IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 12 September 2022, accepted 26 September 2022, date of publication 3 October 2022,
date of current version 10 October 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3211293

TOPICAL REVIEW

A Retrospective on Workload Identifiers: From Data Center to Cloud-Native Networks

ANDREW BABAKIAN[1,2], PERE MONCLUS[1], ROBIN BRAUN[2], (Life Senior Member, IEEE), AND JUSTIN LIPMAN[2], (Senior Member, IEEE)

[1]VMware Inc., Palo Alto, CA 94304, USA
[2]School of Electrical and Data Engineering, University of Technology Sydney, Ultimo, NSW 2007, Australia

Corresponding author: Andrew Babakian (ababakian@vmware.com)

ABSTRACT As applications move to multiple clouds, the network has become a reactive element to support cloud consumption and application needs. Through each generation of network architectures, identifiers and the use of dynamic locators evolved in different levels of the protocol stack. The identifiers and locators type is defined by the isolation boundary and how the architecture considers semantic overload in the IP address. Each solution is an outcome of incrementalism, resulting in application delivery outgrowing the underlying network. This paper contributes an industrial retrospective of how the schemes and mechanisms for identification and location of network entities have evolved in traditional data centers and how they match cloud-native application requirements. Specifically, there is an evaluation of each application artifact that forced necessary changes in the identifiers and locators. Finally, the common themes are highlighted from observations to determine the investigation areas that may play an essential role in the future of cloud-native networking.

INDEX TERMS Workload identifiers, network locators, naming and addressing, data center, cloud-native.

I. INTRODUCTION

In today's dynamic environment, the network is under pressure to deliver connectivity and identity across multiple clouds in an agile manner. The architectural support for identifiers and locator split has solidified their role as an essential function to keep pace with application growth and flexible cloud computing models.

From a network access-control perspective, IP addresses could be viewed as identity labels for authorization in a network zone model and, in parallel, are routing instructions for networked elements to guide a packet to the destination. In each generation of network architectures, there is a realization that the IP space cannot serve two purposes of identity and location in virtualization and cloud computing environments. A technique to overcome the semantic overload of IP addresses was to split the IP space for persistent identity and dynamic location through encapsulation protocols as a point of indirection. However, as the new generation of cloud applications emerges, the IP space has become weak and mutable to maintain its role as a persistent identifier. For example, microservice applications heavily depend on network proxies, known as sidecars, to provide application availability and resiliency. In such network rendezvous architectures, the IP address has no lasting semantics [28]. An observation is made that the cloud-native architectures have shifted workload identifiers and created new namespaces to support service discovery, routing, and identification for applications to extend administrative boundaries.

This paper contributes to a discussion of the schemes and techniques for identifying and locating network entities that have evolved in traditional data centers and how they match cloud-native application requirements. The distinction this paper makes is that it surfaces how each application artifact forced a shift in workload identifiers required for discovery, routing, and identification, which is not so obvious in past papers from a comparison point of view. For example, the rationale behind identifiers, locators, and insertion

The associate editor coordinating the review of this manuscript and approving it for publication was Nitin Gupta.

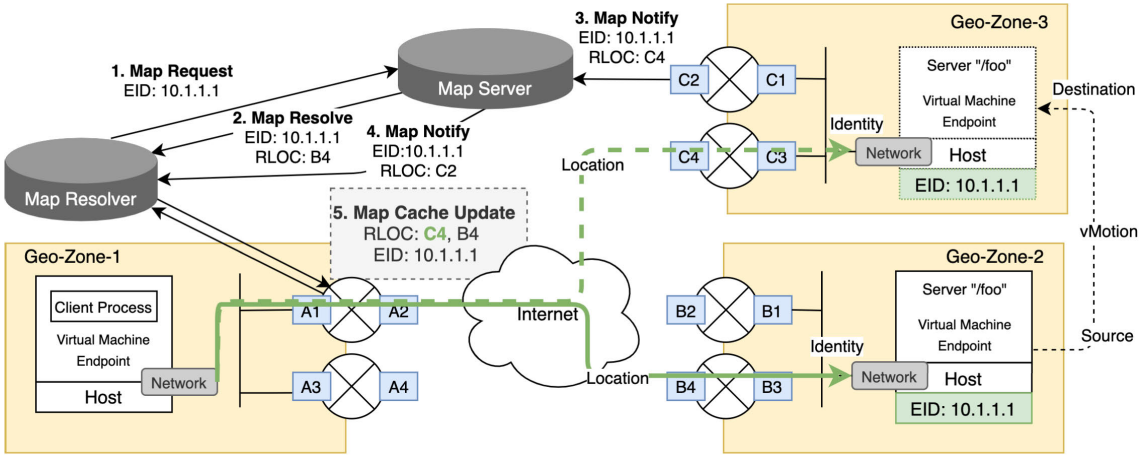

FIGURE 1. LISP Endpoint Identifier and Dynamic Locator.

points required for today's cloud-native networks is emphasized rather than focusing on identity and location split in its generality.

In addition, this paper contributes an increased awareness of evolved cloud application artifacts that lead to the functionality required for Service Mesh and its impact on the use of location-independent identifiers.

Finally, this paper closes by considering the discussed techniques, state-of-the-art opportunities, and directions for future investigation into the critical elements foundational to a cloud-native network.

II. LOCATION/ID SEPARATION PROTOCOL

A. SEMANTIC OVERLOAD IN THE IP ADDRESS

As background, the Internet Architecture Board (IAB) workshop [43], the concerns centered around the current routing information base (RIB) size of the border gateway protocol (BGP) [50]. One of the underlying causes of routing scalability was identifier/locator overload in the IP address. Enterprises were advertising provider-independent prefixes across sites to avoid server renumbering. From a firewall point of view, the server's IP address was its identity; therefore, IP renumbering would be an operational burden.

With the current IP architecture, it seemed impossible to have a single IP space that could serve two purposes, and a split seemed necessary to scale the routing system. This conclusion was the supporting material that led to the creation of the locator/ID separation protocol (LISP) [18]. There are several other proposals to separate identifiers and locators, such as HIP [45], Mobile IP [48] and RINA [29]. To get started, this section selected LISP that addressed the workload mobility challenges traditional in data centers.

The initial technique of LISP was to provide topological aggregation, leveraging a locator approach for the IP routing system. The network's edge contains more specific details needed to reach the endpoint. Provider-independent networks resided behind LISP networks and behaved similarly to network address translation (NAT) [17]. However, in contrast to NAT, the translation state in LISP was not held in a network device but stored in the original IP packet. In LISP, the original IP packet is encapsulated with an outer IP header that contains the locator information.

