COIN
INTERNET-DRAFT
Intended Status: Informational
Expires: August 28, 2020

C. Sarathchandra
InterDigital Inc.
D. Trossen
Huawei
M. Boniface
University of Southampton
February 28, 2020

In-Network Computing for App-Centric Micro-Services
draft-sarathchandra-coin-appcentres-02

Abstract

The application-centric deployment of 'Internet' services has increased over the past ten years with many million applications providing user-centric services, executed on increasingly more powerful smartphones that are supported by Internet-based cloud services in distributed data centres, the latter mainly provided by large scale players such as Google, Amazon and alike. This draft outlines a vision of evolving those data centres towards executing app-centric micro-services; we dub this evolved data centre as an AppCentre. Complemented with the proliferation of such AppCentres at the edge of the network, they will allow for such micro-services to be distributed across many places of execution, including mobile terminals themselves, while specific micro-service chains equal today's applications in existing smartphones. We outline the key enabling technologies that needs to be provided for such evolution to be realized, including references to ongoing IETF work in some areas.

Copyright and License Notice

Table of Contents

1 Introduction

With the increasing dominance of smartphones and application markets, the end-user experiences today have been increasingly centered around the applications and the ecosystems that smartphone platforms create. The experience of the 'Internet' has changed from 'accessing a web site through a web browser' to 'installing and running an application on a smartphone'. This app-centric model has changed the way services are being delivered not only for end-users, but also for business-to-consumer (B2C) and business-to-business (B2B) relationships.

Designing and engineering applications is largely done statically at design time, such that achieving significant performance improvements thereafter has become a challenge (especially, at runtime in response to changing demands and resources). Applications today come prepackaged putting them at disadvantage for improving efficiency due to the monolithic nature of the application packaging. Decomposing application functions into micro-services [MSERVICE1] [MSERVICE2] allows applications to be packaged dynamically at run-time taking varying application requirements and constraints into consideration. Interpreting an application as a chain of micro-services, allows the application structure, functionality, and performance to be adapted dynamically at runtime in consideration of tradeoffs between quality of experience, quality of service and cost.

Interpreting any resource rich networked computing (and storage) capability not just as a pico or micro-data centre, but as an application-centric execution data centre (AppCentre), allows distributed execution of micro-services where the notion of an application constitutes a set of objectives being realized in a combined packaging of micro-services under the governance of the 'application provider'. These micro-services may then be deployed on the most appropriate AppCentre (edge/fog/cloud resources) to satisfy requirements under varying constraints. In addition, the high degree of distribution of application and data partitions, and compute resources offered by the execution environment decentralizes control between multiple cooperating parties (multi-technology, multi-domain, multi-ownership environments). Furthermore, compute resource availability may be volatile, particularly when moving along the spectrum from well-connected cloud resources over edge data centres to user-provided compute resources, such as (mobile) terminals or home-based resources such as NAS and IoT devices.

We believe that the emergence of AppCentreS will democratize infrastructure and service provision to anyone with compute resources with the notion of applications providing an element of governing the execution of micro-services. This increased distribution will lead to new forms of application interactions and user experiences based on

cooperative AppCentreS (pico-micro and large cloud data centres), in which applications are being designed, dynamically composed and executed.

2 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. Use Cases

Although our motivation for the 'AppCentre' term stems from the (mobile) application ecosystem, the use cases in this section are not limited to mobile applications only. Instead, we interpret 'applications' as a governing concept of executing a set of micro-services where the 'application provider' can reach from those realizing mobile applications over novel network applications to emerging infrastructure offerings serving a wide range of applications in a purpose- (and therefore application-)agnostic manner. The following use cases provide examples for said spectrum of applications.

3.1 Mobile Application Function Offloading

Partitioning an application into micro-services allows for denoting the application as a collection of functions for a flexible composition and a distributed execution, e.g., most functions of a mobile application can be categorized into any of three, "receiving", "processing" and "displaying" function groups.

Any device may realize one or more of the micro-services of an application and expose them to the execution environment. When the micro-service sequence is executed on a single device, the outcome is what you see today as applications running on mobile devices. However, if any of the three functions are terminated on the device, the execution of the rest of the functions may be moved to other (e.g., more suitable) devices which have exposed the corresponding micro-services to the environment. The result of the latter is flexible mobile function offloading, for possible reduction of power consumption (e.g., offloading CPU intensive process functions to a remote server) or for improved end user experience (e.g., moving display functions to a nearby smart TV).

