AFIN 2019

The Eleventh International Conference on Advances in Future Internet

ISBN: 978-1-61208-747-4

October 27 - 31, 2019

Nice, France

AFIN 2019 Editors

Eugen Borcoci, University Politehnica of Bucharest, Romania

AFIN 2019

Forward

The Eleventh International Conference on Advances in Future Internet (AFIN 2019), held between October 27, 2019 and October 31, 2019 in Nice, France, continued a series of events dealing with advances on future Internet mechanisms and services.

We are in the early stage of a revolution on what we call Internet now. Most of the design principles and deployments, as well as originally intended services, reached some technical limits and we can see a tremendous effort to correct this. Routing must be more intelligent, with quality of service consideration and 'on-demand' flavor, while the access control schemes should allow multiple technologies yet guarantying the privacy and integrity of the data. In a heavily distributed network resources, handling asset and resource for distributing computing (autonomic, cloud, on-demand) and addressing management in the next IPv6/IPv4 mixed networks require special effort for designers, equipment vendors, developers, and service providers.

The diversity of the Internet-based offered services requires a fair handling of transactions for financial applications, scalability for smart homes and ehealth/telemedicine, openness for web-based services, and protection of the private life. Different services have been developed and are going to grow based on future Internet mechanisms. Identifying the key issues and major challenges, as well as the potential solutions and the current results paves the way for future research.

We take here the opportunity to warmly thank all the members of the AFIN 2019 technical program committee, as well as all the reviewers. The creation of such a high quality conference program would not have been possible without their involvement. We also kindly thank all the authors who dedicated much of their time and effort to contribute to AFIN 2019. We truly believe that, thanks to all these efforts, the final conference program consisted of top quality contributions.

We also thank the members of the AFIN 2019 organizing committee for their help in handling the logistics and for their work that made this professional meeting a success.

We hope that AFIN 2019 was a successful international forum for the exchange of ideas and results between academia and industry and to promote further progress in the field of future Internet. We also hope that Nice, France provided a pleasant environment during the conference and everyone saved some time to enjoy the charm of the city.

AFIN 2019 Chairs

AFIN Steering Committee
Renwei (Richard) Li, Future Networks, Huawei, USA
Eugen Borcoci, University Politehnica of Bucharest, Romania
Alex Galis, University College London, UK
R.D. van der Mei (Rob), Centre for Mathematics and Computer Science (CWI), the Netherlands
Jun Peng, University of Texas - Rio Grande Valley, USA
Hiroyuki Sato, University of Tokyo, Japan
Sergio Ilarri, University of Zaragoza, Spain
Christos Bouras, University of Patras and Research Academic Computer Technology Institute, Greece
Adel Al-Jumaily, University of Technology, Sydney, Australia

AFIN 2019
Committee

AFIN Steering Committee

Renwei (Richard) Li, Future Networks, Huawei, USA
Eugen Borcoci, University Politehnica of Bucharest, Romania
Alex Galis, University College London, UK
R.D. van der Mei (Rob), Centre for Mathematics and Computer Science (CWI), the Netherlands
Jun Peng, University of Texas - Rio Grande Valley, USA
Hiroyuki Sato, University of Tokyo, Japan
Sergio Ilarri, University of Zaragoza, Spain
Christos Bouras, University of Patras and Research Academic Computer Technology Institute, Greece
Adel Al-Jumaily, University of Technology, Sydney, Australia

AFIN Research/Industry Committee

Kiran Makhijani, Huawei Technologies, USA
Alexander Papaspyrou, adesso GmbH, Germany
Martin Zelm, INTEROP - Virtual Lab, Brussels, Belgium

AFIN 2019 Technical Program Committee

Rocío Abascal-Mena, Universidad Autónoma Metropolitana - Cuajimalpa, Mexico
S M Nahian Al Sunny, University of Arkansas, USA
Cristina Alcaraz, University of Malaga, Spain
Muhammad Aleem, Capital University of Science and Technology (CUST), Pakistan
Adel Al-Jumaily, University of Technology, Sydney, Australia
Rafael Angarita, Isep Paris & Inria Paris, France
Rachida Aoudjit, Université Mouloud Mammeri de Tizi Ouzou, Algeria
Zubair Baig, Edith Cowan University, Australia
Marcin Bajer, ABB Corporate Research, Kraków, Poland
Martine Bellaïche, Polytechnique Montréal, Canada
Paolo Bellavista, University of Bologna, Italy
Ana M. Bernardos, Universidad Politécnica de Madrid, Spain
Peter Bloodsworth, University of Oxford, UK
Eugen Borcoci, University Politehnica of Bucharest, Romania
Kechar Bouabdellah, University of Oran 1 Ahmed Ben Bella, Algeria
Christos Bouras, University of Patras and Research Academic Computer Technology Institute, Greece
Stefan Bosse, University of Koblenz-Landau, Germany
Hongyu Pei Breivold, ABB Corporate Research, Sweden
Manuel José Cabral dos Santos Reis, University of Trás-os-Montes e Alto Douro, Portugal
Lianjie Cao, Hewlett Packard Labs, Palo Alto, USA
Stephan Cejka, Siemens AG Austria, Austria
Kevin Chalmers, Edinburgh Napier University, UK
Philippe Cousin, Easy Global Market, France
Monireh Dabaghchian, George Mason University, USA

Table of Contents

5G Slicing Management and Orchestration Architectures - Any Convergence?

Eugen Borcoci, Cosmin Contu, Andra Ciobanu
University POLITEHNICA of Bucharest - UPB
Bucharest, Romania
Emails: eugen.borcoci@elcom.pub.ro, cosmin.contu@elcom.pub.ro, andraciobanu90@yahoo.com

***Abstract* — Management and Orchestration (M&O) are essential activities in 5G slicing systems. Essentially, the integrated M&O based on The European Telecommunications Standards Institute (ETSI) Management and Orchestration (MANO) is the basis, but enriched, in order to cope with slicing. In particular, supporting technologies like Network Function Virtualization (NFV) and Software Defined Networks (SDN) are considered, to deliver functional components for 5G slicing M&O. The multi-tenant, multi-domain, multi-operator, end-to-end (E2E) features of the 5G slicing determine a high complexity for M&O. Consequently, many different architectural variants have been already proposed, studied and developed in recent studies, standards and projects. The study in this paper is useful because, despite many efforts, (spent in the last five years) much heterogeneity and different solutions still exist, even at the M&O architectural level. This paper analyzes the existing common parts and differences between several 5G slicing architectures, in an attempt to identify a degree of "convergence", while considering the MANO as a base architecture.**

Keywords — 5G slicing; Management and Orchestration; Software Defined Networking; Network Function Virtualization. Service management; Resource management.

I. INTRODUCTION

The emergent 5G mobile network technologies offer powerful features, in terms of capacity, speed, flexibility and services, to answer the increasing demand and challenges addressed to communication systems and Internet [1][2]. 5G can provide specific types of services to satisfy simultaneously various customer/tenant demands in a multi-x fashion (the notation –x stands for: tenant, domain, operator and provider).

The 5G network slicing concept (based on virtualization and softwarization) enables programmability and modularity for network resources provisioning, adapted to different vertical service requirements (in terms of bandwidth, latency, mobility, etc.) [2]-[6]. In a general view, a *Network Slice* (NSL) is a managed logical group of subsets of resources, Physical/Virtual network functions (PNFs/VNFs), placed in the architectural Data Plane (DPl), Control Plane (CPl) and Management Plane (MPl). The slice is programmable and has the ability to expose its capabilities to the users.

Network Function Virtualization [7]-[9] and Software Defined Networks can cooperate [10] to manage and control the 5G sliced environment, in a flexible and programmable way.

Management and Orchestration (M&O) is a crucial subsystem in 5G. Such topics constitute the object of standardization organizations and forums among which The 3rd Generation Partnership Project (3GPP), The 5G Infrastructure Public Private Partnership (5GPPP), and ETSI are representative [11]-[16]. They cooperate in order to harmonize their specifications. For instance, the 3GPP-defined management system interacts with ETSI's NFV MANO system to enable the resource management for virtualized Core Network (CN), virtualized Radio Access Network (RAN) and network slicing. ETSI collaboration with 3GPP – especially the Service and System Aspects Fifth (SA5) Working Group – is a key throughout the specification work of both ETSI NFV Releases 2 and 3, to ensure interoperability between management systems.

ETSI NFV has recently designed new features to support 5G networks. 5G resource M&O aspects were added on top of the NFV Release 2 framework. New NFV Release 3 [9] topics related to 5G include: "Support for network slicing in NFV", "Management over multi-administrative domains", and "Multi-site network connectivity". These features are essential to address the variety of applications expected to run on top of a 5G system, whether using distributed resources over multiple sites, centralized or a combination of both.

However, it is recently recognized that a complete understanding of the relationship of an M&O system and a slicing system is still missing [2]. Even more, it is not yet a general/common agreement on the slice itself; several definitions exist, having major impacts and relationships to the M&O.

In the simplest view, a slice is a service with resource guarantees. Here, the slicing system and the orchestration system are identical. At the other end of approaches, a slice is a complex entity, i.e., a collection of resources (computing, networking, storage) – that constitute a virtual logical network (and customizable), embedded in some physical networking infrastructure. Inside such a slice, the slice owner/tenant has partial or even full freedom to enforce its own management and control (M&C) policies and actions. Many studies and standards adopted the slice complex definition; this is also considered in this work, given the high flexibility that it can offer to the tenants. On the other hand, the complex structure of such a slice induces M&O complexity and leads to a large variety of possible architectural approaches.

Given the rather large variety of architectural proposals, there is an interest to evaluate in what degree they have similar approaches of the main "core" architectural functional set of blocks. This similarity level will be called here "convergence", although this word has usually a richer semantic. The focus of this paper is on management and orchestration sub-systems. Due to space limitation, this text cannot afford to offer detailed explanations about the architectures presented; the objective is to identify the major point of similarity of different approaches.

Therefore, this paper is mainly an overview type. Its structure is described below. Section II outlines the stakeholder roles, given that such definitions determine essentially the overall architecture. Section III evaluates whether a unified view exists at architectural level, expressed in so-called *meta-architecture*. Section IV performs an analysis of some factors that lead to non-convergent refined M&O architectures. Section V uses some examples extracted from various studies and projects to illustrate the heterogeneity of solutions. Section VI summarizes conclusions and future work.

II. Stakeholder Roles

The layered structure of the 5G slicing M&O strongly depends on the definition of stakeholder roles (also called *business model*). Different business models aim to support multi-tenant, multi-domain end-to-end (E2E) and multi-operator capabilities. A basic model (see A. Galis, [17]) defines four roles:

Infrastructure Provider (InP) – owns and manages the physical infrastructure (network/cloud/data center). It could lease its infrastructure (as it is) to a slice provider, or it can itself construct slices and then lease the infrastructure in network slicing fashion.

Network Slice Provider (NSLP) – can be typically a telecommunication service provider (owner or tenant of the infrastructures from which network slices are constructed). The NSLP can construct multi-tenant, multi-domain slices, on top of infrastructures offered by one or several InPs.

Slice Tenant (SLT) – is the generic user of a specific slice, including network/cloud/data centres, which can host customized services. The SLTs can request from a NSLP to create a new slice instance. The SLT can lease virtual resources from one or more NSLP in the form of a virtual network, where the tenant can realize, manage and provide *Network Services* (NS) to its individual end users. A NS is a composition of *Network Functions (NFs)*, defined in terms of the individual NFs and the mechanism used to connect them. A single tenant may have one or several slices in its domain.

End User (EU) - consumes (part of) the services supplied by the slice tenant, without providing them to other business actors.

The above business model is recursive (see Ordonez et al., [3]), i.e., a tenant can at its turn to offer parts of its sliced resources to other tenants. Other variants of business models are presented in [17].

Several recent Public Private Partnership (PPP) Phase I/II collaborative research are running, having as objectives 5G technologies [17]. Some of them extended the list of role definitions to allow various possible customer-provider relationships between verticals, operators, and other stakeholders. In [2] one can find a more refined business model:

Service Customer (SC): uses services offered by a Service Provider (SP). The vertical industries are considered as typical examples of SCs.

Service Provider (SP): generic role**,** comprising three possible sub-roles, depending on the service offered to the SC: *Communication SP* offers traditional telecom services; *Digital SP* offers digital services (e.g., enhanced mobile broadband and IoT to various verticals); *Network Slice as a Service* (NSaaS) *Provider* offers an NSL and its services. The SPs have to design, build and operate services using aggregated network services.

Network Operator (NOP): orchestrates resources, potentially from multiple *virtualized infrastructure providers* (VISP). The NOP uses aggregated virtualized infrastructure services to design, build, and operate network services that are offered to SPs.

Virtualization Infrastructure SP (VISP): offers virtualized infrastructure services and designs, builds, and operates virtualization infrastructure(s) (networking and computing resources). Sometimes a VISP offers access to a variety of resources by aggregating multiple technology domains and making them accessible through a single Application Programming Interface (API).

Data Centre SP (DCSP): designs, builds, operates and offers data center services. A DCSP differs from a VISP by offering "raw" resources (i.e., host servers) in rather centralized locations and simple services for consumption of these raw resources.

The hierarchy of this model (in the top-down sense of a layered architecture) is: SC, SP, NOP, VISP, DCSP. Note that, in practice, a single organization can play one or more roles of the above list.