B. LISP FOR VIRTUALIZED DATA CENTERS AND THE VIRTUAL MACHINE ARTIFACT

While the initial motivation of LISP was to solve a route scalability issue, it became a tool to address mobility challenges created by virtualized data centers (VDCs) [4]. VDCs refer primarily to a hosted virtualization platform, where servers are decoupled from their underlying hardware resource. This disaggregation allowed the infrastructure layer to isolate memory, CPU, and storage to each virtual machine (VM) instance managed by the hypervisor [62]. Applications within the VMs artifact remained unaware of this abstraction. As VDCs became mainstream, VM portability between physical hosts became an appealing function for disaster recovery, operations maintenance, and a method to utilize capacity in active-active data centers (DC).

C. IDENTITY AND THE IP NAMESPACE

LISP naturally addressed the VDCs needs by allowing VM migration between data centers without burdening the routing system. The insertion point of LISP was at the edge of the data center core network, where the endpoint identifiers (EIDs), in this case, VMs, are mapped to routing locations (RLOCs) at the geozone level. The VMs are not globally routable and are topologically independent from the underlying network. For example, in Figure 1 VM "/foo" migrates from Geo-Zone-2 to Geo-Zone-3 without disrupting the existing transport-layer session from a client in Geo-Zone-1. Through LISP control-plane notifications, Geo-Zone-1 receives a Map-Notify to inform that "/foo's" IP address of "10.1.1.1"has migrated to Geo-Zone-3. The

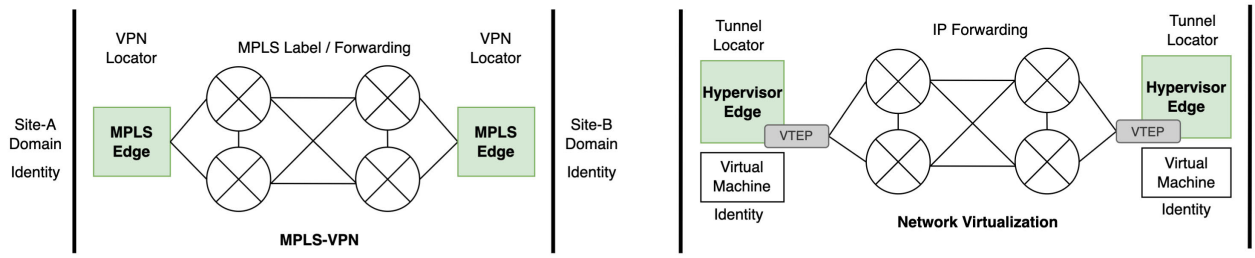

FIGURE 2. Core versus Edge Split: The Newly Revised Edge.

LISP-speaking router "A1" updates "10.1.1.1" entry to map to "C4" in Geo-Zone-3. This map and encapsulation with a dynamic approach allow the VM's identity namespace to remain preserved during the migration procedure.

In summary, LISP introduced a disjoint IP namespace, where IP is separated as an identity (i.e., EID) attribute and IP as used for location. This separated namespace technique in the data center network architecture allowed VMs to preserve their identity while taking advantage of the virtualization operational model, which proved to minimize service disruption based on its mobility usecase. LISP's mapping system introduced a new approach to providing network control at the edge of the data center. The arguments for separation of edge vs. core become an important approach for the IP routing system [30], and for providing a network insertion point to support ephemeral and mobile data center workloads.

III. NETWORK VIRTUALIZATION

A. CORE VS. EDGE SPLIT

In the section above, LISP's identifier mappings are only relevant at the edge (i.e., VMs), while the core of the network is forwarding IP packets to the correct Geo-Zone (i.e., location). This architecture is similar to Multi-protocol Label Switching (MPLS) [54]. In MPLS-VPN [53], the provider core (P) is solely focused on forwarding packets based on MPLS labels, while the provider edge (PE) has specific details of the neighbor domains (i.e., identifier). This separation allowed the PE to create multiple virtualized paths, which cater to overlapping address spaces over the same physical network.

B. NETWORK VIRTUALIZATION FOR CLOUD

In the quest for a purpose-built network for a cloud computing architecture, network virtualization (NV) [36] replicated the same separation of core and edge as demonstrated in the LISP and MPLS design and solved new challenges that cloud computing created.

In the technique of Network Virtualization (NV), the network becomes a reactive element to the application and cloud needs, promulgating the hypervisor vSwitch [49] as the newly revised edge, depicted in Figure 2. This is also known as the virtual network edge (NVE) [37]. Furthermore, cloud computing forced the network architecture to create a unifying abstraction across hypervisors. In contrast to LISP and MPLS-VPN, NV introduced the dynamic creation of virtual networks, each with independent address spaces and topologies that coexist over the same physical network.

In cloud computing, self-service consumption, rapid VM provisioning, and lifecycle management are subsumed in the NV architecture. The NVE allowed tenants to dynamically create network topologies to support the virtualized applications' or VM's lifecycle state. The virtual switch (vSwitch) within the hypervisor becomes the first network control point at the edge. In an NV model, the vSwitches are programmed by a logically centralized control plane responsible for synchronizing the forwarding state across all NVEs. The controller cluster is responsible for the overall state management, including the VM location. In addition, the control plane disseminates the VM identifiers to locator mappings efficiently using an advertisement-based model.

C. IDENTITY AND THE IP NAMESPACE

In the context of NVEs, vSwitches are responsible for providing connections over IP tunnels as paths without modifications to the physical network. While there are many variations of overlays [12], [24], the industry primarily adopted the Virtual eXtensible Local Area Network (VXLAN) [40] encapsulation protocol as the de facto standard in the data center and cloud. VXLAN embeds tenant-specific identifiers for isolation to ensure packets are distinguished between tenants that cohabit on a common platform. VXLAN preserves the original IP space as the VM's identity. The encapsulation contains an outer IP space to be used as a dynamic locator known as virtual-tunnel endpoints (VTEPs), which is a routable interface bound to the hypervisor.

In summary, NV adopted an application first principle, unlike LISP, where the NVE dynamically reacts to the consumption needs of self-service clouds. Rapid VM provisioning and management of lifecycle state requires the control plane to disseminate identifiers and locator mappings to hypervisors on a need-to-know basis. Like LISP, NV created a disjoint IPv4 namespace, separating the role of identifier and locator as independent addresses. Due to the prevalence of internal network-based middleware such as firewalls, the IP address must maintain its role as the core identifier for the VM.