The above scenario can be exemplified in an immersive gaming application, where a single user plays a game using a VR headset. The headset hosts functions that "display" frames to the user, as well as the functions for VR content processing and frame rendering combining

with input data received from sensors in the VR headset. Once this application is partitioned into micro-services and deployed in an app-centric execution environment, only the "display" micro-service is left in the headset, while the compute intensive real-time VR content processing micro-services can be offloaded to a nearby resource rich home PC, for a better execution (faster and possibly higher resolution generation).

Figure 1 shows one realization of the above scenario, where a 'DPR app' running on a mobile device (containing the partitioned Display(D), Process(P) and Receive(R) micro services) over an SDN network. The packaged applications are made available through a localized 'playstore server'. The application installation is realized as a 'service deployment' process (Section 4.2.), combining the local app installation with a distributed micro-service deployment (and orchestration) on most suitable AppCentreS ('processing server').

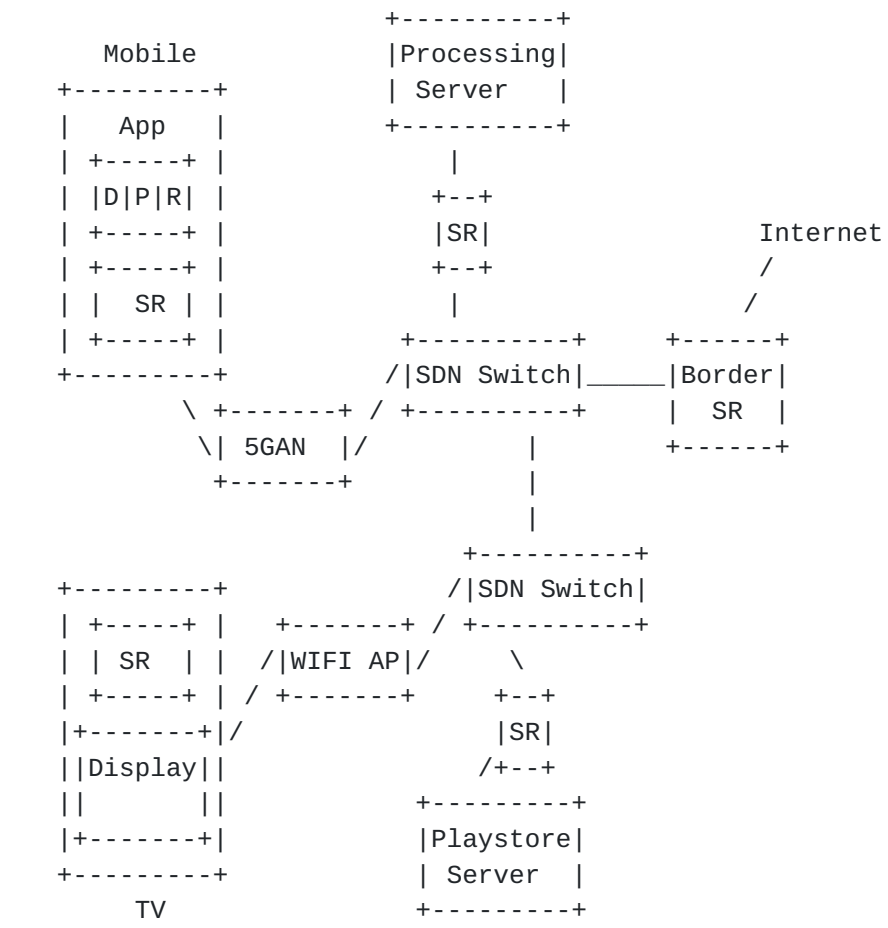

Figure 1: Application Function Offloading Example

Such localized deployment could, for instance, be provided by a

visiting site, such as a hotel or a theme park. Once the 'processing' micro-service is terminated on the mobile device, the 'service routing' (SR) elements in the network (Section 4.3.) route requests to the previously deployed 'processing' micro-service running on the 'processing server' AppCentre over an existing SDN network.

