Service-centric Networking

David Griffin, Miguel Rio
University College London, UK
Pieter Simoens, Piet Smet
iMinds/University of Ghent, Belgium
Frederik Vandeputte, Luc Vermoesen
Alcatel-Lucent Bell NV, Belgium
Dariusz Bursztynowski
Orange, Poland
Folker Schamel, Michael Franke
Spinor, Germany

ABSTRACT

This chapter introduces a new paradigm for *service centric networking*. Building upon recent proposals in the area of information centric networking, a similar treatment of services – where networked software functions, rather than content, are dynamically deployed, replicated and invoked – is discussed. Service-centric networking provides the mechanisms required to deploy replicated service instances across highly distributed networked cloud infrastructures and to route client requests to the closest instance while providing more efficient network infrastructure usage, improved QoS and new business opportunities for application and service providers.

Keywords: Service-centric Networking, Anycast, Execution Zone, Orchestrator, Service Placement, Service Router, FUSION, Session Slot, Evaluator, Cloud Computing, Lightweight Virtualisation.

INTRODUCTION

There is an emerging trend for more demanding services to be deployed across the Internet and in the cloud. Applications such as virtual and augmented reality, vehicle telematics, self-navigating cars/drones and multi-user ultra-high-definition telepresence are envisioned beyond the social and office-based applications such as email and photo sharing applications common in today's cloud computing world. While future deployments such as 5G and all-optical networks are aiming to reduce network latency to below 5ms and increase throughput by up to 1000 times (Huawei, 2013) over both fixed and mobile networks, new techniques for efficiently deploying replicated services close to users and the means for selecting between them at request/invocation time are required. Deploying such highly demanding services and providing the network capabilities to access them requires a focused approach, combining the features of service management and orchestration with dynamic service resolution and routing mechanisms leading to *Service-centric Networking*, the subject of this chapter. The focus of this chapter is how to deploy low latency, high bandwidth services on today's IP infrastructures, but as the next generation of wireless and optical networks are rolled out, service-centric networking techniques for the localisation of processing nodes and the selection of running instances will become even more crucial for supporting the vision of the tactile Internet (Fettweis, 2014).

The Internet was originally conceived as a data communications network to interconnect end-hosts: user terminals and servers. The focus was on delivering data between end points in the most efficient manner. All data was treated in the same way: as the payload of packets addressed for delivery to a specific end-point. In recent years, since the development of the world-wide web, the majority of traffic on the Internet originates from users retrieving content. The observation that many users were downloading the same content led to the development of content delivery/distribution networks (CDNs). CDNs cache content closer to the users to reduce inter-provider traffic, and improve users' quality of experience by reducing server congestion through load balancing requests over multiple content replicas. In a content-centric world, communications are no longer based around interconnecting end-points, but are concerned with *what* is to be retrieved rather than *where* it is located. CDNs achieve this by building overlays on top of the network layer but recent research in the domain of Information-Centric Networking has taken matters a stage further by routing requests for named content to caches that are dynamically maintained by the network nodes themselves, rather than having predefined locations of the content, pushed a priori based on predicted demand. Such an approach represents a basic paradigm shift for the Internet.

Although content/information centric networking has received significant attention recently, the approach, like classical CDNs, was originally designed for the delivery of non-interactive content and additional means are needed to support distributed interactive applications. Cloud computing on the other hand has been developed to deliver interactive applications and services in a scalable manner to cope with elasticity of demand for computing resources, exploiting economies of scale in multi-tenancy data centres. However today's typical cloud-based applications tend to be deployed in a centralised manner and therefore struggle to deliver the performance required by more demanding, interactive and real-time services. Furthermore, deploying cloud resources in highly distributed network locations presents a much more complex problem than those faced in individual data centres or cloud infrastructures with only a handful of geographical locations.