IV. IDENTIFIER LOCATOR ADDRESSING

A. CONTAINER ARTIFACT AND THE ENTERPRISE CLOUD

There has been a great increase in the adoption of delivering applications in the format of containers [3]. It is appealing for cloud-native applications to be delivered in container-based virtulization [61] due to their efficient resource utilization and portability between cloud platforms. There are two new challenges that containers create for networking. Firstly, every container process requires a single IPv4 address, which leads to exhaustion in CIDR allocation due to the scale of processes compared to virtual machines. Secondly, the container's runtime layer is above the hypervisor, requiring a new networking control point.

B. IDENTIFIER LOCATOR ADDRESSING (ILA) NAMESPACE

To address these challenges, identifier locator addressing (ILA) [26] was proposed. Similar to NV described in the previous section, ILA is also an overlay network focused on separating core and edge networks. However, ILA differs from NV in two ways. ILA focuses on the layer above server virtualization, to which NV becomes transparent. In addition, ILA is an IPv6 overlay solution that takes advantage of a larger address space to modify the identifier, thus removing encapsulation overhead that VXLAN and LISP introduced. Its primary motivation is to support container virtualization use cases such as scheduling, task migration, and container orchestration [34], while preserving container identity.

The fundamental principles of ILA were borrowed from earlier work in the identifier-locator network protocol (ILNP) [2] and GSE (8+8) [46], which also addressed the challenges of overloaded semantics in the IP address. The ILNP approach demonstrated a way of using addressing by creating a distinction between the core routing and end systems.

While ILNP had the design properties to fit the dynamic behavior of cloud workloads, the implementation required modifications to the TCP stack to perform dynamic changing of locator bindings. ILA addressed this challenge by implementing an IPv6 [13] transformation technique that ensures the transport layer remains unmodified with an immutable 64-bit identifier and SIR prefix. There is an allocation of a /64 prefix per host representing the locator within the ILA scheme. As described in Figure 3, when a client/server session is initiated between two containers, the ILA host will perform an outside-of-process stateless IP address transformation on the higher-order 64 bits, the locator, to forward packets to the container's destination. On the receiving side, the ILA host rewrites the destination IP address from the locator prefix to the original identifier address and subsequently delivers the packet to the container application.

The control plane of this architecture disseminates identifiers and locator mappings to all ILA host routing elements. The ILA control plane is a distributed key-value store that manages mappings of identifiers to locators in a network. As tasks or containers are registered, the mappings are published to the control plane and disseminated to all ILA hosts through its push-based model, which differs from LISP pull-based model.

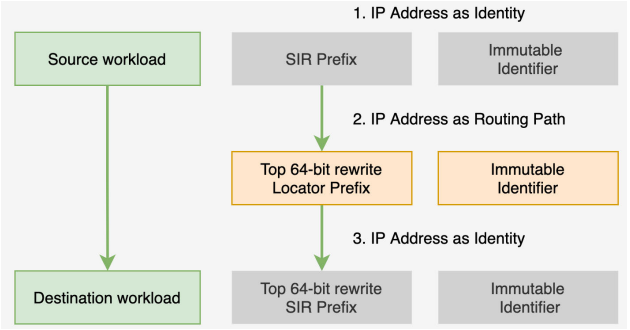

FIGURE 3. IPv6: Identifier and Locator Transformation.

In summary, while the ILA solution applies creative techniques through IPv6 transformations, the administrative boundaries of this architecture assume the control of the host stacks to allow an insertion strategy to preserve the original container identifiers and modify the locator instructions within the packet. In addition, it will continue to become more challenging to preserve the end-to-end identity that IPv6 advocates across heterogeneous cloud providers as NAT, Loadbalancers, and network-based middle-ware will force identifiers higher in the protocol stack.

V. CLOUD-NATIVE SERVICE MESH

A. MICROSERVICES AND SERVICE MESH

Previous sections discussed the networking challenges in cloud computing models that deployed N-Tier applications in VM or container artifacts. This section discusses the network requirements for a new generation of cloud-native applications called microservices [15]. A microservice is a collection of networked services, with each service performing a specific business function. Services are built independently from each other and delivered in the format of containers. The value is that developers can alter and upgrade the application without a full redeployment.

A single cloud-native application can be represented by thousands of services that are continually changing states. The probability of communication failure between services increases as cloud-native applications grows exponentially. Application resiliency leads to the need for a dedicated infrastructure layer handling the service-to-service communication. Service mesh [39], [41] addresses this challenge by injecting a proxy service, known as a sidecar, distributed within an application or microservice boundary. The service mesh ensures an optimal application-level response time by handling requests, retries, and timeouts, such as circuit breaking, load balancing, and traffic engineering. Each application request is routed optimally and is load-balanced without the knowledge of the application. The following sections describe the refinements of identifiers and locators, considering that an application residence is in multiple cloud platforms.

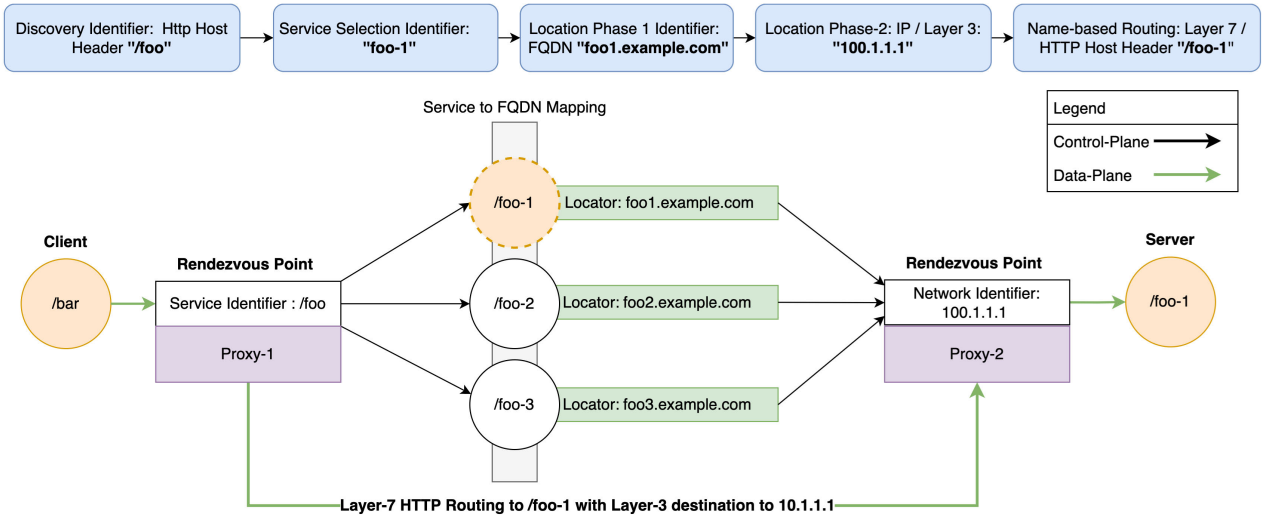

FIGURE 4. Cloud-Native Service Mesh: The Application-level Identifier for Name-based Routing.