3.2 Collaborative Gaming

There has been a recent shift from applications that provide single-user experiences, such as the ones described in the previous section to collaborative/cooperative experiences such as multi-user gaming and mixed/virtual reality. The latter leads to increasing amounts of interaction where input (e.g., gesture, gaze, touch, movement) and output (e.g., visual display, sound, and actuation) needs to be processed within strict timing constraints and synchronized to ensure temporal and spatial consistency with local and distant users. App-centric design allows functions with high data and process coupling to be modularized, deployed and executed, such that the subset of micro-services is cooperatively executed towards optimizing multi-user experiences.

The same example in previous section can be envisaged from a multi-player gaming scenario. Here the micro-services that need to be executed cooperatively are executed in a localized and synchronized manner for player coordination and synchronizing interaction and state between collaborating players.

3.3. Distributed AI

There is a growing range of use cases demanding for the realization of AI capabilities among distributed endpoints. Such demand may be driven by the need to increase overall computational power for large-scale problems. Other solutions may desire the localization of reasoning logic, e.g., for deriving attributes that better preserve privacy of the utilized raw input data. Examples for large-scale AI problems include biotechnology and astronomy related reasoning over massive amounts of observational input data. Examples for localizing input data for privacy reasons include radar-like application for the development of topological mapping data based on (distributed) radio measurements at base stations (and possibly end devices), while the processing within radio access networks (RAN) already constitute a distributed AI problem to a certain extent albeit with little flexibility in distributing the execution of the AI logic.

Reasoning frameworks, such as TensorFlow, may be utilized for the realization of the (distributed) AI logic, building on remote service invocation through protocols such as gRPC [GRPC] or MPI [MPI] with the intention of providing an on-chip NPU (neural processor unit)

like abstraction to the AI framework.

3.4. Content Delivery Networks

Delivery of content to end user often relies on Content Delivery Networks (CDNs) storing said content closer to end users for latency reduced delivery with DNS-based indirection being utilized to serve the request on behalf of the origin server. From the perspective of this draft, a CDN can be interpreted as a (network service level) application with distributed logic for distributing content from origin server to CDN ingress and further to the CDN replication points which ultimately serve the user-facing content requests. Studies such as those in [FCDN] have shown that content distribution at the level of named content, utilizing efficiency Layer 2 multicast for replication towards edge CDN nodes, can significantly increase the overall network and server efficiency, while reducing indirection latency for content retrieval but also reducing required edge storage capacity by benefiting from the increased network efficiency to renew edge content more quickly against changing demand.

3.5. Compute-Fabric-as-a-Service (CFaaS)

App-centric execution environments, consisting of Layer 2 connected appcentres in the sense outlined in this draft, provide the opportunity for infrastructure providers to offer CFaaS type of offerings to application providers for them to utilize the compute fabric exposed by this CFaaS offering for the purposes defined through their applications. In other words, the compute resources can be utilized to execute the desired micro-services of which the application is composed, while utilizing the inter-connection between those compute resources to do so in a distributed manner. We foresee those CFaaS offerings to be tenant-specific, a tenant here defined as the provider of at least one application. For this, we foresee an interaction between CFaaS provider and tenant to dynamically select the appropriate resources to define the demand side of the fabric. Conversely, we also foresee the supply side of the fabric to be highly dynamic with resources being offered to the fabric through, e.g., user-provided resources (whose supply might depend on highly context-specific supply policies) or infrastructure resources of intermittent availability such as those provided through road-side infrastructure in vehicular scenarios. The resulting dynamic demand-supply matching establishes a dynamic nature of the compute fabric that in turn requires trust relationships to be built dynamically between the resource provider(s) and the CFaaS provider. This also requires the communication resources to be dynamically adjusted to interconnect all resources suitably into the (tenant-specific) fabric exposed as CFaaS.

3.6. Requirements Derived from Use Cases

The following requirements are derived from the presented use cases in Section 3.1. to 3.5., numbered according to the section numbers although those requirements apply in some cases across more than one of the presented use cases.

Req 1.1: Any app-centric execution environment MUST provide means for routing of service requests between resources in the distributed environment.

Req 1.2: Any app-centric execution environment MUST provide means for dynamically choosing the best possible micro-service sequence (i.e., chaining of micro-services) for a given application experience.