Service-centric networking (SCN) is a new networking architecture which aims at supporting the efficient provisioning, discovery and execution of service components distributed over the network. Today's cloud computing architectures are centralised and agnostic of wide-area network performance outside of the data centre. This makes them unfit for geographically distributed services with tight QoS constraints and high bandwidth and computation demands. SCN combines service instantiation and network routing at a fine granularity. Dynamic instantiation of services close to the consumers will naturally adapt to variations in demand. An important dimension includes lightweight interactions between layers for service placement and in-network instance selection without overburdening the latter layer with service-specific logic.

In SCN, we build upon the current trend for edge and fog computing (Cisco, 2014) and envision large numbers of service execution environments distributed throughout the Internet: in access points close to the users; co-located with routers within an ISP's network; in local data centres owned and operated by ISPs; and in traditional data centres and service farms operated by cloud and service providers. As an example, Figure 1 shows three interconnected Autonomous Systems (ASes)/Internet Service Providers (ISPs), each has one or more data centres acting as service execution environments. A service has been instantiated in two locations. From a service management and placement perspective, the orchestrator logic needs to decide in which service execution environment a service should be instantiated. Given this rich set of resources, SCN aims to optimise the location of individual service component instances according to the performance requirements of the application, the location of its users and according to the experienced demand. Replicas of service components may be provisioned according to predicted load levels and furthermore they can be instantiated on-the-fly to deal with demand elasticity. The service placement logic needs to trade-off the costs of instantiating services everywhere in terms of the quantity of data-centre resources required to host these instances against the expected performance of the intervening network which will affect the quality of experience of the users.

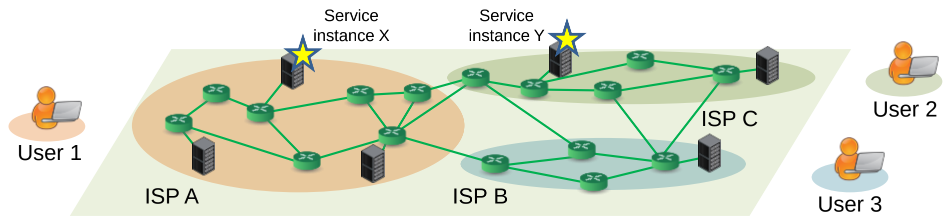

Figure 1: Service-centric Networking – network level view

The service instance placement problem can be formalised as provisioning service component instantiation points, given the location and capabilities of infrastructures, varying demand patterns and the QoS requirements of the service. A key challenge of service-aware networking is the routing and load-balancing of service requests to the best instances given the existence of many different service replicas in the network that could serve the request. To support dynamic service instantiation, lightweight component-based virtualisation technology with reliable isolation properties is required, as well as service description and orchestration languages that can be used to describe an application in terms of service components and interactions.

To meet the service performance targets as well as to support resilience in case of service node failure or network or service-level congestion there will be many replicas of the same service component instance running throughout the Internet. The users, the service providers or the network itself must be able to select an appropriate one. SCN requires a service-anycast capability for resolution in the network so that service instance selection can be optimised on the grounds of proximity, network performance metrics and server load. For instance, with reference to Figure 1, User_1 will receive better network performance from selecting instance X rather than Y, however, if the execution environment hosting instance X is overloaded it may be better for User _1's requests to be resolved to instance Y, or for the SCN system to create an appropriate service instance close to the user on the fly. The user/end host should simply request the service by name, with the binding to instance X or Y being determined by the name resolution/routing system according to a combination of network and service metrics (further details are contained in the section on Service Resolution and Routing, below).

The remainder of this chapter is organised as follows: First related work and background technologies are discussed, followed by the SCN-specific requirements. The problem of service management and orchestration is introduced highlighting the necessary functionality. Then the role of the networking functions for service resolution and routing are discussed. The chapter goes on to describe one possible overall architecture being studied in the FUSION project for bringing together the necessary service and network layer capabilities for SCN operations. The practicalities of designing services for being deployed and dynamically managed in SCN are presented in the next section. Practical system deployment considerations are discussed from the perspectives of the overall business model issues, the software developer and network operator. Finally conclusions on SCN are drawn and future research directions are highlighted.