III. A Generic 5G Management Meta-Architecture

The analysis of the convergence degree between many architectural proposals (in 5G and in particular, in 5G slicing) leads to the question: *is there any high-level consensus architecture?* Recently, the document [2], authored by 5G PPP Architecture Working Group has identified a set of requirements for a consensus/meta 5G high-level architecture (collecting some M&O fundamental functionalities). The identified features are general for 5G and in particular applicable also to the slicing approach. This architecture should be able to support:

a. individual control of NFs (their distribution/placement, number of instances, deployment of an execution environment, management of the instances' states, start/stop the instances).

b. individual NFs chaining into services (NF graphs) facilitated by different control mechanisms at network level (e.g., the NFs chaining can be SDN -controlled).

c. different underlying execution environments: various virtualization techniques (virtual machines (VM), containers, or plain processes) in clusters of different sizes (from a CPU board to an entire large-scale data center) over different, specialized "technological domains" - i.e., from some simple hardware, up to complex networking environments (wireless, optics, cable).

d. working across different "organizational", or administrative domains, i.e., owned by network operators or companies and using various business models (e.g., network operators can be separated from cloud infrastructure operators).

e. a large range of applications with different specific requirements (in terms of resource, deployment, orchestration and optimization goals);

f. subdivision of the infrastructure in logical separated and isolated slices – while offering different levels of guaranteed performance to their tenants.

Note that slicing capabilities – can be seen as part of a M&O system. However, there is no general consensus on this inclusion. There are proposals to position a slicing system underneath or above a MANO system.

Several core roles have emerged from the above requirements: end user, function developer, application developer, validation and verification entity, tenant (owner of applications), operator (not necessarily encompassing slicing operator) infrastructure provider (network, cloud), etc., [2]. These can be mapped onto the roles described in Section II. Overlaps can exist between some of the above. Also, the mapping of the above roles on real organizations roles is flexible.

The requirements listed above actually drive the definition of the M&O meta-architecture, in the sense that no matter the solution will be, the six functionalities should be included. These define a general level of convergence from architectural point of view. A particular architecture will be a refinement of the meta-one.

Another general aspect is related to the different time scales of different operations. One can distinguish between *"orchestration"* and *"control"* actions. The first are mid-long-time scales operations, relatively heavy-weight (e.g., optimization of the overall structure of a service, group of services, or slices). The second class comprises short time scales operations (e.g., light-weight operations, flow routing, etc.). We defend here the idea that such a logical separation should exist (it is natural) between functional elements performing the orchestration w.r.t. those dedicated to control; however, in different refinements of the meta-architecture this separation is not quite obvious; this, again, leads to heterogeneity of approaches.

The basic framework for a high-level meta-architecture is offered by ETSI NFV (Figure 1). The main M&O blocks are: the NFV Orchestration (NFVO), VNF Manager (VNFM) and Virtual Infrastructure Manager (VIM). If the principle of separation between the orchestration and control is applied, then the specific network configuration tasks (e.g., connectivity - related) can be outsourced to a separate SDN controller, working under command of the NFVO. An alternative could be, to split the NFVO into two parts – orchestrator and controller.

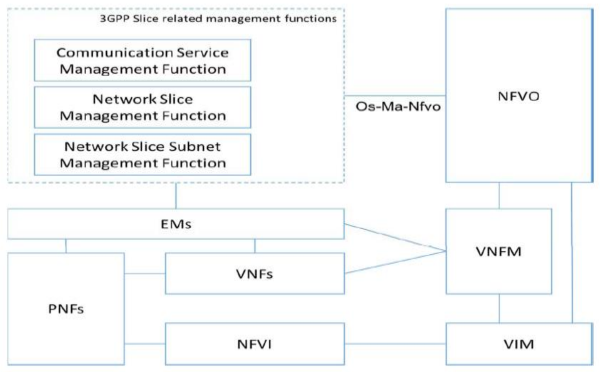

3GPP Slice related management functions
Communication Service Management Function
Network Slice Management Function
Network Slice Subnet Management Function
Os-Ma-Nfvo
NFVO
EMs
VNFM
VNFs
PNFs
NFVI
VIM

Figure 1. Network slice management in an NFV framework (ETSI GR NFV-EVE 012 V3.1.1, [14])

NFV -Network Function Virtualization; EM - Element Manager; MANO - Management and Orchestration (NFVO – NFV Orchestration; VNFM – VNF Manager; VIM Virtual Infrastructure Manager); VNF/PNF – Virtual/Physical Network Function; NFVI -NFV Infrastructure; NS-Network Service; OSS-Operations Support System.

The slicing support feature (i.e., yes/no) introduces significant differentiation between particular architectures. The slice management can be included into the NFVO (because a network slice instance (NSLI) can be actually seen as a guaranteed network service), or a separate slice manager exists (controlled by NFVO). The service management can be defined as separated from resource management, or they can be treated together. A cleaner architecture is resulting in the former case.

In multiple domain cases, the NFVOs should federate in some form with peer NFVOs, placed in a single or in multiple organizations. In some approaches, a hierarchy of service management instances is developed, having on top a multi-domain manager (working at abstract level) and then single-domain managers. The latter should perform also peer interactions.

A typical set of functional M&O blocks for a single-domain meta-architecture is [2] (top-down ordered levels): [Service management, Orchestrator, (MANO controller, SDN controller), VIM, Resources]. In a multi-domain environment, each domain should have the previous set and above all a multi-domain service manager should exist. Note that inter-domain (horizontal) peer interactions must exist between peers (e.g., Orchestrator_X <---> Orchestrator_Y).

The basic 5G slicing high level architecture proposed by ETSI [14] (Figure 1), can be considered as a meta-architecture comprising the six features exposed above. To the original ETSI NFV architecture [7][8], several new functional blocks have been added in order to support the network slicing (ETSI-NFV EVE 012 [14]).

The 3GPP TR 28.801 document [15] defines three new management functions: *Communication Service Management Function* (CSMF) – it translates the communication service requirements to NSL requirements;

Network Slice Management Function (NSMF) - responsible for the management (including instances lifecycle) of NSLIs (it derives network slice subnet requirements from the network slice related requirements); *Network Slice Subnet Management Function* (NSSMF) - responsible for the management (including lifecycle) of *Network Slice Subnet Instances* (NSSIs).

An interface is defined, i.e., Os-Ma-NFVO Reference Point (RP) with ETSI NFV-MANO. To interact in an appropriate way with NFV-MANO, the NSMF and/or NSSMF need to determine the type of NS or set of NSs, VNF and PNF that can support the resource requirements for a NSLI or NSSI, and whether new instances of these NSs, VNFs, and the connectivity to the PNFs, need to be created, or existing instances can be re-used.

Starting from the above basic architecture and considering different visions (shortly presented in the Introduction section), several groups developed a large set of variants of refined architectures [17]. Some of them are substantially different from each other. Currently there is a high heterogeneity seen in this area. The question analyzed in this paper is: *how much convergence/similarity* and how much *mutual compliancy* exists among them?

IV. WHERE DOES THE HETEROGENEITY COME FROM?

This section will summarise the factors leading to heterogeneity in the area of particular architectures. Note that, given the topics complexity, this analysis cannot be exhaustive. Some aspects are not touched, or only briefly mentioned, such as: abstraction aspects, slice isolation and security, slice composition, monitoring issues and slice optimization, details on multi-domain interactions, technological details and so on.

The *services deployment* is inherently heterogeneous, depending on applications to be supported. An example is the traffic locality property (at the edge of the network/slice or crossing the core part). An orchestrator should be aware of such traffic properties and, if necessary, deploy the corresponding network functions at the mobile edge. The orchestrator needs to have enough topology information of slices in order to be able to install appropriate functions at right places.

The *execution environments* at the infrastructure level could also be heterogeneous. The infrastructure should provide an interface to the orchestrator, via which different functions execution can be started, stopped, paused, or migrated; the interface also provides means to influence the transport of data. Variants can exist:

- The infrastructure hides (to MANO) its information on the type of execution elements available. The infrastructure management chooses the right (i.e., “functionally possible”) realization of a function (virtual machine (VM) or container, etc.). This abstraction simplifies the MANO tasks, but makes difficult for the infrastructure manager to decide what is “performance-optimal” in the absence of information about the performance requirements of an entire service, and the relationships to other services.
- The infrastructure provides to the MANO information on available types of execution resources (quantity, locations, etc.). So, the MANO has enough information to optimize the execution environment. The price paid is a higher burden for MANO. Note that such an approach should consider the degree of trust between the infrastructure provider and MANO entity, especially in multi-domain environment.

The *hardware heterogeneity* at infrastructure level can also determine many variants, e.g., virtualization methods and other factors (e.g., Field programmable gate arrays (FPGA), Graphics processing unit (GPU) implementations, hardware accelerators, etc.).

The classical principle of *vertical separation of services* in *network-related* (i.e., connectivity–oriented) and *application-level services* (e.g., caching, video transcoding, content-oriented, web server, etc.) could be preserved or not. The separation will require one orchestrator vs. separate network/service orchestrators. One can speak about segregated or integrated orchestration, respectively. Concerning slicing, one can define some slices offering essentially connectivity services and other dedicated to high-level applications. The clear separation of areas of responsibility over resources could be an advantage for operational stability (e.g., a segregated RAN orchestrator could still maintain basic RAN services even if an application-oriented orchestrator fails). On the other hand, the integrated orchestration could be attractive, in particular for operators, if both kinds of services could be orchestrated in the same fashion (and possibly even with the same orchestration infrastructure). These two options also determine heterogeneity at M&O architectural level.

Segregated orchestrators lead to a more complex overall architecture. One must assign areas of responsibilities from a resource perspective (which orchestrator controls - what resources); one should identify services pertaining to each orchestrator. The split of service is also a problem, i.e., the service description should define the “network” and “application-facing” parts of the service. Aligning the control decisions taken by these two kinds of orchestrators in a consistent way is also not trivial. In an integrated orchestration approach, all these problems disappear. However, an integrated orchestrator might be very complex if required to treat substantially different services (a one-size-fits-all orchestration approach is rather not the best choice). An integrated orchestrator is a more challenging piece of software (from both dependability and performance perspectives) but would result in a simpler overall architecture.

Considering the above rationale, we defend the idea that from the slicing point of view, a segregate orchestrator is a better choice.

However, in practice, both approaches have been pursued in different projects. Currently, a final verdict commonly agreed, on segregated versus integrated orchestration is not yet available. Apparently, there is no need to standardize this option, as long as both of them could be realized inside a meta-architecture. So, for the time being, we can state that M&O heterogeneity, from this point of view, will last.

Another architectural choice is on *"flat"* or *"hierarchical"* orchestration. In the flat solution, a single instance of a particular orchestrator type is in charge of all assigned resources. In the hierarchical solution, there are multiple orchestrators (a "hierarchical" model is needed, when orchestrators know to talk to each other). Note that a hierarchical orchestrator is *not necessarily* a segregated one, because all hierarchy members could deal with the same type of services.

In many projects and studies, the hierarchical M&O option is chosen [6][17]-[20]. However, several issues should be solved in each of the two solutions [2]:

- The *number of hierarchy levels* and each member responsibility area could be fixed or adaptive (upon load changes the responsibility areas can be split/merged; new hierarchy levels can be added/removed and new orchestrator instances can be started or some old ones can be stopped). However, the adaptive option is highly complex, given the inherent dynamicity capability required.
- *North/south vertical interfaces* between the orchestrators must be defined. In a flat model, the service requests are received by an orchestrator's northbound interface (NBI). At its south bound the orchestrator communicates with NBI of the abstracted infrastructure (VIM). These two NBIs are structurally different. In a hierarchical model, an orchestrator should be able to communicate with a lower level orchestrator through a different interface than for VIM. So, an orchestrator should be able to use different NBIs (NBI of a VIM, or NBI of a lower-level orchestrator). It is still in study to create uniform interfaces; the advantage would be that from the perspective of a higher-level orchestrator, it always talks to a VIM-style interface. The recursive orchestration could be much easier implemented.
- *Horizontal interfaces* (east/west) should be defined between peer orchestrators (those who are on the same level), if they are allowed to negotiate directly with each other (for resources). Such interfaces are naturally to exist in cross-domain slicing scenarios.
- *Multi-domain scenarios* create new problems (e.g., in the case of a multi-domain "federated" slice) [6][18]. In a flat model, each orchestrator of a domain is actually multi-orchestration capable, i.e., it can discuss/negotiate with other domains' orchestrators. In the hierarchical model, a higher-level orchestrator could exist, in charge of harmonizing multiple organizations cooperation. However, several issues are not fully solved today: which entity would run that multi-domain orchestrator, trust issues, preservation of domains independency, assuring the fairness, etc.
- *Mapping* of the *orchestration entities* (and their areas of responsibility) onto *"domains"* (in a very general sense of the word) is still an open research issue and it is also a factor of heterogeneity of the refined M&O architectures. For instance, one could have separate orchestrators for different technological domains (e.g., computational resources, optical networking infrastructure, wireless edge, etc.). However, the word "domain" can be associated to organizations/companies boundaries. Such domains have overlap with the technological ones. A third semantic is that a "domain" could be a subdivision of a larger infrastructure into an edge domain, a core domain, etc. (each one spanning multiple technologies, possibly dealing with all kinds of services in a non- segregated way).
- *Relationship* of the *M&O system* and a *slicing system* is another factor of architectural variability, depending on what the definition of a slice is. A largely agreed solution is to have a general orchestrator (configured offline), capable to trigger the construction of a new slice and then to install in this new slice its own dedicated orchestrator (before the slice run-time). To still assure the basic services outside any slice (e.g., packet forwarding at network level) one can construct an additional special orchestrator installed outside of all slices. Currently, many combinations have been proposed, and there is still no consensus on such matters. The convergence of solutions will be determined probably by the adoption of a more unique definition of a slice – which could assure better inter-operability.