Req 1.3: Any app-centric execution environment MUST provide means for pinning the execution of a specific micro-service to a specific resource instance in the distributed environment.

Req 1.4: Any app-centric execution environment SHOULD provide means for packaging micro-services for deployments in distributed networked computing environments, including any constraints regarding the deployment of service instances in specific network locations or compute resources. Such packaging SHOULD conform to existing application deployment models, such as mobile application packaging, TOSCA orchestration templates or tar balls or combinations thereof.

Req 2.1: Any app-centric execution environment MUST provide means for real-time synchronization and consistency of distributed application states.

Req 3.1: Any app-centric execution environment MUST provide means to specify the constraints for placing (AI) execution logic in certain logical execution points (and their associated physical locations).

Req 3.2: Any app-centric execution environment MUST provide support for app/micro-service specific invocation protocols.

Req 4.1: Any app-centric execution environment SHOULD utilize Layer 2 multicast transmission capabilities for responses to concurrent service requests.

Req 5.1: Any app-specific execution environment SHOULD expose means to specify the requirements for the tenant-specific compute fabric being utilized for the app execution.

Req 5.2: Any app-specific execution environment SHOULD allow for dynamic integration of compute resources into the compute fabric

being utilized for the app execution; those resources include, but are not limited to, end user provided resources.

Req 5.3: Any app-specific execution environment MUST provide means to optimize the inter-connection of compute resources, including those dynamically added and removed during the provisioning of the tenant-specific compute fabric.

Req 5.4: Any app-specific execution environment MUST provide means for ensuring availability and usage of resources is accounted for.

4 Enabling Technologies

EDITOR NOTE: Section 4 will be updated and extended in the next version of the draft, including the addressing of specific requirements listed in Section 3.6.

4.1 Application Packaging

Applications often consist of one or more sub-elements (e.g., audio, visual, hepatic elements) which are 'packaged' together, resulting in the final installable software artifact. Conventionally, application developers perform the packaging process at design time, by packaging a set of software components as a (often single) monolithic software package, for satisfying a set of predefined application requirements.

Decomposing micro-services of an application, and then executing them on peer execution points in AppCentreS (e.g., on an app-centric serverless runtime [SRVLESS]) can be done with design-time planning. Micro-service decomposition process involves, defining clear boundaries of the micro-service (e.g., using wrapper classes for handling input/output requests), which could be done by the application developer at design-time (e.g., through Android app packaging by including, as part of the asset directory, a service orchestration template [TOSCA] that describes the decomposed micro-services). Likewise, the peer execution points could be 'known' to the application (e.g., using well-known and fixed peer execution points on AppCentreS) and incorporated with the micro-services by the developer at design-time.

Existing programming frameworks address decomposition and execution of applications centering around other aspects such as concurrency [ERLANG]. For decomposing at runtime, application elements can be profiled using various techniques such as dynamic program analysis or dwarf application benchmarks. The local profiler information can be combined with the profiler information of other devices in the

network for improved accuracy. The output of such a profiler process can then be used to identify smaller constituting sub-components of the application in forms of pico-services, their interdependencies and data flow (e.g., using caller/callee information, instruction usage). Due to the complex nature of resulting application structure and therefore its increased overhead, in most cases, it may not be optimal to decompose applications at the pico level. Therefore, one may cluster pico-services into micro-services with common characteristics, enabling a meaningful (e.g., clustering pico-services with same resource dependency) and a performant decomposition of applications. Characteristics of micro-services can be defined as a set of concepts using an ontology language, which can then be used for clustering similar pico-services into micro-services. Micro-services may then be partitioned along their identified borders. Moreover, mechanisms for governance, discovery and offloading can be employed for 'unknown' peer execution points on AppCentreS with distributed loci of control.

Therefore, with this app-centric model, application packaging can be done at runtime by constructing micro-service chains for satisfying requirements of experiences (e.g., interaction requirements), under varying constraints (e.g., temporal consistency between multiple players within a shared AR/VR world)[SCOMPOSE]. Such packaging includes mechanisms for selecting the best possible micro-services for a given experience at runtime in the multi-X environment. These run-time packaging operations may continuously discover the 'unknown' and adapt towards an optimal experience. Such decision mechanisms handle the variability, volatility and scarcity within this multi-X framework.