B. SERVICE MESH NAMESPACE FOR DISCOVERY AND ROUTING

The service mesh is responsible for discovering, locating, and routing between proxies to establish application connections, as depicted in Figure 4. In this example, the client ''/bar'' service requests to communicate with the server ''/foo'' service. Firstly, the client sends an HTTP [20] request to query for foo, modifying the host header field to ''Host: foo''. The service name ''/foo'' is an anycast identifier [7] mapped to multiple instances or replicas of the service. The service mesh has a selection criterion to map the service identifier to a locator. In this example, the query for ''/foo'' is based on the proximity and latency between the client and server, and therefore ''foo-1'' is matched against the Fully Qualified Domain Name (FQDN) ''foo1.example.com ''. The resolution of this FQDN is IP address ''100.1.1.1'', which is the proxy rendezvous point. Furthermore, proxy-1 sets up a TCP connection with Proxy-2. Proxy-1 modifies the HTTP host header field to ''Host: foo-1'' to connect with the foo-1 service. In this example, the HTTP host header is a namespace that shifted the unique identifier from Layer-3 to Layer-7. The namespace to distinguish services is not limited to HTTP. With HTTP/S [51], the host header is encrypted, and the server name indicator (SNI), which is an extension of the TLS protocol [16] provides the namespace to distinguish services.

These name-based techniques were borrowed from earlier work in virtual-host networking [33], where web servers were consolidated to a common platform, overloading a single IP address with multiple service aliases [38]. A client would replicate the name previously learned from the Domain Name System (DNS) [44] into the HTTP application namespace, e.g., Host header field, to provide an application demuxer beyond the network layers. The HTTP/S protocol became the vehicle for carrying names and has been widely known as a name-based routing approach [35]. Content Delivery Networks (CDNs) also adopted the name-based virtual hosting to disentangle the conventional bindings between IP addresses, the applications they represent, and the servers to which they are assigned to [19].

C. SERVICE MESH IDENTITY AND THE URI NAMESPACE

Regarding identity, It is important to note that there are three distinct layers: the platform layer, the host layer, and the process/service layer. IP addresses have been the common identifier for modeling host-to-host communication. However, the scenarios where applications span multiple clouds present three challenges to model identity at the host layer:

- Workloads are ephemeral in nature, and the IP address is short-lived as an identifier.
- Workloads are deployed outside traditional administrative boundaries, such as the public cloud, which does not intersect with the internal enterprise network.
- Given that the service mesh proxies TCP connections and the network is riddled with network-based middleware boxes, the IP address becomes a weak and mutable identifier.

The objective of workload identity is to model service-to-service communication without preserving the IP address as the identifier. Applications that span administrative domains and cloud realms require identifiers to be uncoupled from the underlying infrastructure.

For cloud-native applications, the secure production identity framework for everyone (SPIFFE) addresses the workload identity challenge [58] using its naming schema independent of the discovery and routing described in this section. There are three components of SPIFFE: a specification and identifier used as a referral for a service (SPIFFE-ID), a SPIFFE Verifiable Identity Document (SVID) for embedding the SPIFFE-ID that is signed by a trusted authority, and the Workload API, which is an agent running within the

cloud that provides a method of obtaining the SVID. The SPIFFE-ID is a uniform resource identifier (URI) [5] that includes the scheme ''spiffe://,'' a trust domain, and a path that specifies the name of the service. For example, the foo service is a billing system whose administrative domain is in the production environment. The identifier can be represented as the following:

spiffe://production.example.com/billing/foo

The SPIFFE ID is codified into an SVID, which is an extension of the x509 certificate [11], signed by an authority inside the trust domain. Within the specification, the SPIFFE ID is set to the URI type in the subject alternative name extension (SAN) [11]. Thus, each workload will have a SPIFFE bundle used for path validation against all the SVIDS that belongs to the trust domain. SVIDs allow clients and servers to gain mutual knowledge by asserting their provable cryptographic identity for mutual TLS authentication [14], [23].

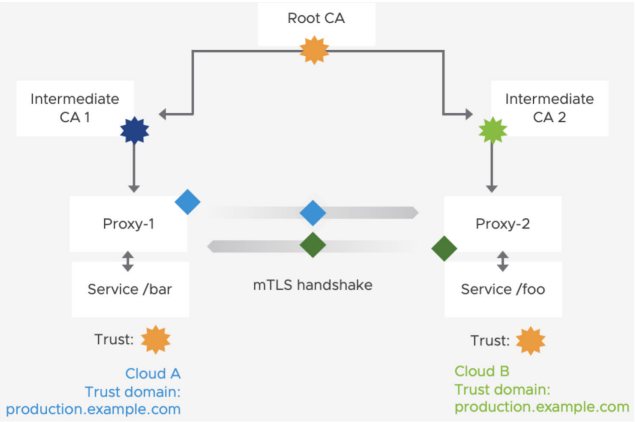

FIGURE 5. PKI: Mutual Authentication within a Trust Domain.

As depicted in Figure 5 an X509 certificate is issued to services ''/bar'' and ''/foo''. Both services are part of the trust domain of ''production.example.com' and have a common root of trust in the PKI hierarchy [9]. Both services will undergo the verification process in this scenario and exchange their x509 certificate as their identity document for mutual TLS authentication. The trust domain within the SVID identifier allows the administrative domain the flexibility to expand outside its platform and across multiple clouds where control of identity at the IP level is harder to achieve.

In summary, in the cloud-native service mesh, IP addresses are no longer identifiers for applications, and the distinguishing factors are names. Names do not represent a host, VM, or container; they refer to services inside an application. Services are assigned a persistent name as the identifier with a flexible administrative scope, spanning multiple cloud environments and organizations. Therefore, it is essential to delineate naming systems' for identifiers and locators in the cloud-native service mesh. First, the service name is derived from the HTTP header for the discovery and locating phase, which provides a mapping to a concrete FQDN. The URI within the SVID provides the verification phase for digital identity.

VI. COMPARISONS AND FUTURE DIRECTIONS

This paper discussed how the iteration of each network architecture addressed a new set of challenges that were non-existent in the previous generation. The paper includes an observation that the application artifact and cloud consumption model forced changes in identifiers and locators at various layers of the protocol stack. Firstly, this section will summarize how the core vs. edge evolved based on the network control required to meet the cloud-native needs. After that, the paper discusses future directions and provides topical areas that may play an essential role in future cloud-native networking.