V. EXAMPLES OF SLICED 5G MANAGEMNT AND ORCHESTRATION ARCHITECTURES

This section will provide some examples to illustrate the major M&O options and also the heterogeneity of the refined architectures. Given the limited dimension of this paper, the examples are included mainly for illustrative purposes, i.e. this text cannot cover a large set of refined architectures.

The 5GPPP Working Group details a 5G multi-domain architecture by defining four planes [1]: *Service, M&O, Control* and *Data* planes. The architecture also includes a *Multi-Domain Network Operating System* containing different adaptors and network abstractions above the networks and clouds heterogeneous fabrics. The *M&O* plane comprises a general *Service Management*, the *Software-Defined Mobile Network Orchestrator* (SDMO) and the ETSI NFV lower level managers (i.e., VNFM and VIM). The SDMO is composed of a *domain specific application management*, an *Inter-slice Resource Broker* and *NFV-NFVO*. The SDMO performs the E2E management of network services; it can set up slices by using the network slice templates and merge them properly at the described multiplexing point. Note the definition of a separated Control Plane. It is "horizontally" separated in two parts: intra and inter-slice control functions. "Vertically", it is organized in SDN style, i.e., with three planes: *Control applications* (inter and intra-slice); *SDN controllers*; *SDN nodes* (these are actually slicing control function blocks realized as PNF/VNFs). Note also the flexibility of SDN-NFV cooperation: some slicing control functions are seen and realized as SDN nodes. The SDN controllers are two types: *Software-Defined Mobile Network Coordinator* (SDM-X) and *Software-Defined Mobile Network Controller* (SDM-C).

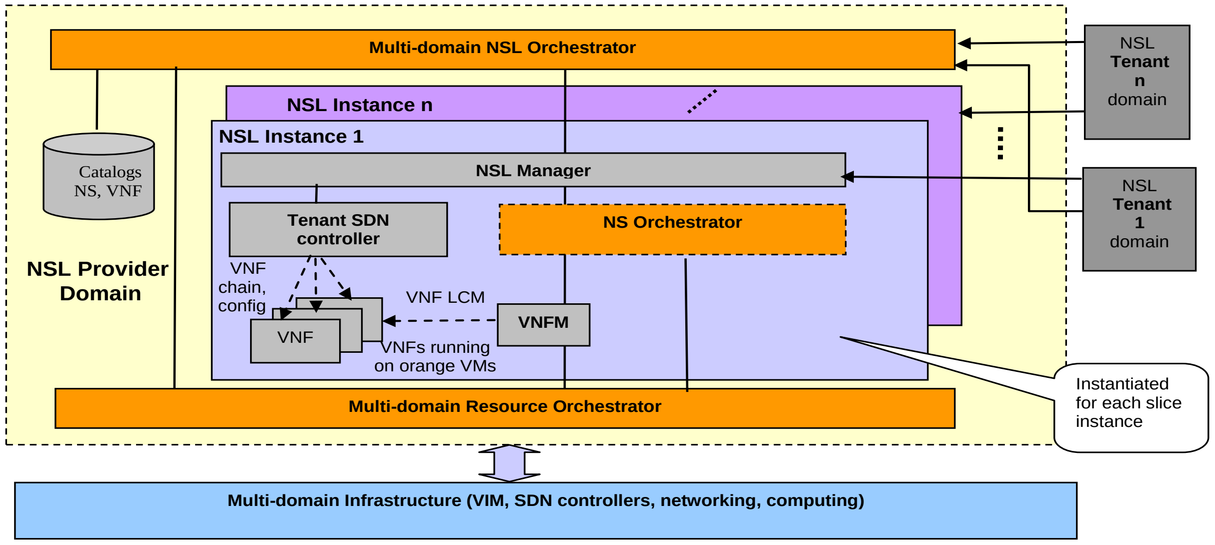

Multi-domain NSL Orchestrator
NSL Instance n
NSL Instance 1
NSL Manager
Catalogs NS, VNF
Tenant SDN controller
NS Orchestrator
NSL Provider Domain
VNF chain, config
VNF LCM
VNF
VNFM
VNFs running on orange VMs
NSL Tenant n domain
NSL Tenant 1 domain
Instantiated for each slice instance
Multi-domain Resource Orchestrator
Multi-domain Infrastructure (VIM, SDN controllers, networking, computing)

Figure 2. Run-time view of a multi-domain slicing hierarchical architecture example 1
(adapted from ETSI GR NFV-EVE 012 [14] and Ordonez-Lucena [3][19])

NS – Network Service; NSL - Network Slice; VNF – Virtualized Network Function; VNFM – VNF Manager; SDN Software Defined Networking; LCM – Life Cycle Management; VIM – Virtual Infrastructure Manager

In Figure 2, a multi-domain hierarchical slicing architecture (viewed at run-time phase) is presented according to the proposal from ETSI GR NFV-EVE 012 [14] and J.Ordonez-Lucena et al. [3][19]. The main M&O entity is the *Network Slice Provider (NSLP).* Note the multiple levels of orchestrators and separation between service and resource management. Inside NSLP, a highest layer *NSL Orchestrator* (NSLO) (configured offline) has a main role in the *creation* phase of slices and also in the *run-time* phase. In the creation phase, NSLO receives the order to deploy a NSLI for a tenant (or the NSLP decides itself to construct a slice). The NSLO should have enough information (including on multi-domain resource availability) in order to check the feasibility of the order. To accomplish this, it interacts with a lower level *Resource Orchestrator* (RO) (which aggregates resource information from several domains (InPs)), and also accesses the VNF and NS catalogues.

The NSL provider plays a role of an infrastructure tenant; it rents the infrastructure resources owned by the underlying infrastructure providers, and uses them to provision the NSL instances. The RO uses the set of resources supplied by the underlying VIMs/WIMs and optimally dispatches them to the NSL instances. All the NSL instances are simultaneously provided with the needed resources to satisfy their requirements and preserve their performance isolation. Note that in this high-level architecture proposal it is not detailed how the multi-domain capable RO is implemented in order to assure inter-domain independence.

For each network slice instance (NSLI), individual M&O entities are dynamically created. Each NSLI has its own management plane (to get slice isolation) composed of: NSL Manager, NS Orchestrator (NSO), Tenant SDN Controller and VNF Manager (VNFM).

Taleb et al. [6] recently proposed a multi-domain slicing hierarchical, complex orchestration architecture (see Figure 3). It is structured into four major strata: *Multi-domain Service Conductor*, *Domain-specific Fully-Fledged Orchestration*, *Sub-Domain Management and Orchestration (MANO) and Connectivity*, *and Logical Multidomain Slice Instance* stratum. The architecture introduces (at top level) a novel architectural plane - *Service Broker* (SB), to handle incoming slice requests from verticals, for instance *Mobile Virtual Network Operators* (MVNO), and application providers. The main SB operations are: NS admission control and negotiation, considering service aspects; management of slice user/owner relationship enabling a direct tenant interface with the MSC plane; billing and charging; NSLI scheduling, i.e., start and termination instant of time, related with slice composition and decommission.

Below the SB, a *Multi-domain Service Conductor* (MSC) plane is defined, to perform service management across *federated domains*. The MSC stratum analyzes and maps the service requirements of incoming multi-domain slice requests onto the respective administrative domains. It also maintains the desired service performance throughout the entire service life-cycle. Inside MSC, a *Service Conductor (SC)* is placed on top; the SC analyses and maps the service requirements of incoming slice requests onto appropriate administrative domains and maintains the desired service performance during service lifecycle. Below SC, a *Cross-domain Slice Coordinator* is defined for each slice, which aligns cloud and networking resources across federated domains and carries out the Life Cycle Management (LCM) operations of a multi-domain slice. It also establishes and controls inter-domain transport layer connectivity, assuring the desired performance. A multi-domain NSLI can combine several *Fully-Fledged NSLIs* that belong to distinct administrative domains, to get an E2E multi-domain (i.e., federated NSLI).

For each domain a *Fully-fledged NetSlice Orchestration Plane* is constructed, dealing with specific operations associated to slices instance in that domain (such as service management and slice lifecycle management). The lower layers of this specific orchestration plane comprise NFV MANO functionalities (NFVO, VNFM and VIM). Low level connectivity tasks between VNF/PNFs are performed by an SDN controller.

The 5G-MoNArch H2020 project [20] develops a hierarchical architecture consisting of four layers: *Service, M&O, Controller and Network* layer (similar to that proposed in [1] by 5GPPP). The overall functional architecture is presented in Figure 4. The Service layer comprises *Business Support Systems (BSS)*, business-level *Policy and Decision* functions, and further applications and services operated by a tenant or other external entities.

The M&O layer contains M&O functions from different network, technology, and administration domains (e.g., 3GPP public mobile network management, ETSI NFV MANO, ETSI Multi-access Edge Computing functions [ETSI MEC16], management functions of transport network or enterprise networks. The M&O layer is divided into an End-to-End (E2E) service M&O sublayer and an additional sublayer containing domain-specific management functions. An E2E network slice is composed of *Network Slice Subnet Instances (NSSIs)*, typically each from a different network domain, including subnets from radio access network (RAN), transport, and core network domains, or private networks. The M&O layer performs cross-domain coordination actions.

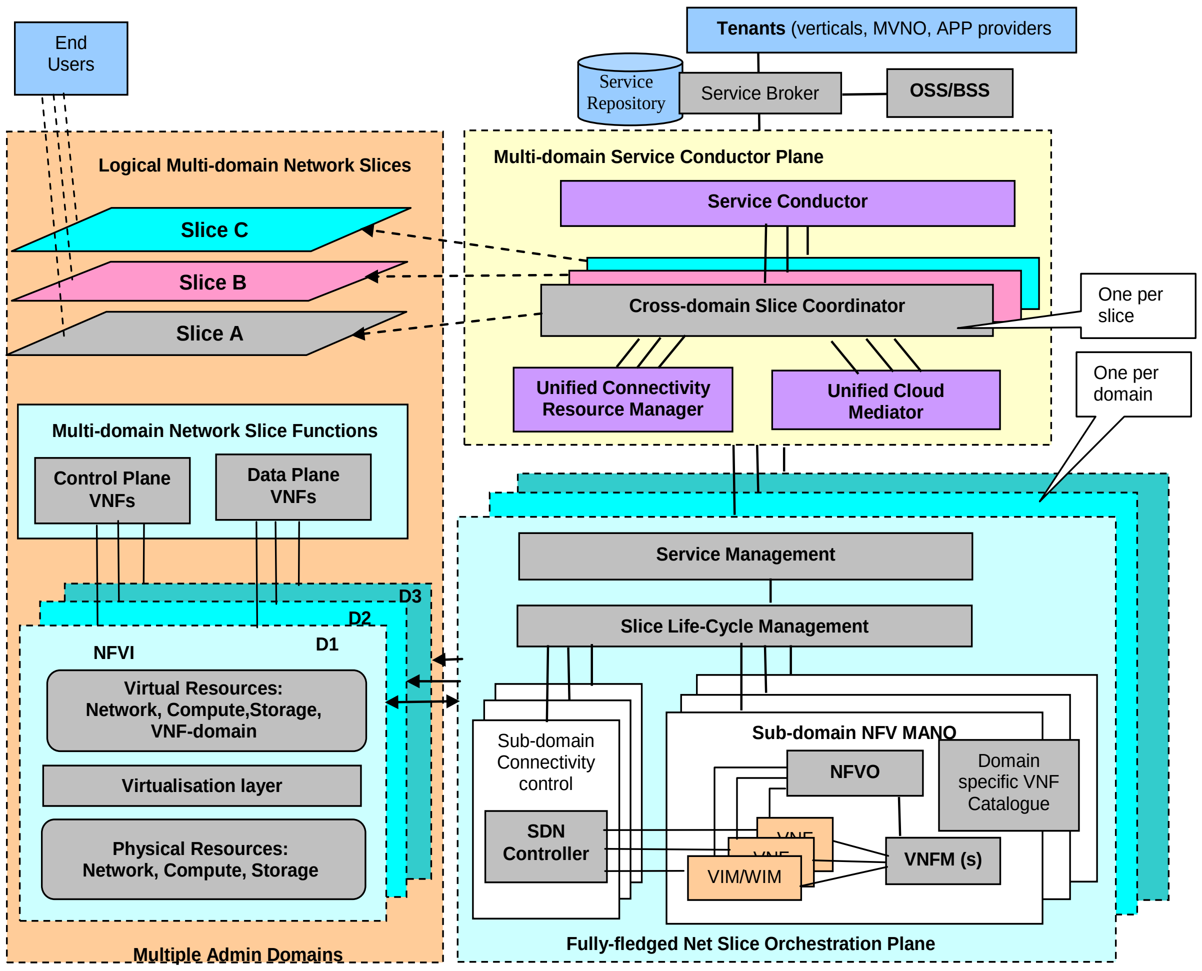

End Users
Tenants (verticals, MVNO, APP providers
Service Repository
Service Broker
OSS/BSS
Logical Multi-domain Network Slices
Slice C
Slice B
Slice A
Multi-domain Service Conductor Plane
Service Conductor
Cross-domain Slice Coordinator
One per slice
Unified Connectivity Resource Manager
Unified Cloud Mediator
One per domain
Multi-domain Network Slice Functions
Control Plane VNFs
Data Plane VNFs
D3
D2
D1
NFVI
Virtual Resources: Network, Compute,Storage, VNF-domain
Virtualisation layer
Physical Resources: Network, Compute, Storage
Multiple Admin Domains
Service Management
Slice Life-Cycle Management
Sub-domain Connectivity control
SDN Controller
Sub-domain NFV MANO
NFVO
Domain specific VNF Catalogue
VNF
VIM/WIM
VNFM (s)
Fully-fledged Net Slice Orchestration Plane