4.2 Service Deployment

The service function chains, constituting each individual application, will need deployment mechanisms in a true multi-X (multi-user, multi-infrastructure, multi-domain) environment [SDEPLOY1][SDEPLOY2]. Most importantly, application installation and orchestration processes are married into one, as a set of procedures governed by device owners directly or with delegated authority. However, apart from extending towards multi-X environments, the process also needs to cater for changes in the environment, caused, e.g., by movement of users, new pervasive sensors/actuators, and changes to available infrastructure resources. Methods to deploy service functions as executable code into chosen service execution points, supporting the various endpoint realizations (e.g., device stacks, COTS stacks, etc.), and service function endpoint realization through utilizing existing and emerging virtualization techniques.

A combination of application installation procedure and orchestrated

service deployment can be achieved by utilizing the application packaging with integrated service deployment templates described in Section 4.1 such that the application installation procedure on the installing device is being extended to not only install the local application package but also extract the service deployment template for orchestrating with the localized infrastructure, using, for instance, REST APIs for submitting the template to the orchestrator.

4.3. Compute Inter-Connection

NOTE: left for future revision

4.4. Dynamic Contracts

NOTE: left for future revision

4.5 Service Routing

Service routing within a combined compute and network infrastructure that will enable true end-to-end experiences across distributed application execution points provisioned on distant cloud, edge and device-centric resources (e.g., using ICN/name-based routing methods), is a key aspect of app-centric micro-service execution. Once the micro-services are packaged and deployed in such highly distributed micro-data centres, the routing mechanisms will ensure efficient information exchange (e.g., for satisfying application requirements) between corresponding micro-services within the multi-X execution environment.

Routing becomes a problem of routing the micro-service requests, not just packets, as done through IP. Traditionally, the combination of the Domain Naming Service (DNS) and IP routing has been used for this purpose. However, the advent of virtualization with use cases such as those outlined above have made it challenging to further rely on the DNS. This is mainly down to the long delay in updating DNS entries to 'point' to the right micro-service instances. If one was to use the DNS, one would be updating the DNS entries at a high rate, caused by the diversity of trigger, e.g., through movement. DNS has not been designed for such frequent update, rendering it useless for such highly dynamic applications. With many edge scenarios in the VR/AR space demanding interactivity and being latency-sensitive, efficient routing will be key to any solution.

Various ongoing work on service request forwarding [nSFF] with the service function chaining [RFC7665] framework as well as name-based routing [ICN5G][ICN4G] addressing some aspects described above albeit with a focus on HTTP as the main invocation protocol. Extensions will be required to support other invocation protocols, such as GRPC or

MPI (for distributed AI use cases, as outlined in Section 3.3.).

4.6 Service Pinning

Allocating the right resources to the right micro-services is a fundamental task when executing micro-services on such highly distributed app-centric micro-data centres (e.g., resource management in cloud [CLOUDFED]), particularly in the light of volatile resource availability as well as concurrent and highly dynamic resource access. Once the specific set of micro-services of an application has been identified, during the lifetime of the application, requirements (e.g., QoS) must be ensured by the execution environment. Therefore, all micro-data centres and the execution environment will realize mechanisms for ensuring the utilization of specific resources within a pool of resources (i.e., resources in all app-centric micro-data centres), for a specific set of micro-services belonging to one application, while also ensuring integrity in the wider system.

4.7. Opportunistic Multicast

NOTE: left for future revision

4.8 State Synchronization

Given the highly distributed nature of app-centric micro-services, their state exchange and synchronization is a very crucial aspect for ensuring in-application and system wide consistency. Mechanisms that ensure consistency will ensure that data is synchronized with different spatial, temporal and relational data within a given time period.

5 Security Considerations

N/A

6 IANA Considerations

N/A

7 Conclusion

This draft positions the evolution of data centres as one of becoming execution centres for the app-centric experiences provided today mainly by smart phones directly. With the proliferation of data centres closer to the end user in the form of edge-based micro data centres, we believe that app-centric experiences will ultimately be executed across those many, highly distributed execution points that this increasingly rich edge environment will provide, such as smart

glasses and IoT devices. Although a number of activities are currently underway to address some of the challenges for realizing such AppCentre evolution, we believe that the proposed COIN research group will provide a suitable forum to drive forward the remaining research and its dissemination into working systems and the necessary standardization of key aspects and protocols.