A. COMPARISON BETWEEN DISCUSSED TECHNIQUES

Table 1 summarizes the association between the generation of the application artifact, identity controller, identifier namespace, and the network architecture insertion point. LISP and NV were the generation architectures that intersected with server virtualization. The insertion point in both approaches assumes control of the network platform and therefore takes a bottom-up approach to identifiers and locators anchored to the IP layer. In contrast to LISP, NV shifted the edge to the hypervisor, projecting an application-first architecture supporting the dynamic characteristics of enterprise cloud computing. ILA adopted similar techniques to NV by adding a layer of indirection with IPv6 to preserve the container identifier. However, the ILA edge model shifted to a layer above NV by inserting control at the container OS network namespace level.

The public cloud created a new challenge not present in previous approaches. LISP, NV, and ILA are approaches where the insertion point can assert identity, independent of location, as long as there is network control of the platform. However, as the public cloud infrastructure layers are hidden from consumers, it negates the ability of the network-centric approaches to provide a persistent identity at the IP layer using overlay techniques. A new generation of applications is designed to be less dependent on the underlying platform. The artifact comprises ancillary services known as a sidecar, which provides reachability and resiliency. The cloud-native service mesh addressed this challenge by creating an insertion point as distributed proxies to be the new edge. The abstractions for identity evolved to be an attribute of the application. Therefore it did not serve the purpose of structuring the IP address plan for identity and location as demonstrated in predecessor architectures. The use of naming as a point of indirection demonstrated that network flows could be distinguished in the context of the application-level session.

SPIFFE was separated into its own category with the cloud-native workload identities framework. Many application instances could represent a service in a plurality of cloud platforms. SPIFFE addresses this challenge by disseminating identifiers in the form of the URI to genuine instances of a

TABLE 1. Supporting Network Architectures, Namespace, Idenitifers and Insertion Points.

Application Artifact	Identity Controller	Identity Namespace	Distinguished Identifier	Insertion Point
Single-Tier: Virtual Machine	Locator/Identifier Seperation Protocol	Disjoint IPv4	IPv4 Address	Data Center Core Network
Multi-Tier: Virtual Machines	Network Virtualization	Disjoint IPv4	IPv4 Address	Hypervisor Network
Multi-Tier: Containers	Identifier Locator Addressing	Separated IPv6	IPv6 Address	Container OS - Network Namespace
Microservice: Containers	Cloud-Native Service Mesh	HTTP/S Host Header	Service & Canonical Name	Container Sidecar Proxy
Microservice: Hyperscale	SPIFFE	X509	URI	PKI Certificate Issuance

service. The x509 document underpins the SPIFFE solution; therefore, cloud-native workloads must have a common root of trust to authenticate mutually. If two services are under different trust hierarchies, then a third-party broker is required to exchange trust bundles between authority domains for mutual authentication.

Based on the observations of the cloud-native network requirements, there are four distinct roles of naming used as identifiers:

1) **Service.** A public-like persona to discover and identify services needed to compose an application.
2) **Locator.** The core or canonicalized name and the locator identifier to resolve the IP address of the service instance.
3) **Routing.** An application-level identifier replicated in the application protocol namespace to distinguish network sessions over a common IP address target.
4) **Identification.** Verifiable identifier to provide assurance it is a genuine copy or application instance.

Using these identifiers is only possible with the strict governance of naming inside an isolation boundary. Therefore, a namespace is required to ensure the scope of discovery, identifying, and locating workloads is within the trust domain.

B. COMPARISONS WITH THE STATE-OF-THE-ART

In recent years, developers have shifted their focus to constructing and deploying microservices without considering the underlying infrastructure requirements [52]. Serverless computing is an example of this paradigm that allows developers to focus on the business logic of the application without the consideration of provisioning and scaling of the entire system. Serverless functions are seen as event-driven, which triggers a temporary execution environment only when required. Amazon Web Services (AWS) Lambda, Microsoft Azure Functions, Google Functions, and Cloudflare Workers have introduced serverless capabilities, known as Function-as-a-Service (FaaS) [8].

In the event the serverless function is called, a virtualized workload is provisioned with application files and dependencies required to execute the function and terminates after the operations are performed [25]. In most implementations, containers are used as the primary artifact for their lightweight, application process isolation, and resource provisioning [21], [31]. Although serverless was originally enabled for cloud environments, it gained traction to cement its position in edge computing [59] where computational resources are closer to the data source [1]. However, as the edge environments consist of resource-constrained devices and diverse CPU architectures, WebAssembly (Wasm) [55] has emerged as a promising artifact for future serverless support at the cloud-edge [21], [22], [25], [42].

The serverless model has created new challenges in cloud-native networking that were not present in previous sections of this paper. In comparison, cloud-native service mesh provides end-to-end network control through distributed proxies. Each proxy enacts as an ambassador for each service-to-service communication, using the above four identifiers. Unlike service mesh, serverless computing offered by cloud providers does not expose the naming and addressing of the infrastructure platform [32], which thwarts the ability to bind the identifiers and locators for network control.

Irrespective of the serverless artifact type, the cloud providers only expose a publicly accessible Universal Resource Locator (URL) [6] identifier, which is used in the HTTP protocol as an access mechanism to a serverless function. The URL namespace describes the path instructions to a location of the available service. For clarity, serverless applications do not reside in a single location but are replicated globally; therefore, the same URL is also required to access multiple instantiations or replicas.

In Figure 6, the domain name portion of the URL is referred to as anycast identifier [7], which is persistent and resolves to multiple locators or IP addresses. The domain name to IP address translation is coordinated by the cloud-providers DNS-based global traffic management system (GTM) [27]. The DNS-Based GTM enacts as a geographic site-selector

