop message processing latency and infer performance bottlenecks. Our work proposes procedure-driven microservice deployment, and is thus complementary to this line of work.

Protocol inference. Extracting protocol state information from network traces, or *reverse engineering* a protocol, has been widely studied in the literature. Prior work extracts specifications of unknown protocols [5, 6, 8, 17], and uses inferred message formats to detect malware signatures [19]. *Invenio* also exploits protocol header information, but utilizes protocol analyzer tools to extract user-identifying headers.

Other categories of work in this area use xml or json formats exported by tools such as wireshark to derive protocol state machines [13, 27]. While *Invenio* shares finite-state machine extraction techniques with these papers, it differs in one important aspect: we use the extracted state information to compute affinity between NFs for an entire function graph. In contrast, prior work extracts the state machine for a single NF and does not merge state machines from multiple NFs to derive procedure information for a function graph.

7 CONCLUSIONS

This paper gives a brief overview of a system that enables service providers to better manage the ever-growing complexity of microservice-based network functions. We showed that we can automatically compute communication affinity values for each user-triggered procedure. Our work empowers service providers to make and update placement decisions without the time-consuming and error-prone manual analysis currently used. Our preliminary experiments with the Clearwater IP Multimedia Subsystem are promising. We expect the importance of this work to grow as more complex disaggregated services are deployed.

ACKNOWLEDGMENTS

This work was performed while Amit Sheoran and Navin Modi were with Purdue University, and sponsored in part by NSF grant CNS-1717493. The work was started at and initially supported by HPE. The authors would like to thank Vinay Saxena and Lianjie Cao (HPE) for their valuable input.

REFERENCES

[1] 3GPP. *TS 23.228, IP Multimedia Subsystem (IMS).* http://www.3gpp.org/DynaReport/23228.htm.

[2] 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.

[3] 3GPP. *TS 23.501, System Architecture for the 5G System.* http://www.3gpp.org/ftp/Specs/html-info/23501.htm.

[4] Abhayawardhana, V. S., and Babbage, R. A traffic model for the IP multimedia subsystem (IMS). In *2007 IEEE 65th Vehicular Technology Conference - VTC2007-Spring* (2007), pp. 783–787.

[5] Antunes, J., Neves, N., and Verissimo, P. Reverse engineering of protocols from network traces. In *2011 18th Working Conference on Reverse Engineering* (Oct 2011), pp. 169–178.

[6] Bossert, G., Guihéry, F., and Hiet, G. Towards automated protocol reverse engineering using semantic information. In *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security* (New York, NY, USA, 2014), ASIA CCS '14, ACM, pp. 51–62.

[7] Cerny, T., Donahoo, M. J., and Trnka, M. Contextual understanding of microservice architecture: Current and future directions. *SIGAPP Appl. Comput. Rev. 17*, 4 (Jan. 2018), 29–45.

[8] Cui, W., Kannan, J., and Wang, H. J. Discoverer: Automatic protocol reverse engineering from network traces. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium* (Berkeley, CA, USA, 2007), SS'07, USENIX Association, pp. 14:1–14:14.

[9] Dialogic. *IMS and VoLTE 3GPP Interfaces.* https://edit.dialogic.com/edu/interfaces.

[10] Docker. *Build, Ship, and Run Any App, Anywhere.* https://www.docker.com.

[11] Gan, Y., Zhang, Y., Cheng, D., Shetty, A., Rathi, P., Katarki, N., Bruno, A., Hu, J., Ritchken, B., Jackson, B., Hu, K., Pancholi, M., Clancy, B., Colen, C., Wen, F., Leung, C., Wang, S., Zaruvinsky, L., Espinosa, M., He, Y., and Delimitrou, C. An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud and Edge Systems. In *Proc. of ASPLOS* (April 2019).

[12] Gember, A., Grandl, R., Anand, A., Benson, T., and Akella, A. Stratos: Virtual middleboxes as first-class entities. Tech. rep., University of Wisconsin-Madison, 2012. Technical report TR1771.

[13] Griffeth, N., Cantor, Y., and Djouvas, C. Testing a network by inferring representative state machines from network traces. In *Software Engineering Advances, International Conference on* (Oct 2006), pp. 31–31.

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

[15] Hu, Y., de Laat, C., and Zhao, Z. Optimizing service placement for microservice architecture in clouds. *Applied Sciences 9*, 21 (2019).

[16] 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.