Figure 3. Multi-domain multi-tenant slicing architecture example 2 (adapted from [6])

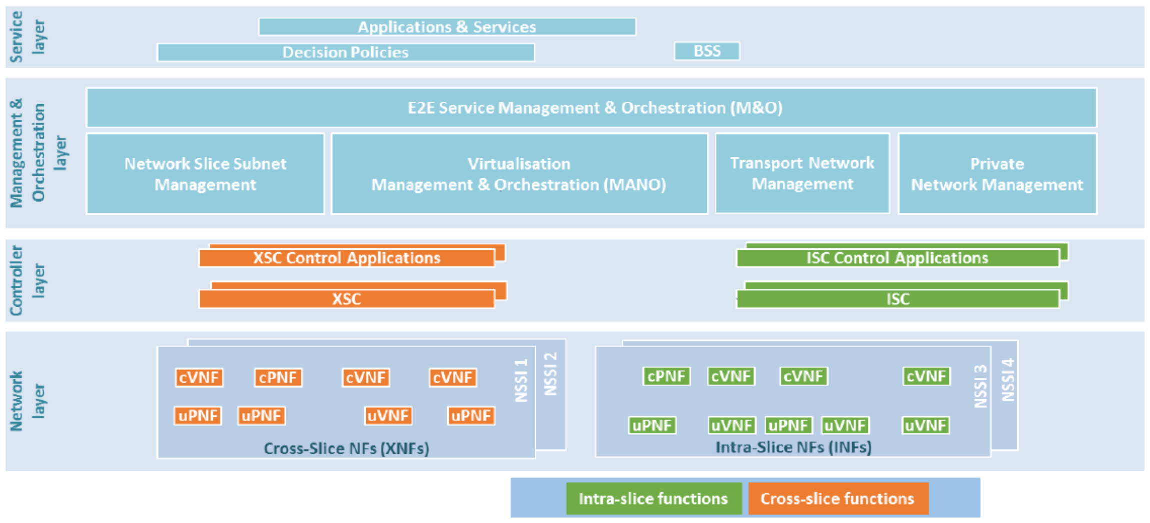


Figure 4. 5G-MoNArch high-level structure of the overall functional architecture (Source: [20])

Note again the architectural separation between the management and control. The Controller layer comprises two types of controllers- cross-slice and the intra-slice (XSC and ISC, respectively). On top of the controllers, there are *Control Applications*; together they realise the network programmability in SDN style. Each network domain has a dedicated controller that is aware of the domain technology and implementation characteristics.

Other architectures are proposed and developed in different research projects [17]. Again, all of them satisfy the characteristics of the meta-architecture described in Section III. However, different specific developments are present in their refined version.

VI. CONCLUSIONS AND FUTURE WORK

This is an overview-type paper; it analyzed different M&O architectures for 5G slicing, in order to evaluate the degree of their similarity/convergence, given the large variety of proposals existing in various studies, standards and projects.

It has been shown that business model definitions (actors) and their roles (Section II) have an important impact on the high-level definition of the architectural assembly. Actually, the variety of business models is a primary factor of architectural heterogeneity, given the different definition of actors and roles, adopted mainly from business reasons and only secondly from technical ones. Also, the definition of a slice itself is still not globally agreed upon and this naturally leads to different architectures.

However, a unifying meta-architecture has been defined (see Section III), answering to some basic requirements for 5G systems and, in particular, for 5G M&O slicing. It has been derived from ETSI MANO work complemented with additional functionalities slice-oriented. The most relevant architecture examples found in literature and developments are essentially compliant with the basic meta-architecture. It is important to note that, all relevant architectures proposed in different studies, standards and projects generally try to achieve the main meta-architecture capabilities.

On the other hand, many factors are inducing heterogeneity of the refined architecture variants, such as: multi-domain, multi-tenant, multi-operator, multi-technology

Future work can go further to consider more deeply the multi-x aspects, implementation and performance. Future work can concentrate on M&O issues such as: an appropriate cooperation between slice-specific management functional blocks. Policies need to be captured in a way that they can be automatically validated. This automation enables slice-specific functional blocks to be authorized to perform the corresponding management and configuration actions in a timely manner.

Designing computationally efficient resource allocation algorithms and conflict resolution mechanisms at each abstraction layer is also a way to flexibly assign resource on-the-fly to slices.

Lastly, one should mention new approaches for 5G slicing M&O architectures: usage of artificial intelligence and in particular, machine learning techniques in order to provide more M&O automation and capabilities of dealing with big volumes of data [21]-[24]. This domain is only at its beginning, so is an open field for further studies.

REFERENCES

[1] 5GPPP Architecture Working Group, “View on 5G Architecture”, Version 2.0, December 2017, https://5g-ppp.eu/wp-content/uploads/2017/07/5G-PPP-5G-Architecture-White-Paper-2-Summer-2017_For-Public-Consultation.pdf, [retrieved June, 2019].

[2] 5GPPP Architecture Working Group, “View on 5G Architecture”, Version 3.0, June, 2019, https://5g-ppp.eu/wp-content/uploads/2019/07/5G-PPP-5G-Architecture-White-Paper_v3.0_PublicConsultation.pdf, [retrieved June, 2019].

[3] J. Ordonez-Lucena et al., “Network Slicing for 5G with SDN/NFV: Concepts, Architectures and Challenges”, IEEE Communications Magazine, 2017, pp. 80-87, Citation information: DOI 10.1109/MCOM.2017.1600935.

[4] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, “Network Slicing in 5G: Survey and Challenges”, IEEE Communications Magazine, May 2017, pp. 94-100.

[5] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, “Network Slicing & Softwarization: A Survey on Principles, Enabling Technologies & Solutions”, IEEE Communications Surveys & Tutorials, March 2018, pp. 2429-2453.

[6] T. Taleb, I. Afolabi, K. Samdanis, and F. Z. Yousaf, "On Multi-domain Network Slicing Orchestration Architecture & Federated Resource Control", http://mosaic-lab.org/uploads/papers/3f772f2d-9e0f-4329-9298-aae4ef8ded65.pdf, [retrieved June, 2019].

[7] ETSI GS NFV 002, “NFV Architectural Framework”, V1.2.1, December, 2014.

[8] ETSI GS NFV-IFA 009, “Network Functions Virtualisation (NFV); Management and Orchestration; Report on Architectural Options”, Technical Report, V1.1.1, July, 2016.

[9] ETSI GR NFV-IFA 028, “Network Functions Virtualisation (NFV) Release 3; Management and Orchestration; Report on architecture options to support multiple administrative domains”, Technical Report, V3.1.1, January, 2018.

[10] ONF TR-526, “Applying SDN Architecture to 5G Slicing”, April 2016.

[11] 5G Network and Service Management Including Orchestration v3.14.0, Project 5G NWMO, 2019, https://www.ngmn.org/fileadmin/ngmn/content/downloads/Technical/2019/190312_5G_Network_and_Service_Management__including_Orchestration_3.14.0.pdf, [retrieved June, 2019].

[12] ETSI NFV Announces New Features to its Architecture to support 5G, https://www.etsi.org/newsroom/press-releases/1622-2019-07-etsi-nfv-announces-new-features-to-its-architecture-to-support-5g, [retrieved July, 2019].

[13] G. Daniels, ETSI tightens 3GPP 5G compatibility with NFV Release 3, July 2, 2019, https://www.telecomtv.com/content/nfv/etsi-tightens-3gpp-5g-compatibility-with-nfv-release-3-35653/, [retrieved July, 2019].

[14] ETSI GR NFV-EVE 012, Release 3 “NFV Evolution and Ecosystem; Report on Network Slicing Support with ETSI NFV Architecture Framework”, Technical Report, V3.1.1, December, 2017.

[15] 3GPP TR 28.801, "Telecommunication management; Study on management and orchestration of network slicing for next generation network", V15.0.0, September, 2017.

[16] 3GPP TS 28.530, “Management of 5G networks and network slicing; Concepts, use cases and requirements”, Rel. 15, April 2018.

[17] A. Galis, “Network Slicing- A holistic architectural approach, orchestration and management with applicability in mobile and fixed networks and clouds”, http://discovery.ucl.ac.uk/10051374/, [retrieved July, 2019].

[18] K. Katsalis, N. Nikaein and A. Edmonds, "Multi-Domain Orchestration for NFV: Challenges and Research Directions", 2016 15th Int'l Conf. on Ubiquitous Computing and Communications and International Symposium on Cyberspace and Security (IUCC-CSS), pp. 189–195, DOI: 10.1109/IUCC-CSS.2016.034, https://ieeexplore.ieee.org/document/7828601, [retrieved July, 2019].

[19] J. Ordonez-Lucena et al., "The Creation Phase in Network Slicing: From a Service Order to an Operative Network Slice", European Conference on Networks and Communications (EuCNC), 2018, https://arxiv.org/abs/1804.09642 [retrieved July, 2019].

[20] H2020-ICT-2016-2, Monarch Project, 5G Mobile Network Architecture for diverse services, use cases and applications in 5G and beyond, Deliverable D2.3, “Final overall architecture”, 2019, https://5g-monarch.eu/wp-content/uploads/2019/05/5G-MoNArch_761445_D2.3_Final_overall_architecture_v1.0.pdf, [retrieved June, 2019].

[21] V. P. Kafle et al., “Consideration on Automation of 5G Network slicing with Machine Learning”, ITU Kaleidoscope Santa Fe, 2018.

[22] J. Moysen and L. Giupponi, “From 4G to 5G: Self-organized Network Management meets Machine Learning”, arXiv:1707.09300v1 [cs.NI] 28 July 2017.

[23] S. Ayoubi et al., Machine Learning for Cognitive Network Management, IEEE Communications Magazine, January 2018, pp.158-165.

[24] D. Lorenz et al., “SliceNet – Cognitive Slice Management Framework for Virtual Multi-Domain 5G Networks”, https://www.systor.org/2018/pdf/systor18-21.pdf, [retrieved September, 2019].

Optimization of Service Function Assignment and Shortest Path for Network Function Virtualization

Arvind Kalyan
Westview High School
San Diego CA 92129
arvindkrishnakalyan@gmail.com

***Abstract*—This paper focuses on the concept of Network Function Virtualization (NFV): the implementation of requests consisting of various service functions on servers located in data centers. This paper attempts to minimize both the cost of routing and service function assignment of requests from source to destination node on a network. This problem falls under the class of Integer Linear Programming (ILP), which is NP-Hard and cannot be solved in polynomial time. Towards developing a solution, it is proposed to split the problem into two separate optimization subproblems: shortest path routing and service function assignment. We utilize Dijkstra's Shortest Path algorithm and a Greedy method for service function assignment to propose a new heuristic algorithm that minimizes the total cost of routing and service functions assignment. The experimental results suggest that the proposed algorithm matches the optimal ILP solution within acceptable limits.**

Keywords – Network Function Virtualization; Integer Linear Programming; Dijkstra's Shortest Path Algorithm; Greedy Heuristic.

I. Introduction

In the last few years, the rise of powerful new networking technologies and big data has generated a strong need for equally powerful, versatile network services. Internet traffic is higher than it has ever been, and continues to grow in an exponential fashion. The current technologies available to respond to this increased need are inflexible, featuring rigid physical servers that host several separate packages of an operating system with its respective network functions. These functions, such as Deep Packet Inspection, Firewalls, and Content Delivery Network (CDN) Servers, are chained to exactly one operating system, dramatically reducing the efficiency of their host. Servers are forced to run at efficiencies far below their optimal rate, and as the demand for traffic increases has become more and more dynamic in nature, this current technology is simply inadequate. The concept of Network Function Visualization (NFV) has risen as a solution to these presented challenges. NFV, at its core, involves the deployment of several network functions in the form of software on high volume shared servers in data centers. Network controllers dictate the route of this flow from the source node, through the data centers, and to the destination. This concept allows for the programmability of network control, thus allowing the network to adapt flexibly to the required functions. Also, NFV greatly increases the efficiency of each of these servers, and while there remains room for improvement, the decoupling of function and operating system offers versatility with the potential to break through the current limitations. The process of implementing virtual network functions onto a single server and allocating the necessary resources to carry it out is called service function chaining (SFC). SFC creates a chain of different services to be carried out in an appropriate order, taking available resources, size, and other factors into consideration. This process may be automated to provide the fastest and most efficient execution of a set of assigned services, and this has been the focus of much of the previous work regarding the topic.

Section II delves into the contributions of this paper with respect to existing solutions. Section III establishes the mathematical model of the problem, including the constraints and objective function that are to be minimized. Section IV introduces and explains the algorithm we have devised using pseudo-code and a run-time analysis in comparison with existing solutions. The experimental results are graphed and explained in Section V, and Section VI contains the conclusion and final remarks.

II. Related Work and Paper Contribution

Previous authors have utilized an ILP based solution in order to place virtual network functions into appropriate data centers. These authors have also provided heuristics and several approximation algorithms to solve this problem. However, unlike this proposed model, few of them have taken into consideration the routing cost from source to its destination while assigning and placing network functions.