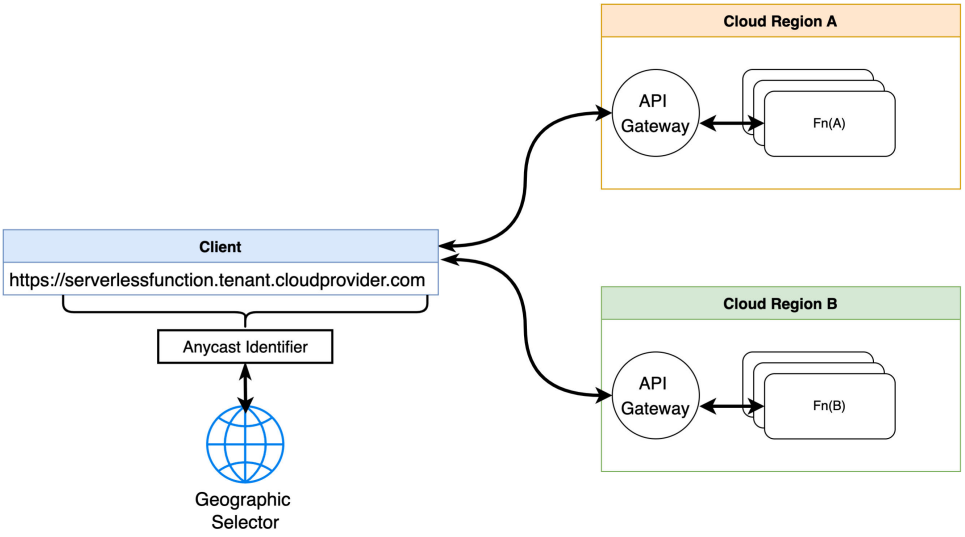

FIGURE 6. Persistent Identifier for Serverless in Hyperscaler Environments.

directing client connections to the closest serverless function [56]. This allows the URL to remain immutable, allowing the DNS to be the identity-like and public persona, returning customized answers based on the information from the client queries [10], [57].

C. FUTURE DIRECTIONS AND INVESTIGATIONS

A preview of future directions using Internet identifiers to support cloud-native services is appearing. Globally decentralized cloud-native applications rely on URLs to mimic and behave like URIs; To refer to the service independent of its location. This leads to three investigative areas to provide identifiers and the underlying infrastructure support that will play an essential role in cloud-native networks.

1) PERSISTENT IDENTIFIER

In the digital library space, the Digital Object Identifier (DOI) [47] is a persistent identifier that refers to a book or document residing in multiple locations at the same time. Analogous to a library, a cloud-native application also contains instances simultaneously running in multiple environments and locations. However, the complexity is that cloud-native applications today require four separate identifiers to be referred to and verified as the given instance. Cloud-native architectures need to adopt similar principles from the digital libraries; all application service instances should refer to as a single identifier. What an application is should be separated from where it is.

Furthermore, an investigation is also required to provide referential consistency that anchors application instances to a common identification scheme. This would comprise a directory service where the identifier could be passed between parties to provide locator mappings to cloud resources. In addition, the identifier is decorated with additional metadata such that the identifier would remain persistent. Unlike copies of books, applications evolve; therefore, providing additional metadata to adorn the query is a possible technique to obviate the need to create a new identifier.

2) IDENTITY RENDEZVOUS

The identity rendezvous provides two functions. Firstly, identity attestation to analyze credentials from all workloads to assure they are qualified application replicas or instances. Secondly, to connect and mutually authenticate between source and destination service. This approach is only possible when there is control over the workload deployment pattern, e.g., container workload and its ancillary service. When microservices are also constructed with serverless functions, it exposes new identity challenges that require the following two investigation areas. Firstly, as described in this section, cloud providers hide the interconnections of a serverless environment. It is possible to provide identity rendezvous as a sidecar or proxy as long as it is outside the providers' realm. The challenge it may create is additional network latency costs, which are worth exploring. Secondly, the execution of a serverless function has a temporary runtime environment which creates performance vs. security pressure to minimize application startup delays. In the context of the SPIFFE example, the framework assumes control of the underlying operating system of where the application is running. Since the serverless environment does not provide access, it is unclear how the attestation process will be completed to ensure the application is a genuine instance and the true owner of the identifier it passes before mutual authentication. This is use-case is not limited to serverless but future applications where only the API is exposed without access to the underlying infrastructure.

3) DISSEMINATION OF IDENTITIES

In order to construct complex applications between network and cloud realms, isolation boundaries can span multiple clouds and be created on-demand. This ensures that

application instances within an isolation boundary are uniquely rooted in a common trust model. If the industry wishes to continue using a SPIFFE-like model, where the certificate chain of trust underpins the identity framework, the challenge arises when applications are not within a common isolation boundary. While federation techniques are well understood in the industry with various approaches to exchanging bundles [60], the architecture lacks a third-party identity broker to manage and distribute trust bundles for general use and on a need-to-know basis.

VII. CONCLUSION

This paper highlighted how each generation of network architectures addressed a challenge created by evolved application artifacts. It highlights that cloud-native applications have shifted identifiers from the IP layer and have become an attribute of the application layer used to distinguish workloads and provide location-independent identification schemes. In addition, this paper reviewed the current state-of-the-art application deployment patterns and the future identifier challenges that may arise. This is an important topic that requires further investigation into three topical areas. Firstly, evaluate existing persistent identifiers in the digital space that could be applied for broader use. Regardless of location, application instantiations should be referred to as the single identifier. Secondly, the verification process, where credentials are analyzed to provide identity attestation and issuance of its imprimatur, should be consistent across any cloud and edge computing platform. Lastly, the delivery method to control and disseminate trust anchors to allow application communication between trust realms will underpin the success of a common identity system. It is hoped this retrospective of identifiers may influence the future design of identifiers and locators used in cloud-native architectures.

REFERENCES

[1] M. S. Aslanpour, A. N. Toosi, C. Cicconetti, B. Javadi, P. Sbarski, D. Taibi, M. Assuncao, S. S. Gill, R. Gaire, and S. Dustdar, "Serverless edge computing: Vision and challenges," in *Proc. Australas. Comput. Sci. Week Multiconference*, Feb. 2021, pp. 1–10.

[2] R. J. Atkinson and S. N. N Bhatti. *Identifier-Locator Network Protocol (ILNP) Architectural Description*, document RFC 6740, RFC Editor, Nov. 2012.

[3] G. Banga, P. Druschel, and J. C. Mogul, "Resource containers: A new facility for resource management in server systems," in *Proc. OSDI*, vol. 99, 1999, pp. 45–58.

[4] M. F. Bari, R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, and M. G. Rabbani, "Data center network virtualization: A survey," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 2, pp. 909–928, 2nd Quart., 2013.

[5] T. Berners-Lee, R. T. Fielding, and L. Masinter. *Uniform Resource Identifiers (URI): Generic Syntax*, document RFC 2396, RFC Editor, Aug. 1998. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2396.txt

[6] T. Berners-Lee, L. Masinter, and M. McCahill. *Uniform Resource Locators (URL)*, document RFC 1738, RFC Editor, Dec. 1994. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1738.txt

[7] B. Carpenter and S. Brim.*Middleboxes: Taxonomy and Issues*, document RFC 3234, RFC Editor, Feb. 2002. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3234.txt

[8] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, "The rise of serverless computing," *Commun. ACM*, vol. 62, no. 12, pp. 44–54, Nov. 2019.