[17] Krueger, T., Krämer, N., and Rieck, K. ASAP: Automatic semantics-aware analysis of network payloads. In *Proceedings of the International ECML/PKDD Conference on Privacy and Security Issues in Data Mining and Machine Learning* (Berlin, Heidelberg, 2011), PSDML'10, Springer-Verlag, pp. 50–63.

[18] Kubernetes. *Kubernetes.* https://kubernetes.io/.

[19] Leita, C., Mermoud, K., and Dacier, M. Scriptgen: an automated script generation tool for honeyd. In *21st Annual Computer Security Applications Conference (ACSAC'05)* (Dec 2005), pp. 12 pp.–214.

[20] Microsoft. *Designing microservices: Logging and monitoring*, 2018. https://docs.microsoft.com/en-us/azure/architecture/microservices/logging-monitoring.

[21] Naik, P., Shaw, D. K., and Vutukuru, M. NFVPerf: Online performance monitoring and bottleneck detection for NFV. In *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)* (Nov 2016), pp. 154–160.

[22] Nam, J., Seo, J., and Shin, S. Probius: Automated approach for VNF and service chain analysis in software-defined NFV. In *Proceedings of the Symposium on SDN Research* (New York, NY, USA, 2018), SOSR '18, ACM, pp. 14:1–14:13.

[23] OpenStack. *Open source software for creating private and public clouds.* https://www.openstack.org/.

[24] Panda, A., Sagiv, M., and Shenker, S. Verification in the age of microservices. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems* (New York, NY, USA, 2017), HotOS '17, ACM, pp. 30–36.

[25] Project Clearwater. *IMS in cloud.* http://www.projectclearwater.org/.

[26] Qazi, Z. A., Walls, M., Panda, A., Sekar, V., Ratnasamy, S., and Shenker, S. A high performance packet core for next generation cellular networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (New York, NY, USA, 2017), SIGCOMM '17, ACM, pp. 348–361.

[27] Rafique, M. Z., Caballero, J., Huygens, C., and Joosen, W. Network dialog minimization and network dialog diffing: Two novel primitives for network security applications. In *Proceedings of the 30th Annual Computer Security Applications Conference* (New York, NY, USA, 2014), ACSAC '14, ACM, pp. 166–175.

[28] Raza, M. T., Kim, D., Kim, K. H., Lu, S., and Gerla, M. Rethinking LTE network functions virtualization. In *2017 IEEE 25th International Conference on Network Protocols (ICNP)* (Oct 2017), pp. 1–10.

[29] Raza, M. T., and Lu, S. Enabling low latency and high reliability for IMS-NFV. In *2017 13th International Conference on Network and Service Management (CNSM)* (Nov 2017), pp. 1–9.

[30] Sampaio, A. R., Rubin, J., Beschastnikh, I., and Rosa, N. S. Improving microservice-based applications with runtime placement adaptation. *Journal of Internet Services and Applications 10* (2019).

[31] Selimi, M., Cerdà-Alabern, L., Sánchez-Artigas, M., Freitag, F., and Veiga, L. Practical service placement approach for microservices architecture. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)* (2017), pp. 401–410.

[32] Sharma, S., Miller, R., and Francini, A. A cloud-native approach to 5G network slicing. *IEEE Communications Magazine 55*, 8 (2017), 120–127.

[33] Sheoran, A., Fahmy, S., Osinski, M., Peng, C., Ribeiro, B., and Wang, J. Experience: Towards automated customer issue resolution in cellular networks. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking* (2020), MobiCom '20.

[34] Sheoran, A., Sharma, P., Fahmy, S., and Saxena, V. Contain-ed: An NFV micro-service system for containing e2e latency. In *Proceedings of the Workshop on Hot Topics in Container Networking and Networked Systems* (2017), HotConNet '17, pp. 12–17.

[35] SIPp. *Welcome to SIPp.* http://sipp.sourceforge.net/.

[36] Wireshark. *Wireshark Go Deep.* https://www.wireshark.org.

[37] Yu, Y., Yang, J., Guo, C., Zheng, H., and He, J. Joint optimization of service request routing and instance placement in the microservice system. *Journal of Network and Computer Applications 147* (2019), 102441.

[38] Zhang, Y., Hua, W., Zhou, Z., Suh, G. E., and Delimitrou, C. Sinan: ML-based and QoS-aware resource management for cloud microservices. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (2021), ASPLOS 2021, p. 167–181.