Blenk et al. [1] delved into the placement of hypervisors, which serve as a layer between a Software Defined Network (SDN) controller and networks, consisting of various necessary functions. The authors stated the importance of a ”good placement” of these hypervisors. Amaya et al. [2] explored the specifics of service chaining, creating a model that allows various orderings to be ranked and evaluated. They also derive a route from their ordering and apply various constraints to the data centers. Ghasem et al. [3] explored the details of service chaining and the concept of network function virtualization, while Addis et al. [4] focused on the benefits of optimizing the route taken by function virtualization requests. Crichigno et al. [5] considered a routing and placement scheme like the one proposed in this work, however they propose a different heuristic to solve the problem. Subsequent analysis in this paper suggests that the proposed algorithm has superior run time complexity as compared to Crichingo et al. [5]. Cohen, et al. [6] proposed a near optimal placement of virtual network functions using approximation algorithms guaranteeing a placement with theoretically proven performance. Luizelli et al [7] formulated a network function placement and chaining problem and proposed an ILP model to solve it. Moens et al. [8] proposed a formal model for virtual network function placement (VNF-P) with focus on a hybrid scenario with part of services provided by dedicated physical hardware and the rest provided using virtualized service instances. Gupta et al. [9] provided a mathematical model for the placement of VNFs which ensures the service chaining as required by the traffic flows.

This paper is unique in attempting to explore a solution toward minimizing cost by splitting the problem into two subproblems: optimization of both network service function assignment among data centers and the routing cost. Since the problem is ILP, which is NP-Hard and cannot be solved in polynomial time, we search for a solution of lower complexity. Utilizing a Greedy method and Dijkstra's shortest path algorithm [10][11], we create a heuristic algorithm in order to minimize both costs. The algorithm provides complete source to destination route for all requests along with the assignment of the network functions requested.

III. Mathematical Formulation of the Problem

A. Model

We represent the network using a graph $G = (V, E)$, where V represents the set of nodes on the graph and E represents the links that connect each node. In this set V of nodes, we define derived subset $D \subseteq V$ as the set of data centers where each network function will be virtualized. Each network function is denoted by $f \in F$. A request $r \in R$ from source node src_r to destination node dst_r contains a group of n functions $f_{r,1}, f_{r,2}, \ldots, f_{r,n} \in F_r$ to be implemented by the network. For a pair of nodes $(i,j) \in E$, $c_{i,j}$ is the predetermined cost of traversing that link, from node i to node j. $x^r_{i,j}$ is a binary variable (either 0 or 1) that denotes whether that link is traversed or not by request r. For each data center $d \in D$, W_d is the maximum capacity available on data center d to virtualize the network function. There are many types of resources that may be denoted by this variable, including storage, memory, and CPU cores. We denote the resource required to virtualize network function $f \in F$ from data center $d \in D$ as $w_{d,f}$, and the cost of virtualizing the network function on the data center is denoted by $c_{d,f}$. We define $y^r_{d,f}$ as another binary variable, specifying whether or not a function in f is virtualized or not on a specific data center d on a certain request r.

B. Objective

This problem involves routing and assignment, so it requires two functions to optimize the problem.

First, we aim to minimize the total Service Function Assignment cost across the given request. That is, the sum, across all services $f \in F$ and data centers $d \in D$, of the cost of implementing a given function multiplied by whether or not the service is implemented $(c^r_{d,f} \cdot y^r_{d,f})$.

$$\min \sum_{r \in R} \sum_{f \in F} \sum_{d \in D} c^r_{d,f} \cdot y^r_{d,f} \tag{1}$$

Next, we aim to minimize the routing cost for all requests $r \in R$, given as the sum across all pairs $(i,j) \in E$ of the product of associated cost and whether that link is traversed or not $(c_{i,j} \cdot x^r_{i,j})$.

$$\min \sum_{r \in R} \sum_{(i,j) \in E} c_{i,j} \cdot x^r_{i,j} \tag{2}$$

The overall objective function is to minimize both the service function assignment cost, as well as total routing cost.

$$\min \sum_{r \in R} \sum_{(i,j) \in E} c_{i,j} \cdot x^r_{i,j} + \min \sum_{r \in R} \sum_{f \in F} \sum_{d \in D} c^r_{d,f} \cdot y^r_{d,f} \tag{3}$$

C. Constraints

We constrain decision variable $x^r_{i,j}$ to equal 1 when the link is traversed from node i to node j in a route r and 0 when not traversed:

$$x^r_{i,j} = \begin{cases} 1 \text{ if traversed} \\ 0 \text{ else} \end{cases} \tag{4}$$

Similarly, decision variable $y^r_{d,f}$ is set to 1 when a service function f is assigned to the data center d in a route r and 0 when it is not:

$$y^r_{d,f} = \begin{cases} 1 \text{ if assigned} \\ 0 \text{ else} \end{cases} \tag{5}$$

The third constraint governs the virtualization of each network function f for a route r, allowing each to be implemented only once on a data center.

$$\sum_{d \in D} y^r_{d,f} \leq 1, \forall r \in R, f \in F \tag{6}$$

Additionally, we must ensure the balance of inflow and outflow for each node. Thus, we create a flow constraint for each node traversed in a particular route. The source node must have an outflow of 1, while the destination has an outflow of -1. Every node in between must have a net flow of 0, where the inflow equals outflow.

$$\sum_{(i,j) \in E} x^r_{i,j} - \sum_{(i,j) \in E} x^r_{j,i} = \begin{cases} 1, \ i = src_r \\ -1, \ i = dst_r \\ 0, \ \text{otherwise} \end{cases} \tag{7}$$

Each data center has a maximum ability to implement functions, and we attempt to prevent possible overuse of a data center d by adding a capacity constraint to each d. This constraint prevents each data center from virtualizing functions whose total resource requirement exceeds the given capacity of the center.

$$\sum_{f \in F} w_{d,f} \cdot y^r_{d,f} \leq W_d, \forall r \in R, d \in D \tag{8}$$

We must ensure that there is both an inflow and outflow out of an assigned data center. Therefore, we require the outflow at each data center to be greater or equal to the binary variable denoting its activation. A data center d that virtualizes a function f for route r, for example, will have a $y^r_{d,f}$ value of 1; this constraint requires its outflow to also be at least 1. A data center that is unused, meanwhile, may carry any outflow.

$$\sum_{j \in V} x^r_{i,j} \geq y^r_{d,f}, \forall i \in V, d \in D, r \in R \tag{9}$$

D. Proposed Solution

The above problem belongs to the class of ILP problems, which are NP-Hard problems that cannot be solved in polynomial time. In order to develop a heuristic solution to the problem, we propose splitting the overall problem into two separate optimization subproblems: a) find the shortest route from a source to a destination node, and b) assignment of the network functions through appropriate data center from source to destination node. This novel approach leads us to propose a heuristic that will utilize Dijkstra's shortest path algorithm and a Greedy network function assignment algorithm to solve the problem.

First, we have an assignment problem, with an objective function given by (1). We also utilize (5) governing data center use, (6) limiting service functions to be implemented just once, and (8) that applies the capacity constraint.

We can then model the remainder of the problem as a pure shortest path problem, using Dijkstra's algorithm to develop a solution. For this, we have an objective function given by (2), and constraints (4) and (7).

Finally, we can tie the two problems together using (9), giving us a solution to the overall problem.

IV. Algorithm for Modified Routing and Greedy Assignment

Based on the above problem split, we propose an algorithm using Dijkstra's shortest path and a Greedy heuristic algorithm to assign network functions to solve the problem.

A. Proposed Modified Dijkstra's Algorithm with Greedy Service Function Assignment

We propose a heuristic algorithm that may be utilized on networks. The algorithm requires an input of a graph G with vertices V and edges E. In addition, we specify the costs of implementing a service on each data center ($c^r_{d,f}$) and the cost of routing services from all nodes i to j ($c_{i,j}$). Finally, we set the capacity of each data center (W_d) and the resources taken by each network function on a data center ($w_{d,f}$). The algorithm returns values for $x^r_{i,j}$ which dictate the route taken by the network functions, as well as $y^r_{d,f}$, which denotes the data centers that are assigned to implement each network function. The algorithm also assumes the use of an adjacency matrix to implement the network graph and allow us to easily find the neighbors of a node in the graph through lookup.

We begin by going through each of the requests in a set R. Using the source and the destination nodes of this request, we use Dijkstra's algorithm to discover the shortest path for the request, returning the list of nodes that make up the path as SP_r. From here, we create two separate lists: the first, PDC_r, will store data centers found on the shortest path given by Dijkstra's, while the second list, NDC_r, will store data centers that are neighbors along the shortest path. The shortest path is iterated through and data centers found on the path are added to the first list. Using an adjacency matrix, we define a function $neighbors(n)$ to return all nodes that are linked to node n. Any data centers found among these neighbors are added to NDC_r.

Next, we provide network function assignment for the functions requested by r. For each, we use a Greedy heuristic algorithm in order to select a data center from all those detected by the previous scan, using the list comprised of the union of PDC_r and NDC_r. This is done by comparing the different costs of $c^r_{d,f}$ for each data center and ensuring that the necessary resources are available for implementation. Once the lowest cost data center d has been located, we set the corresponding value of $y^r_{d,f}$ to equal 1, signifying the use of that data center for the function. We then update the available resources of the data center by reducing its capacity W_d by $w_{d,f}$. If the data center is on the path, the shortest path SP_r does not need to be updated. However, if the data center is found from the neighbors list, we insert a one-link detour to that node into SP_r. This process is repeated for every function in the request. This is the key modification to the shortest path algorithm, hence we call it the modified Dijkstra's algorithm.

By the process described above, we expect a majority of the network functions requested to have been assigned and virtualized at an appropriate data center. However, there is a possibility that the request still has unassigned functions, in which case we propose the use of a breadth-first search (BFS) to locate a capable data center. Once a data center is located by BFS, if it contains the necessary resources, we assign to it the remaining functions and update its capacity, $y^r_{d,f}$, and SP_r accordingly. This is the worst-case scenario of the algorithm.

Once all functions have been assigned, we use SP_r to update the values of $x^r_{i,j}$ for the route, setting the variable to 1 when the link is traversed. After this process has been repeated for every request, the algorithm return $x^r_{i,j}$ and $y^r_{d,k}$, for all $r \in R$.

The proposed heuristic algorithm utilizes Dijkstra's shortest path algorithm and a Greedy method in order to calculate an optimal route for a set of requests R. The first component of the algorithm uses the shortest path algorithm to find a path SP_r from the source node src_r to the destination node dst_r. The second uses a Greedy method to find an optimal data center $d \in D$ along the previously discovered path for each service function $f \in F_r$. The Greedy method runs with a linear complexity, while Dijkstra's algorithm implemented with a binary heap will run in logarithmic time. Additionally, the neighbor search is implemented as a simple table lookup due to the adjacency matrix implementation of the network

Algorithm 1 Modified Dijkstra's Algorithm with Greedy Service Function Assignment

```
  Input: $G(V, E); c_{i,j} \forall (i, j) \in E$ ; $W_d \forall d \in D$ ; $w_{d,f}, c_{d,f} \forall d \in D$
  Output: $y^r_{d,f}, x^r_{i,j} \forall r, d, f$
  for all $r \in R$ do
    $src_r$ = source node of request
    $dst_r$ = destination node of request
    $SP_r = Dijkstra(src_r, dst_r)$
    initialize $PDC_r$ (set of data centers found on shortest path $SP_r$) as $\phi$
    initialize $NDC_r$ (set of data centers found among path neighbors) as $\phi$
    for node $n \in SP_r$ do
      if $n \in D$ then
        $PDC_r = PDC_r \cup n$
      end if
      for node $m \in neighbors(n)$ do
        if $m \in D$ then
          $NDC_r = NDC_r \cup m$
        end if
      end for
    end for
    for all $f \in F_r$ do
      Select $d \in PDC_r \cup NDC_r$, the data center that implements function $f$ at lowest cost, contains necessary resources (Greedy Assignment)
      set $y^r_{d,f} = 1$
      update resources for $d$ appropriately
      if $d \in NDC_r$ then
        update $SP_r$, insert $d$
      end if
    end for
    if any $f \in F_r$ remains unimplemented then
      run breadth-first search (BFS) starting from the most connected node $\in SP_r$
      if data center $d$ is found by BFS then
        implement $f \in F_r$ (provided $d$ has necessary resources available)
        update $y^r_{d,f}$
        insert path to $d$ into $SP_r$
      end if
    end if
    update all $x^r_{i,j}$ using $SP_r$
  end for
  return $x^r_{i,j}, y^r_{d,f} \forall r, d, f$
```

and is $O(|V|)$. Therefore, we may conclude that the total algorithm is dominated by the first stage Dijkstra's algorithm run-time complexity. The worst case running time of our implementation of Dijkstra's algorithm with a binary heap is given by $O(|E|log|V|)$, and we call the Dijkstra's once for each request R in a scenario. Therefore, our total run time for the proposed heuristic algorithm is $O(|R||E|log|V|)$. Notably, as a comparison, the algorithm of Crichigno et al. [5] utilizes Dijkstra's algorithm twice, resulting in a worst-case run-time of $O(|R||D||E|log|V|)$, where D is the total number of data-centers contained in the topology. Thus the proposed heuristic algorithm is able to run at a faster run time by a factor of $|D|$ as compared to the algorithm proposed by Crichigno et al. [5].