[9] S. Chokhani *et al. Internet X. 509 Public Key Infrastructure Certificate Policy and Certification Practices Framework*, document RFC 3647, RFC Editor, Nov. 2003.

[10] C. Contavalli *et al. Client Subnet in DNS Queries*, document RFC 7871, RFC Editor, May 2016.

[11] D. Cooper *et al. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, document RFC 5280, RFC Editor, May 2008. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5280.txt

[12] B. Davie and J. Gross. *A Stateless Transport Tunneling Protocol for Network Virtualization (STT)*. document Internet-Draft draft-davie-stt-08, IETF Secretariat, Apr. 2016.

[13] S. E. Deering and R. M. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*, document RFC 2460, RFC Editor, Dec. 1998. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2460.txt

[14] T. Dierks and C. Allen. *The TLS Protocol Version 1.0*, document RFC 2246, RFC Editor, Jan. 1999. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2246.txt

[15] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: Yesterday, today, and tomorrow," *Present and Ulterior Software Engineering*. Cham, Switzerland: Springer, 2017, pp. 195–216.

[16] D. Eastlake. *Transport Layer Security (TLS) Extensions: Extension Definitions*, document RFC 6066, RFC Editor, Jan. 2011. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6066.txt

[17] K. B. Egevang and P. Francis. *The IP Network Address Translator (NAT)*, document RFC 1631, RFC Editor, May 1994. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1631.txt

[18] D. Farinacci *et al. The Locator/ID Separation Protocol (LISP)*, document RFC 6830, RFC Editor, Jan. 2013. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6830.txt

[19] M. Fayed, L. Bauer, V. Giotsas, S. Kerola, M. Majkowski, P. Odintsov, J. Sitnicki, T. Chung, D. Levin, A. Mislove, C. A. Wood, and N. Sullivan, "The ties that un-bind: Decoupling IP from web services and sockets for robust addressing agility at CDN-scale," in *Proc. ACM SIGCOMM Conf.*, Aug. 2021, pp. 433–446.

[20] R. T. Fielding *et al. Hypertext Transfer Protocol—HTTP/1.1*, document RFC 2616, RFC Editor, Jun. 1999. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2616.txt

[21] P. K. Gadepalli, G. Peach, L. Cherkasova, R. Aitken, and G. Parmer, "Challenges and opportunities for efficient serverless computing at the edge," in *Proc. 38th Symp. Reliable Distrib. Syst. (SRDS)*, Oct. 2019, pp. 261–2615.

[22] P. K. Gadepalli, S. McBride, G. Peach, L. Cherkasova, and G. Parmer, "Sledge: A serverless-first, light-weight wasm runtime for the edge," in *Proc. 21st Int. Middleware Conf.*, 2020, pp. 265–279.

[23] A. Goel and B. Thangaraju, "Authenticating distributed systems using SPIRE over kubernetes cluster," in *Proc. IEEE Int. Conf. Electron., Comput. Commun. Technol. (CONECCT)*, Jul. 2022, pp. 1–6.

[24] J. Gross, I. Ganga, and T. Sridhar. *Geneve: Generic Network Virtualization Encapsulation*, document RFC 8926, RFC Editor, Nov. 2020.

[25] A. Hall and U. Ramachandran, "An execution model for serverless functions at the edge," in *Proc. Int. Conf. Internet Things Design Implement.*, Apr. 2019, pp. 225–236.

[26] T. Herbert and P. Lapukhov. *Identifier-Locator Addressing for IPv6*, document Internet-Draft draft-herbert-intarea-ila-01, IETF Secretariat, Mar. 2018.

[27] C. Huang, D. A. Maltz, J. Li, and A. Greenberg, "Public DNS system and global traffic management," in *Proc. IEEE INFOCOM*, Apr. 2011, pp. 2615–2623.

[28] G. Huston and C. S. Apnic, "In defence of NATs," *Internet Protoc. J.*, vol. 20, no. 3, pp. 25–33, 2017.

[29] V. Ishakian, J. Akinwumi, F. Esposito, and I. Matta, "On supporting mobility and multihoming in recursive internet architectures," *Comput. Commun.*, vol. 35, no. 13, pp. 1561–1573, Jul. 2012.

[30] D. Jen, M. Meisel, H. Yan, D. Massey, L. Wang, B. Zhang, and L. Zhang, "Towards a new internet routing architecture: Arguments for separating edges from transit core," in *Proc. HotNets*, 2008, pp. 103–108.

[31] E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. Yadwadkar, J. E. Gonzalez, R. A. Popa, I. Stoica, and D. A. Patterson, "Cloud programming simplified: A Berkeley view on serverless computing," 2019, *arXiv:1902.03383*.

[32] K. Kaffes, N. J. Yadwadkar, and C. Kozyrakis, "Centralized core-granular scheduling for serverless functions," in *Proc. ACM Symp. Cloud Comput.*, Nov. 2019, pp. 158–164.

[33] C. Karayiannis, "The apache web server," in *Web-Based Projects that Rock Class*. Berlin, Germany: Springer, 2019, pp. 1–37.

[34] A. Khan, "Key characteristics of a container orchestration platform to enable a modern application," *IEEE Cloud Comput.*, vol. 4, no. 5, pp. 42–48, Sep./Oct. 2017.