8 References

8.1 Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC7665] Halpern, J., Ed., and C. Pignataro, Ed., "Service Function Chaining (SFC) Architecture", RFC 7665, DOI 10.17487/RFC7665, October 2015, <https://www.rfc-editor.org/info/rfc7665>.

8.2 Informative References

[MSERVICE1] Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: yesterday,today, and tomorrow. In Present and Ulterior Software Engineering (pp. 195-216). Springer, Cham.

[MSERVICE2] Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2016). Microservices architecture enables devops: Migration to a cloud-native architecture. IEEE Software, 33(3), 42-52.

[SRVLESS] C. Cicconetti, M. Conti and A. Passarella, "An Architectural Framework for Serverless Edge Computing: Design and Emulation Tools," 2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Nicosia, 2018, pp. 48-55. doi: 10.1109/CloudCom2018.2018.00024

[TOSCA] Topology and Orchestration Specification for Cloud Applications Version 1.0. 25 November 2013. OASIS Standard. <http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html>.

[ERLANG] Armstrong, Joe, et al. "Concurrent programming in ERLANG." (1993).

[SCOMPOSE] M. Hirzel, R. Soule, S. Schneider, B. Gedik, and R. Grimm, "A Catalog of Stream Processing Optimizations", ACM Computing Surveys,46(4):1-34, Mar. 2014

[SDEPLOY1] Lu, H., Shtern, M., Simmons, B., Smit, M., & Litoiu, M. (2013, June). Pattern-based deployment service for next generation clouds. In 2013 IEEE Ninth World Congress on Services (pp. 464-471). IEEE.

[SDEPLOY2] Eilam, T., Elder, M., Konstantinou, A. V., & Snible, E. (2011, May). Pattern-based composite application deployment. In 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops (pp. 217-224). IEEE.

[nSFF] Trossen, D., Purkayastha, D., Rahman, A., "Name-Based Service Function Forwarder (nSFF) component within SFC framework", <https://datatracker.ietf.org/doc/draft-trossen-sfc-name-based-sff> (work in progress), April 2019.

[ICN5G] Ravindran, R., Suthar, P., Trossen, D., Wang, C., White, G., "Enabling ICN in 3GPP's 5G NextGen Core Architecture", <https://tools.ietf.org/html/draft-ravi-icnrg-5gc-icn-03> (work in progress), March 2019.

[ICN4G] Suthar, P., Jangam, Ed., Trossen, D., Ravindran, R., "Native Deployment of ICN in LTE, 4G Mobile Networks", <https://tools.ietf.org/html/draft-irtf-icnrg-icn-lte-4g-03> (work in progress), March 2019.

[CLOUDFED] M. Liaqat, V. Chang, A. Gani, S. Hafizah Ab Hamid, M. Toseef, U. Shoaib, R. Liaqat Ali, "Federated cloud resource management: Review and discussion", Elsevier Journal of Network and Computer Applications, 2017.

[GRPC] High performance open source universal RPC framework, https://grpc.io/

[MPI] A. Vishnu, C. Siegel, J. Daily, "Distributed TensorFlow with MPI", https://arxiv.org/pdf/1603.02339.pdf

[FCDN] M. Al-Naday, M. J. Reed, J. Riihijarvi, D. Trossen, N. Thomos, M. Al-Khalidi, "fCDN: A Flexible and Efficient CDN Infrastructure without DNS Redirection of Content Reflection", https://arxiv.org/pdf/1803.00876.pdf

Authors' Addresses

Chathura Sarathchandra
InterDigital Europe, Ltd.
64 Great Eastern Street, 1st Floor
London EC2A 3QR
United Kingdom

Email: Chathura.Sarathchandra@InterDigital.com

Dirk Trossen
Huawei Technologies Duesseldorf GmbH
Riesstr. 25C
80992 Munich
Germany

Email: Dirk.Trossen@Huawei.com

Michael Boniface
University of Southampton
University Road
Southampton SO17 1BJ
United Kingdom

Email: mjb@it-innovation.soton.ac.uk