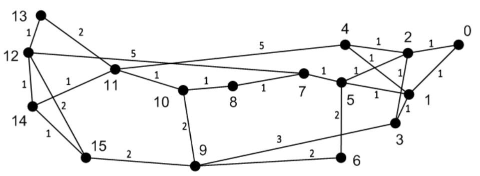
V. Experimental Results

A. Simulation Setup

We tested the algorithm using two different network topologies. The simulations were carried out using a Python script running on a 2.70 GHz. The first is the NSF-Net graph, forming an approximate outline of the United States of America and placing nodes at major cities, including Chicago, New York, Atlanta, and Los Angeles. The second is a fully connected hexagon graph with a central node connecting the 6 vertices. The graphs of the same topologies differ from each other in network configuration and data center specifications. For example, Figure 1 features an NSF-Net graph, but with various link costs. Figure 2 features the same topology, however, each of its links has a cost of 1, and the graph denotes different data centers. Each of these data centers implements functions at a different price than Figure 1. Similarly, Figures 3-5 correspond to a fully connected hexagon graph with one data center B (in Figure 3), data centers B and C (in Figure 4) and data centers B, C and T (in Figure 5).

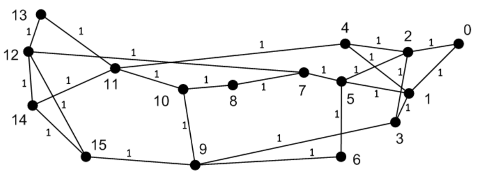
For each topology, we generated 4 requests $r \in R$ consisting of a group of functions $f \in F$ to be virtualized. Each one of these requests had a different source and destination node found across the topology that the route must traverse between. These routes are designated by Route 1-4 in the results (Figures 6 to 10).

For every request within each topology, we only changed the source and the destination nodes (src_r, dst_r) of the route with respect to the various data centers. These costs of implementing the network function on the data centers were randomly distributed among a set of target values to provide a more reliable testing set. The evaluation metrics for the results are as follows: for each test case using the selected topology configuration (from Figures 1-5), the simulation compares the costs for the routes (Routes 1-4) as given by the proposed heuristic algorithm with that of the optimal ILP solution. Figures 6-10 depict the comparison of the routing cost, assignment cost and the overall cost, given by the sum of the routing and assignment cost, for the algorithmic and optimal ILP solution. The next section discusses the results in detail, in particular the gap between the proposed algorithm and the optimal ILP solution.



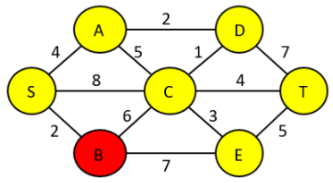
	Data Center 14	Data Center 10	Data Center 5	Data Center 0
Function1	5	1	10	14
Function2	9	7	6	10
Function3	3	14	9	3

Fig. 1. NSF 1 Network Graph



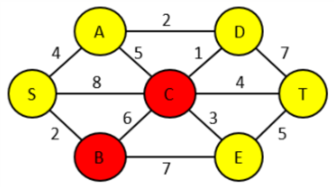
	Data Center 12	Data Center 8	Data Center 7	Data Center 5
Function1	12	8	7	5
Function2	6	5	1	15
Function3	6	10	4	7
Function4	6	3	6	8

Fig. 2. NSF 2 Network Graph



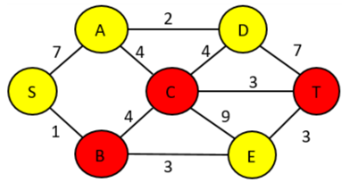
	Data Center B
Function1	3
Function2	6
Function3	1

Fig. 3. HEXAGON 1 Network Graph



	Data Center B	Data Center C
function1	3	6
function2	2	1
function3	4	4

Fig. 4. HEXAGON 2 Network Graph



	Data Center B	Data Center C	Data Center T
Function1	12	12	20
Function2	30	30	20
Function3	60	60	100
Function4	50	50	45

Fig. 5. HEXAGON 3 Network Graph

B. Simulation Results and Discussion

The first test involved the NSF-Net with routing costs randomly generated from 1 through 5. Data centers were placed at Nodes 12, 8, 7, and 5, with implementation costs given by Figure 1. The data centers were chosen to emulate central locations in the continental United States, allowing direct access to a larger set of nodes on the graph. The algorithm was able to compute a solution that equals the optimal total cost in Route 1, and was within 23% over the optimal cost in the remaining 3 routes.

In the second test using the NSF-Net, we set each route link to 1 and increased the weights of the data center cost, simulating an assignment cost-dominated scenario. Data centers were moved to 14, 10, 5, and 0 as seen by Figure 2. The algorithm matched the optimal net cost in two of the routes and was able to come close to reaching the optimal overall cost, within 22%.

The third test involved the fully connected hexagonal network, as shown in Figure 3. We designated one of the nodes on the periphery, B, as the data center and gave the assignment and routing costs an approximately equal weight. Since we assigned only 1 data center, the requests were able to meet the optimal assignment cost on each route. The overall algorithmic cost was an acceptable range of 20% within optimal cost.

The fourth test added another data center to node C, as shown in Figure 4. These tests were designed to add variation to the routing costs, as the request may route through multiple centers, incurring a different cost each time. Overall, as shown in Figure 9, the results of the algorithm come within 24% of the optimal case in each scenario.

The final test incorporated another data center at node T and increased the cost of assigning each function on a node, as shown in Figure 5. In addition, the cost of implementation for each function was increased. In this trial, as shown in Figure 10, the algorithm was able to come the closest to the optimal ILP solution, within 2.9%.

Overall, we find that the proposed heuristic algorithm produces solutions over the five different configurations that match the optimal ILP solutions within an acceptable range, while providing an efficient practical solution.

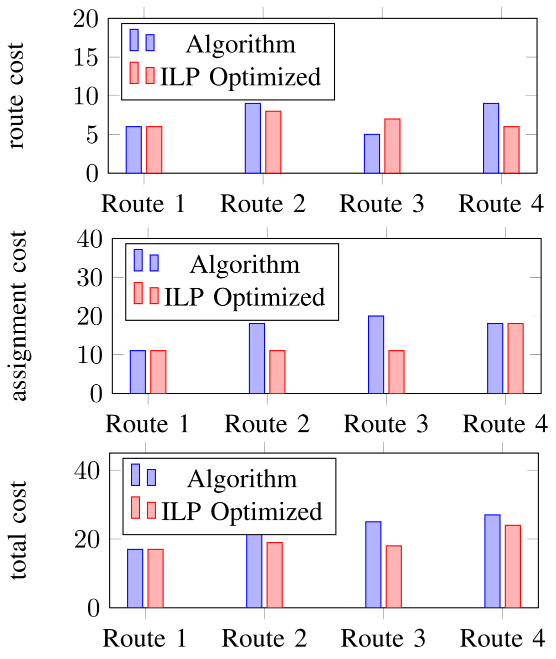


Fig. 6. NSF-Net 1 route, assignment and total cost comparison for Algorithm vs ILP Optimized

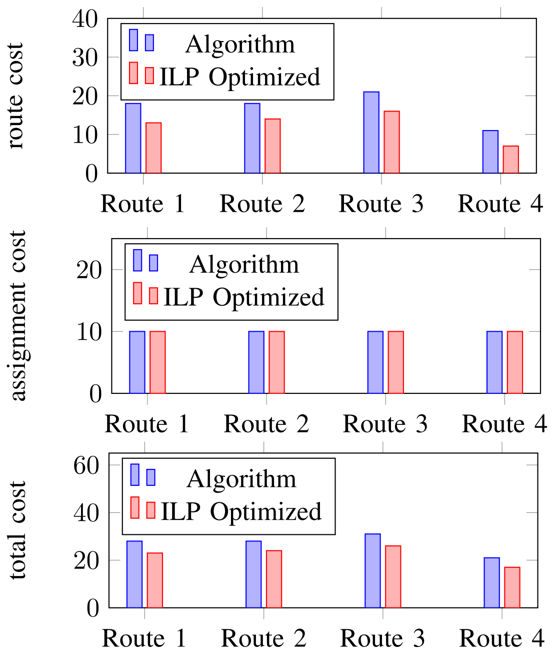


Fig. 8. HEXAGON 1 route, assignment and total cost comparison for Algorithm vs ILP Optimized

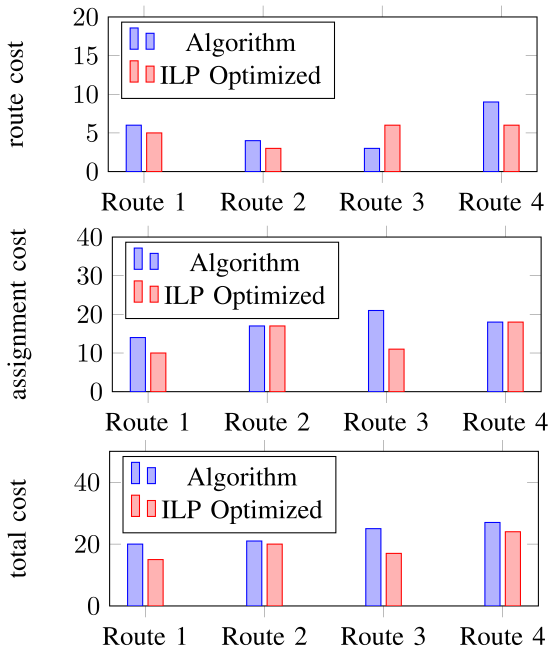


Fig. 7. NSF-Net 2 route, assignment and total cost comparison for Algorithm vs ILP Optimized

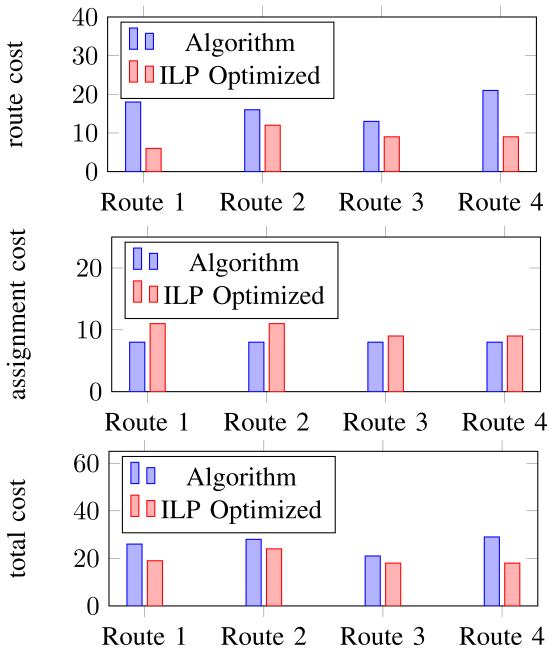


Fig. 9. HEXAGON 2 route, assignment and total cost comparison for Algorithm vs ILP Optimized

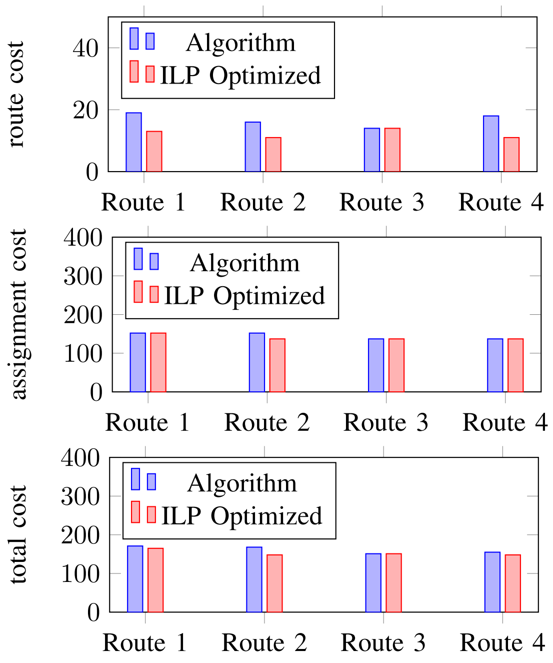


Fig. 10. HEXAGON 3 route, assignment and total cost comparison for Algorithm vs ILP Optimized

VI. Conclusions

This paper poses an optimization problem of service function assignment and shortest path for network function virtualization and proposes a solution that splits the problem into two separate optimization subproblems: shortest path and service function assignment. The algorithm uses the combination of a modified Dijkstra's shortest path algorithm and Greedy heuristic algorithm for network function assignment. Some of the key challenges lie in thoroughly validating the algorithm performance on larger networks. In future work, we propose to expand the scope of testing to validate the algorithm against larger network configurations, using more complex optimization packages such as Gurobi, CPLEX, etc.