[35] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," in *Proc. Conf. Appl., Technol., Archit., Protocols Comput. Commun. (SIGCOMM)*, 2007, pp. 181–192.
[36] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, P. Ingram, E. Jackson, and A. Lambeth, "Network virtualization in multi-tenant datacenters," in *Proc. 11th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2014, pp. 203–216.
[37] M. Lasserre *et al. Framework for Data Center (DC) Network Virtualization*, document RFC 7365, RFC Editor, Oct. 2014.
[38] E. Lear and R. Droms. *What's in a name: Thoughts from the NSRG*, document Internet-Draft draft-irtf-nsrg-report-10, IETF Secretariat, Sep. 2003.
[39] W. Li, Y. Lemieux, J. Gao, Z. Zhao, and Y. Han, "Service Mesh: Challenges, state of the art, and future research opportunities," in *Proc. IEEE Int. Conf. Service-Oriented Syst. Eng. (SOSE)*, Apr. 2019, pp. 122–1225.
[40] M. Mahalingam *et al. Virtual Extensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks Over Layer 3 Networks*, document RFC 7348, RFC Editor, Aug. 2014. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7348.txt
[41] A. E. Malki and U. Zdun, "Guiding architectural decision making on service mesh based microservice architectures," in *Proc. Eur. Conf. Softw. Archit.* Cham, Switzerland: Springer, 2019, pp. 3–19.
[42] J. Ménétrey, M. Pasin, P. Felber, and V. Schiavoni, "WebAssembly as a common layer for the cloud-edge continuum," in *Proc. 2nd Workshop Flexible Resource Appl. Manag. Edge*, Jul. 2022, pp. 3–8.
[43] D. Meyer, L. Zhang, and K. Fall. *Report From the IAB Workshop on Routing and Addressing*, document RFC 4984, RFC Editor, Sep. 2007. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4984.txt
[44] P. Mockapetris and K. J. Dunlap, "Development of the domain name system," in *Proc. Symp. Proc. Commun. Archit. Protocols (SIGCOMM)*, 1988, pp. 123–133.
[45] P. Nikander, A. Gurtov, and T. R. Henderson, "Host identity protocol (HIP): Connectivity, mobility, multi-homing, security, and privacy over IPv4 and IPv6 networks," *IEEE Commun. Surveys Tuts.*, vol. 12, no. 2, pp. 186–204, 2nd Quart., 2010.
[46] M. D. O'Dell. *GSE—An Alternate Addressing Architecture for IPv6*, document Internet-Draft draft-ietf-ipngwg-gseaddr-00, IETF Secretariat, Feb. 1997.
[47] N. Paskin, "Digital object identifier (DOI) system," *Encyclopedia Library Inf. Sci.*, vol. 3, pp. 1586–1592, Dec. 2010.
[48] C. E. Perkins, "Mobile IP," *IEEE Commun. Mag.*, vol. 35, no. 5, pp. 84–99, May 1997.
[49] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, and K. Amidon, "The design and implementation of open vSwitch," in *Proc. 12th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2015, pp. 117–130.
[50] Y. Rekhter, T. Li, and S. Hares. *A Border Gateway Protocol 4 (BGP-4)*, document RFC 4271, RFC Editor, Jan. 2006. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4271.txt
[51] E. Rescorla. *HTTP Over TLS*, document RFC 2818, RFC Editor, May 2000. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2818.txt
[52] M. Roberts and J. Chapin, *What is Serverless?*. Sebastopol, CA, USA: O'Reilly, 2017.
[53] E. Rosen and Y. Rekhter. *BGP/MPLS IP Virtual Private Networks (VPNs)*, document RFC 4364, RFC Editor, Feb. 2006.
[54] E. Rosen, A. Viswanathan, and R. Callon. *Multiprotocol Label Switching Architecture*, document RFC 3031, RFC Editor, Jan. 2001. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3031.txt
[55] A. Rossberg, B. L. Titzer, A. Haas, D. L. Schuff, D. Gohman, L. Wagner, A. Zakai, J. F. Bastien, and M. Holman, "Bringing the web up to speed with WebAssembly," *Commun. ACM*, vol. 61, no. 12, pp. 107–115, Nov. 2018.
[56] K. Schomp and R. Al-Dalky, "Partitioning the internet using anycast catchments," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 50, no. 4, pp. 3–9, Oct. 2020.
[57] K. Schomp, O. Bhardwaj, E. Kurdoglu, M. Muhaimen, and R. K. Sitaraman, "Akamai DNS: Providing authoritative answers to the world's queries," in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. Appl., Technol., Archit., Protocols Comput. Commun.*, Jul. 2020, pp. 465–478.
[58] *Secure Production Identity Framework for Everyone*. Accessed: May 8, 2021. [Online]. Available: https://www.spiffe.io
[59] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.
[60] M. S. L. D. Silva *et al. Integrating SPIFFE and SCONE to Enable Universal Identity Support for Confidential Workloads*, Universidade Federal de Campina Grande, 2021.
[61] A. Tosatto, P. Ruiu, and A. Attanasio, "Container-based orchestration in cloud: State of the art and challenges," in *Proc. 9th Int. Conf. Complex, Intell., Softw. Intensive Syst.*, Jul. 2015, pp. 70–75.
[62] B. Walters, "VMware virtual platform," *Linux J.*, vol. 1999, no. 63, p. 6, 1999.

ANDREW BABAKIAN is currently pursuing the Ph.D. degree in engineering with the University of Technology Sydney, Australia. He is also a Technologist and the Director at the Networking and Advanced Security Business Group, VMware, driving innovation efforts in software defined networking and identity technologies for cloud-native architectures. His current research interests include naming systems and the evolution of Internet identifiers.

PERE MONCLUS received the double M.S.E.E. degree from the Universitat Politecnica de Catalunya and the Politecnico di Torino. He is currently the CTO at the Networking and Advanced Security Business Group, VMware. He is responsible for defining strategy and leading an innovation team driving the evolution of software defined networking and cloud-native architectures. Previously, a Distinguished Engineer at Cisco, where he led multiple successful products in the area of security, load balancing, and data centers. He holds over 30 patents.

ROBIN BRAUN (Life Senior Member, IEEE) received the B.Sc. degree (Hons.) from Brighton University, Brighton, U.K., in 1980, and the M.Sc.(Eng.) and Ph.D. degrees from the University of Cape Town, Cape Town, South Africa, in 1982 and 1986, respectively. He started his academic career at the University of Cape Town, in 1986. In 1998, he moved to the University of Technology Sydney, Australia, where he occupied the Chair of Telecommunications Engineering. Prior to moving to academia, he spent ten years in industry, mostly with Philips and Plessey, where he worked on the design of precision electronic distance measuring equipment. He is currently an Honorary Professor with the School of Electrical and Data Engineering. His recent work has been in network protocols and the management of complex next-generation networks. He is very active in software-defined networks. He is also a Founder Member of Australia and New Zealand Software Defined Networking (ANZSDN).

JUSTIN LIPMAN (Senior Member, IEEE) received the Ph.D. degree in telecommunications engineering from the University of Wollongong, Australia, in 2004. He is an Industry Associate Professor at the University of Technology Sydney (UTS) and a Visiting Associate Professor at the Hokkaido University's Graduate School of Engineering. He is the Director of research translation with the Faculty of Engineering and IT and the Director of the RF Communications Technologies (RFCT) Laboratory, where he leads industry engagement in RF technologies, cybersecurity, privacy preserving technologies, the Internet of Things, and tactile Internet. He was previously the Deputy Chief Scientist of the Food Agility Cooperative Research Center. Prior to joining UTS, over a 12 year period, he held a number of senior management and technical leadership roles at Intel and Alcatel driving research and innovation, product development, architecture, and IP generation. His research interests include enabling "things" to be adaptive, connected, distributed, ubiquitous, and secure. He serves as a Committee Member at Standards Australia contributing to International IoT Standards and Digital Twins.