References

[1] A. Blenk, A. Basta, W. Kellerer, and J. Zerwas, "Pairing SDN with Network Virtualization: The Network Hypervisor Placement Problem", *IEEE NFV-SDN*, November 2015, pp. 198-204.
[2] D. Amaya, Y. Sumi, S. Homma, T. Okugawa, and T. Tachibana, "VNF Placement with Optimization Problem Based on Data Throughput for Service Chaining", pp. 1-3. *IEEE CloudNet*, March 2018.
[3] M. Ghasem and L. Zhiquan, "Optimal Network Function Virtualization and Service Function Chaining: A Survey", *Chinese Journal of Electronics, Vol.27, No.4*, July 2018, pp. 704-717.
[4] D. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual Network Functions Placement and Routing Optimization", *IEEE CloudNet*, March 2015, pp. 171-177.
[5] J. Crichigno, D. Oliveira, M. Pourvali, N. Ghani, and D. Torres, "A Routing and Placement Scheme for Network Function Virtualization", *IEEE TSP*, July 2017, pp. 26-31.
[6] R. Cohen, L. Lewin-Eytan, J. Naor, and D. Raz, "Near optimal placement of virtual network functions", *IEEE INFOCOM*, April 2015, pp. 1346-1354.
[7] M.C. Luizelli, L.R. Bayes, L.S. Buriol, M.P. Barcellos, and L.P. Gaspary, "Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions", *IFIP/IEEE International Symposium on Integrated Network Management*, May 2015, pp. 98-106.
[8] H. Moens, and F. De Turck, "VNF-P: A model for efficient placement of virtualized network functions", *10th International Conference on Network and Service Management (CNSM) and Workshop*, 2014, pp. 418-423.
[9] A. Gupta, M.F. Habib, P. Chowdhury, M. Tornatone, and B. Mukherjee, "Joint Virtual Network Function Placement and Routing of Traffic in Operator Networks", *Netsoft*, 2015, pp. 1-5.
[10] S. Skiena, The Algorithm Design Manual, *Springer-Verlag*, 1998.
[11] Coursera.com Algorithms Specialization: Graph Search, Shortest Paths, and Data Structures, *[Online] Available: coursera.com, [retrieved: 08, 2019].*

Risk Based Web Authentication Using Bluetooth Devices

Asad Ali
Digital Identity and Security
Thales
Austin, USA
asad.ali@thalesgroup.com

Darmawan Suwirya
IBM Technology Services
IBM
Austin, USA
dsuwirya@ibm.com

Abstract - **There has always been a growing need for online access solutions that use strong authentication methods without encumbering the user. Current solutions that are easy to use and deploy offer weak security, while those that offer strong security are hard to use and deploy. The solution architecture presented in this paper allows users to continue using their existing authentication method, which in itself may be weak, but can be made stronger by augmenting it with standard Bluetooth devices that are part of the user's everyday work environment. These Bluetooth devices seamlessly offer additional factors of authentication without any explicit user intervention. This approach creates opportunities for adaptive, continuous and risk-based authentication where proximity of known Bluetooth devices is an input to risk management policies.**

Keywords - bluetooth devices; user convenience; adaptive online authentication; browser extension.

I. INTRODUCTION

"On the Internet, nobody knows you're a dog" [1] is an adage and meme about identity verification on the Internet, or rather lack thereof. It began as a cartoon caption by Peter Steiner and was published by The New Yorker on July 5, 1993. Ironically, a quarter of a century later, we still seem to be battling the same challenges of identity verification for Web applications, though on a different scale. Despite the collective desire of the security industry to get rid of knowledge-based authentication, passwords still remain the primary method of proving a person's identity on the Internet. This absence of progress has not been for lack of trying. There have been several approaches that augment the single authentication factor based on what-you-know, with a second factor based on what-you-have, or even a third one based on what-you-are. All these multi-factor solutions demand a dedicated hardware token. While these hardware tokens enhance the level of assurance associated with the authentication process, they also add complexity, thereby reducing usability, especially for millennials who are born and raised in the social media era and shun products that lack crisp user experience. For this reason, multi-factor authentication techniques are generally confined to controlled settings, such as enterprise and office environments where the possession of dedicated tokens can be made mandatory. However, the need for strong authentication is universal and is equally applicable in everyday Internet use outside these controlled environments.

This paper describes a technique of authenticating users based on their typical surroundings and work environment, by checking the presence of known or expected Bluetooth devices in the proximity of the user. The list of what devices to look for can either be explicitly specified by the user, or it can be implicitly learned by the system through previous successful logins.

The rest of the paper is organized as follows. Section II provides a background to the authentication challenge, and explains why the industry is still looking for a solution. Section III describes the detailed design, architecture and implementation of one such solution. Section IV offers an analysis of this proposed architecture in terms of security and user experience. We then offer our conclusions in Section V.

II. BACKGROUND

When using a Web browser to authenticate to an online service provider, a user typically enters his/her credentials into the Web page. In most cases, these login credentials are username and password. This single (what-you-know) factor is very easy to deploy, but universally considered weak. To improve the security of authentication, a second factor is added. This factor can be a smart card connected to the computer, a One-Time Password (OTP) generated on a dedicated device, a numeric code sent to the user's phone, etc. In all cases, the user has possession of this hardware token (what-you-have) that constitutes the second factor.

This second-factor hardware token has to be issued or configured by the service provider, thereby adding to the cost and complexity of deployment. The user cannot unilaterally select his/her own hardware tokens, such as one or more of the following devices: mobile phone, wireless keyboard, wireless mouse, wearable devices, wireless speaker, or any other Bluetooth enabled device and then use them as second-factor when authenticating online. These devices have to be issued by the Identity Provider. The reason for this restriction lies in the underlying authentication technology that demands a tight coupling of cryptographic algorithm between the client device and the backend authentication server. For example, the use of OTP or Public Key Infrastructure (PKI) requires both the device and the server to be updated together. How do the secret algorithm and the counter get exchanged between the device and the server? Will authentication flow use OATH Challenge-Response Algorithm (OCRA) [2], a challenge-response flavor of OTP? Similar complexities arise when using public key cryptography protocols.

The Fast Identity Online (FIDO) standard [3] addresses some of these issues by relaxing the tight binding between edge devices and backend authentication servers. However, this adds a new constraint of only using devices that have FIDO stack built into them. The list of such devices, though slowly growing, is still a tiny fraction of the otherwise abundantly available Bluetooth devices.

There are additional usability concerns with existing approaches. For example, one-time numeric codes pushed to user's phone assume the phone is on the network and then also require manual action for the user to type the codes into a browser window. Biometric systems need specialized scanners and backend modifications. FIDO Universal Two Factor (U2F) [3] tokens require an action on the part of the user, such as pushing a button manually on the device.

Our solution enables an adaptive authentication framework so that the user's login is controlled by the presence of Bluetooth devices that can be preselected by the user. This authentication solution works without any explicit user action or interaction with hardware tokens, thereby addressing the following shortcomings identified in other existing approaches:

1. Devices for two-factor authentication have to be customized. They are also selected at the discretion of the authentication server or identity provider.
2. The deployment of these two-factor authentication devices is costly for merchants. Most often than not, this cost is indirectly passed to the user.
3. Users are required to carry these devices, which negatively impacts user experience.
4. The turn-around time to replace a device is long. User cannot select a new device unilaterally.

III. PROPOSED SOLUTION

This section describes the philosophy, architecture and implementation of the proposed solution.

A. *Approach Overview*

The solution presented in this paper can enable an adaptive authentication framework so that the user's login is controlled by the presence of Bluetooth or Bluetooth Low Energy (BLE) devices that are explicitly or silently selected at user's discretion, and not dictated by the Identity Provider. In addition to the primary authentication method (e.g., username and password), the presence of such previously identified Bluetooth devices is also checked. Virtually any Bluetooth device can be used based on user preference. Some examples include a user's mobile phone, wearable device (such as smart watch or fitness band), wireless keyboard, wireless mouse, wireless headphone, wireless speaker, smart pen, modern workout bench, weight scale, etc.

Risk based authentication already relies on data signals, such as network information, to confirm user's location, typing patterns to confirm user behavior, etc. Similarly, proximity of known Bluetooth devices can also indicate a trusted environment and hence allow for a more frictionless user login experience.

The challenge is to build this data signal handling into Web applications and enforce them through Web agents running on the user's computer. Web browsers can check the presence of Bluetooth devices in proximity by using Web Bluetooth Application Programming Interface (API) [5] or browser extension. One example of such Web browser extension is SConnect [4] that allows Web applications to seamlessly connect to security devices using a range of communication protocols, including Bluetooth.

B. *High Level Design*

The solution is largely based on common existing Web authentication system design, with slight enhancements added on both client and server sides in order to support the proximity of known Bluetooth devices as second factor authenticators. Figure 1 illustrates these components.

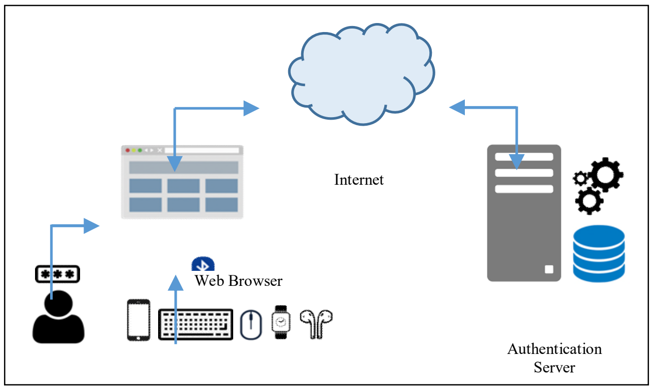


Figure 1. High-level block diagram of solution components.

These enhancements are simple to implement, yet powerful in their impact on delivering a seamless and secure authentication experience.

1) Client Modifications: On the client side, modification is done to the authentication page where the user would normally enter his/her username and password. The login page is augmented through a JavaScript code that uses Web Bluetooth API to silently scan and retrieve the list of Bluetooth devices found in proximity of the user's primary access device, for example his/her Personal Computer (PC). This API is currently only available in some modern browser versions. However, in the cases where Web Bluetooth API is not yet available on the user's browser, browser extension can be used to poly-fill the required functionality. This is standard practice in Web application development to compensate for missing browser capability by using an external library or component.

A list of the Bluetooth devices which are found in proximity of user, some minimal information about these devices, and the credentials entered by the user are then sent to the backend authentication server. The device information may contain its name, type, signal strength, etc. The user information can be as simple as user identifier, or could include actual credentials such as password, OTP or some other form of challenge-response.

Figure 2 shows a sample of code snippet that can be used for scanning Bluetooth devices in proximity using Web Bluetooth API specification. Once inserted into the login page, this script is expected to trigger itself when page is loaded. It makes the necessary API calls to instruct the Web browser to start scanning for Bluetooth devices. The script collects some minimal but useful information from every Bluetooth device detected in proximity of the device on which this Web browser is running. It then constructs the authentication request payload using this information. When the user has completed the login process, the login page along with this script is unloaded. Just before this unload step, the script can also do necessary clean up and instruct the Web browser to stop scanning for Bluetooth devices.

```
let btScan;
let btDevices = [];

// scan for BLE devices on page load
window.onload = () => {

 let bt = navigator.bluetooth;
 let opts = {acceptAllAdvertisements: true};

 btScan = bt.requestLEScan(opts).then(() => {

  let eventName = ‘advertisementreceived’;
  bt.addEventListener(eventName, event => {
   let device = event.device;
   btDevices.push({
    ‘name’: device.name,
    ‘id’: device.id,
    ‘serviceIDs’: event.uuids,
    ‘appearance’: event.appearance,
    ‘txPower’: event.txPower,
    ‘rssi’: event.rssi,
    ‘connected’: device.gatt.connected
   });
  });
 });
};

// stop scanning for BLE devices on page unload
window.onunload = () => {
 btScan.stop();
```

Figure 2. Code snippet for scanning Bluetooth devices.

A sample of the authentication request payload could look like the data blob shown in Figure 3. The payload would still contain the user credentials as before, but in addition it will now also contain information regarding Bluetooth devices which are found in proximity. Examples of such information are device identifier (ID), device name, service IDs, appearance types, transmission power class, signal strength, and connection status. Individually each piece of information is minimal, but when used collectively, they are quite useful and unique enough to distinguish one device from another. Upon receiving authentication request payload, the backend authentication server then uses all this information to determine whether user is really who he claims to be.

A better alternative design could be to direct the client to look only for a predetermined collection of Bluetooth devices, guided through a set of high-level filter criteria. These criteria are determined by the server and passed to the client. The client then returns information related to devices that match these criteria. This design has several benefits, such as faster scanning time, collection of more relevant device information, and thereby smaller authentication request payload sent to the server. Furthermore, this approach is equally secure since it significantly reduces the risk of exposing policy or user-device mapping information.

```
GET /authenticate HTTP/1.1
{
  “username”: “bob”, “password”: “1234”,
  “devices”: [

    {“id”: “00:11:22:33:FF:EE”,
     “serviceIDs”: “0x2a01”,
     “appearance”: “0x10”,
     “txPower”: “4dBm”,
     “rssi”: “-40dBm”,
     “connected”: “false”},

    {“id”: “00:12:23:34:AB:CD”,
     “serviceIDs”: “0x3440”,
     “appearance”: “0x20”,
     “txPower”: “0dBm”,
     “rssi”: “-52dBm”,
     “connected”: “true”}
  ]
}
```

Figure 3. Authentication payload sent by browser to server.

2) Server Modifications: On the server side, modification is done to the back-end authentication logic and infrastructure. The server needs to be able to accept additional payload in the authentication request. It then processes the information contained in this payload, including the information about Bluetooth devices. In addition to matching the username/password against user database, the authentication logic now also does verification of Bluetooth device information against the user/device database using policies that control how this new device information is expected to contribute to the final authentication decision. For example, one policy may require the presence of only one known device in the proximity of the user, while another policy may mandate that at least two such devices be present.

This processing not only yields a decision to grant or deny access to requested resource, but perhaps more importantly it also feeds the outcome into machine learning infrastructure that will make future authentication decisions smarter and more accurate.

C. Environment Initialization

Depending on the expected security level and type of authentication flow that is offered, the user may be requested to perform a set of minimal setups prior to using the system. Performing these optional setups has benefits. It increases the overall security of device selection and identification process. It also provides a fine grain control for the users to choose which devices they want to explicitly trust and use for this purpose. This is done through two sequences: device pairing and device binding. The user is free to choose either of these sequences or both of them, if so desired.

1) Device Pairing: As the name suggests, pairing means introducing two Bluetooth capable devices to each other.

After the pairing handshake, each device knows about the other. For example, pairing a Bluetooth headset with your smartphone or paring the smartphone with the infotainment system of your car. In the context of this paper user pairs one or more Bluetooth devices to his computer. This process is either done through tools available from the computer OS, or can be driven through the Web application itself with the help of either Web Bluetooth API or browser extension. This step is optional, but preferred as it increases trust level of the paired device, hence increasing the security of authentication. Only connection specific information is retrieved from the device. Device application related data such as workout history of a fitness band, are neither needed nor fetched during pairing. This can allay concerns about security and privacy of user data when such devices are used for two-factor authentication.

2) Device Binding: By "device binding" we mean the mapping of a user account to certain Bluetooth devices. This mapping is maintained on the authentication server. There are two ways authentication server can construct this map:

1. The user is asked to explicitly select devices.
2. The authentication server implicitly builds this list of devices over time, by monitoring available devices at the time of multiple successful logins.

In the first case, once the user has logged in to online service provider using an existing authentication method, the Web application will use the Web Bluetooth API or browser extension to look up for all Bluetooth devices found in the proximity and (optionally) paired to user's computer. The list of these available devices is shown to user in the Web browser. The user can now select one or more devices he wants to bind to his account. This information is stored in authentication server database and all subsequent access to user account will be granted only when the selected devices are also present at the time of authentication.

For risk-based authentication systems where explicit user involvement is not desired, or for user convenience, the system can decide not to expose user to device selection steps and can make device binding process transparent. The authentication server automatically builds the knowledge of which devices are relevant and associated to a user based on the device data silently collected over time, through multiple successful user authentications. For example, the user will never explicitly say "I want to associate my mobile phone and wireless mouse with my account". However, if on every login attempt the server notices that these two devices are present, it will implicitly bind these to the user account. After a certain number of such logins, as determined by the policy, device proximity factor can be enabled. At this point, the user can enter his login user identifier (e.g., email address) and authentication server will allow access without asking for any login credential, provided the same two devices are detected in the proximity.

D. *Authentication Flow*

This section explains detailed flows of authentication after the one-time environment initialization has been completed.

1) Adaptive Silent Authentication: In this flow, all required proximity devices are present, so the user can login simply by providing his/her user ID. No additional credentials such as password or OTP are required. This flow is illustrated in Figure 4, and outlines the steps taken by a user as well as authentication application from the initial intent by user to login to a Web resource, to the final action to grant access to the requested resource.

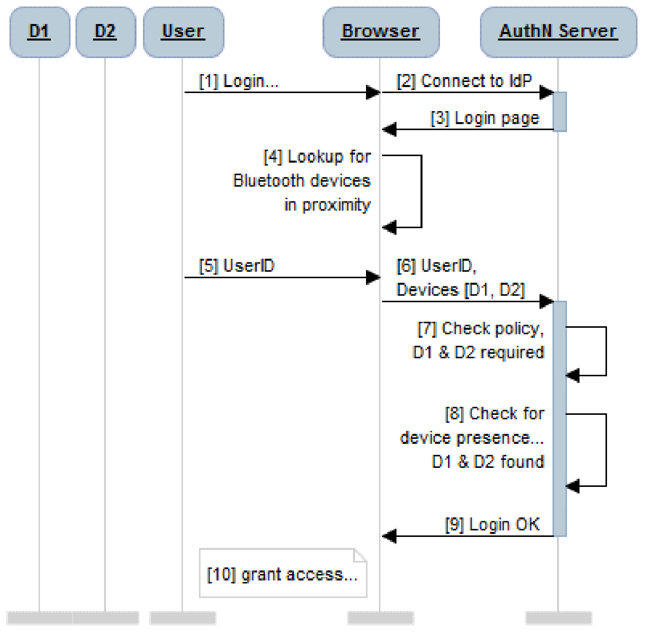

D1
D2
User
Browser
AuthN Server
[1] Login...
[2] Connect to IdP
[3] Login page
[4] Lookup for Bluetooth devices in proximity
[5] UserID
[6] UserID, Devices [D1, D2]
[7] Check policy, D1 & D2 required
[8] Check for device presence... D1 & D2 found
[9] Login OK
[10] grant access...

Figure 4. Adaptive authentication flow not requiring any user credential.

As shown in Figure 4, the user has two devices, D1 and D2, in close proximity during the login attempt. The numbered steps and message exchanges (1 to 10) are explained below:

1. The user initiates the login process by opening a Web browser from the access device such as a PC.
2. The Web browser client connects to the login portal of the authentication server: the Identity Provider.
3. The Authentication server sends back the login page to be rendered in the browser, asking the user to enter his *UserID*, such as username. This login page content also includes logic to scan for Bluetooth devices in the proximity of the user's computer.
4. This logic to scan for Bluetooth devices can be implemented by some JavaScript code that is automatically triggered by the Web browser when the login page is loaded. This code builds a list of devices it finds in the proximity of the user's computer. This is done either through direct use of the Web Bluetooth API, or through a browser

extension that invokes operating system level APIs to search for Bluetooth devices.

5. In parallel, the user enters his/her *UserID* (not the login credential, such as password) as required by the Web application. This information is entered within the same login page.
6. The Web browser sends this *UserID* to the authentication server, along with the list of Bluetooth devices found in proximity; in this case D1 and D2.
7. The authentication server looks up the login policy for this user, identified by *UserID*. It notices that the user had previously associated two Bluetooth devices with his/her account: a keyboard (D1) and a smartphone (D2).
8. The authentication server analyses the list of Bluetooth devices found in the proximity received earlier and verifies that both devices D1 and D2 are present.
9. The authentication server then determines that, since both devices are present, no further authentication credentials are required. It sends back a "Login OK" message.
10. User is granted access to the requested resource.

The user experience of this silent login flow is similar to password caching by the browser, or authentication server storing a cookie in the browser and automatically granting access to the user once that cookie is presented. However, the proposed Bluetooth device-proximity based logic offers a stronger assurance level since it relies on multi-factor authentication. In addition, it allows Web applications to create different policies to access resources of varying security levels, or when not all the expected devices are found in the vicinity of the access device.

2) Adaptive Step-up Authentication: An alternate flow can be when devices found in proximity of user are different from the expected list, thereby forcing the authentication server to ask for additional credential from the user. Depending upon the policy set by administrator of the resource being accessed, this credential could be a simple password, an OTP or even an elaborate PKI based challenge response through a dedicated security token. This step-up authentication flow is also triggered when no device is found in the vicinity.

3) Continuous Authentication: The Web application can optionally also perform continuous monitoring of the Bluetooth devices detected during initial login to make sure they remain in the proximity of the PC. In case they are removed, the application can be put in "stand-by" mode where further interaction is disallowed, until the missing device returns. Such continuous monitoring is possible through a browser extension. The user can now be logged in seamlessly without having to re-enter his primary credential. As an example of such continuous authentication, a user's Web session can be automatically put in "lock" mode when he/she moves away from his/her computer with device D2 (e.g. smartphone) in his pocket, and then restored when he returns to his/her computer.

IV. ANALYSIS OF PROPOSED SOLUTION

The design and architecture presented in this paper is a good compromise between security and usability. It is hard to find solutions that deliver on both. Most often, authentication solutions that are usable are not secure, while those that offer strong security generally do it at the expense of usability. Similar approach has been promoted in other reports [6] as well, further validating the value of using Bluetooth devices for authentication.

A. User Experience

Usability and user experience (UX) are gaining prominence in all security related products. This is not just for aesthetics or to merely appease users. Extremely robust and secure solutions that are not easy to use, end up degrading overall system security since users stop using them, and instead may find alternative options. The proposal identified in this paper is designed to improve UX without compromising security. It is modeled after "recognize" rather than ask for user credentials. The intent is to reduce user interactions with the authentication system. If a user connects from an office network, using his laptop, during office hours, and has his two previously identified Bluetooth device in close proximity, some application may consider this as sufficient authentication. Why bother the user for any login credential in this scenario? Some critical applications may additionally ask for only a single factor, like password. Still other highly sensitive applications could demand a higher assurance level of authentication like OTP or PKI based challenge-response from a dedicated hardware token. But this choice is based on the sensitivity of the Web application or the value of resource being accessed. In all cases, the surrounding environment acts as a silent "second-factor" to authenticate the user, thereby reducing the friction of login.

B. Security

As evident through numerous data breaches [7], attackers can easily compromise the what-you-know factor of a user's login credential such as password. If access to protected resources is controlled by this credential alone, an attacker can have full access to the resource. The promise of two-factor authentication (2FA) is to prevent attackers from having this free access using stolen or shared credentials. However, as mentioned earlier, the deployment and use of such dedicated tokens can be cumbersome for service providers and users alike. The silent use of Bluetooth devices serves the same purpose, but with far less overhead. As an attacker sends user's credential to the server, along with any Bluetooth devices in the proximity (if any), the server validates this credential and then matches the list of Bluetooth devices with the devices either explicitly bound to the account by user, or implicitly bound by the server based on prior repeated successful authentications. Since the attacker does not have the specified configuration of devices, the login request is denied.

C. Policy

The proposed solution gives Web applications flexibility in deciding what authentication method is needed for which scenario. Similarly, the user is free to bind any number of devices to his/her account. In case multiple devices are bound to an account, the user or authentication server can specify whether all or a specific number of devices are mandatory for authentication. For example, a user binds his/her phone, his/her watch, and keyboard to his/her account, and specifies that any two of these three devices are needed for authentication. This rule is then enforced by the server when the user logs in.

Multiple valid login locations can be enabled with different configurations of known devices. In this way, a user may have a “work” or “home” configuration of specific Bluetooth devices that need to be present when logging to the account from these predefined locations.

D. Advantages

This adaptive and risk-based authentication using the proximity of user selected devices offers the simplicity of a single-factor knowledge-based authentication, but with the added security of multi-factor authentication. It also enables enterprise administrators to calibrate adaptive authentication with a range of policy options. Users will see these benefits:

1. User can continue to use the existing authentication method, e.g., password.
2. The second factor is based on devices picked by the user, not the device enforced by service provider.
3. Use of second factor is transparent, requiring little interaction or learning, especially when device use is automatically detected by server.
4. User can “walk away” from the computer and Web application is automatically put in stand-by mode.

Authentication servers will see these benefits:

1. Service provider can enforce strong multi-factor authentication at a very low cost of deployment.
2. There is no need to distribute dedicated 2FA hardware tokens to user. The user can use any existing Bluetooth token.
3. Service provider can offer adaptive authentication policies, based on resource being accessed.
4. Easy adoption by users means increased user base.

A key aspect of the approach in this paper is that Bluetooth devices can be accessed through a standard Web browser. There is no need to install a thick client on user’s computer. Any Bluetooth device can be used, regardless of what application level software stack it supports. The only requirement is that the device advertises itself as a Bluetooth device. This approach is different from FIDO U2F devices, which rely on FIDO protocol stack in them. In our approach, any Bluetooth device of user’s choice can be turned into a two-factor authenticator hardware token. Examples of such devices are: mobile phone, wearable devices like watch and fitness band, wireless keyboard, wireless mouse, wireless headphone, wireless speaker, smart pen, modern workout bench, weight scale, etc. The list is quite long, further validating the flexibility offered by this approach.

V. CONCLUSIONS

In the current era of digital transformation, all access control relies on some form of user authentication. This makes strong and context-based authentication an integral part of any system that protects resources and only grants access to authorized users. As the use of Internet and cloud-based services evolves, so does the expectation of “proper” authentication. Users now demand tailor made and fine-grained solutions that address their needs; nothing more, nothing less. They expect similar customization from authentication systems. In order to respond to this trend, identity providers will have to offer adaptive and risk-based authentication models. The goals should be to encumber the user with stronger authentication only if the resources being protected are worth the effort to use these stricter measures. The approach presented in this paper is one example of this adaptive and seamless authentication trend. Its efficacy can be further augmented by advances in artificial intelligence and machine learning.

ACKNOWLEDGEMENT

This research was done when the second author was working at Gemalto. Gemalto has since been acquired by Thales.

REFERENCES

[1] M. Cavna, “Nobody knows your’re a dog: As iconic Internet cartoon turns 20, creater Peter Steiner knows the joke rings as relevant as ever”, Washington Post, July 13, 2013. [Online, retrieved 10/2019] Available from https://www.washingtonpost.com/blogs/comic-riffs/post/nobody-knows-youre-a-dog-as-iconic-internet-cartoon-turns-20-creator-peter-steiner-knows-the-joke-rings-as-relevant-as-ever/2013/07/31/73372600-f98d-11e2-8e84-c56731a202fb_blog.html

[2] Internet Engineering Task Force, “OCRA: OATH Challenge-Response Algorithm”, Specifications, [Online, retrieved 10/2019] Available from https://tools.ietf.org/html/rfc6287

[3] FIDO Alliance, Specifications, [Online, retrieved 10/2019] Available from https://fidoalliance.org/specifications

[4] D. Suwirya, H. Lu and L. Castillo, “Managing Access to Security Hardware in PC Browsers”, Web Applications and Secure Hardware Workshop (WASH) pp. 3-9, London, UK, 2013, [Online, retrieved 10/2019] Available from http://ceur-ws.org/Vol-1011/1.pdf

[5] Github Repository. Web Bluetooth API, [Online, retrieved 10/2019] Available from https://webbluetoothcg.github.io/web-bluetooth

[6] B. Garska, “Two-Factor Authentication (2FA) Explained: Bluetooth Authentication” [Online, retrieved 2019] Available from https://blog.identityautomation.com/two-factor-authentication-2fa-explained-bluetooth-authentication

[7] T. Hunt, “The 773 Million Record Collection #I Data Breach”, 2019, [Online, retrieved 10/2019] Available from https://www.troyhunt.com/the-773-million-record-collection-1-data-reach