BACKGROUND

Over more than a decade, Content Delivery Networks (CDNs) have become one of the most important technologies commonly used throughout the Internet in support of scalable delivery of content including rich media, web acceleration/caching/small file delivery and large file/software delivery. CDNs cache content closer to the users to reduce traffic in interconnection links, and improve the quality of experience by providing higher downloading speed, lower delays and improve availability of content compared to what is achievable with standalone servers. To this end CDNs perform optimisations on different levels of system architecture. At the network level, they intelligently optimise the allocation of content among data centres, and route requests so as to assign clients to optimal servers. In addition, multiple content delivery mechanisms are used including such features as Web application acceleration, IP acceleration, Web application streaming, secure content delivery, large file optimisation, download manager, to mention a few (Akamai, 2014b; Conboy, 2014; Deutsche Telekom, 2014; Edgecast, 2014; Incapsula, 2014). Accordingly, a high-level functional architecture of a typical CDN consists of three[i] main building blocks, namely content deployment (responsible for policy-based replication and caching), content delivery (including request routing and lower level content delivery mechanisms), and monitoring (providing measurement data for the purposes of two former blocks) (Buyya, Pathan, & Vakali, 2008). At the heart of CDN is the request routing system, which typically uses customised Domain Name System (DNS)-based resolution to direct client requests to optimal servers in compliance with CDN provider policies. To offer advanced optimisation capabilities such as those mentioned before, CDN solutions are often being combined with network appliances to form application delivery networks (ADN). In such a setting, application delivery controllers (ADC) of ADN optimise the delivery of application traffic from/to distributed data centres at the transport level and load balance traffic within data centres, while the CDN is responsible for routing user requests to data centres hosting appropriate instances of the application.

Application delivery networks (ADN) mentioned above share many principles with CDNs, and the basic difference between them is that ADNs are able to recognise multiple applications on-the-fly and optimise their performance by using different forms of application acceleration and employing layer 4-7 switches to load balance traffic over a pool of servers located in a single data centre. In fact, while CDNs are strongly oriented towards optimal delivery to large populations of clients using CDN resources distributed in many sites, most ADNs operate locally at the level of a single data centre. The latter provides an explanation why merging both technologies becomes a natural evolutionary step for CDN providers. A simpler, yet still distributed scenario assumes Geo DNS- based resolution of client requests among multiple instances of the application with ADCs playing the role of reverse proxies, for example see (Aiscaler, 2014).

The concept of SCN builds on CDNs and ADNs which provide partial solutions to the problems targeted by SCN. In particular, SCN, accounting for service-level information, fills the gaps in network-wide service orchestration and introduces service routing to provide intersection with traffic engineering in transport network and data centres.

Information Centric Networking (ICN) has attained significant attention in recent years (Aranda & Zitterbart, 2010), (Trossen, 2011), (Sail, 2011), (Named Data Networking, 2013), (GreenICN, 2013). A dedicated research group in IRTF has been established (ICNRG, 2014). Representative ICN proposals include such designs as CCN, PSIRP/PURSUIT and NetInf, to mention a few[ii]. As noted in (Ghodsi, Shenker, Koponen, Singla, Raghavan, & Wilcox, 2011), all these architectures share three main design principles, namely use of Publish-Subscribe primitives, adoption of universal (in network) caching, and content-oriented security model tightly coupled with the naming scheme adopted by the design. The Pub/Sub communication model makes the provider and the user mutually invisible and allows them to be online independently of each other. This feature also opens the door to the use of ubiquitous caching with the aim of optimising performance and saving network resources. To allow ubiquitous caching, all ICN designs introduce content-oriented security in place of classical models based on securing the connection. For a comprehensive comparative survey of all recent ICN designs the reader is referred to (Xylomenos et al., 2013). From our perspective, an important fact about ICN is that the introduction of this paradigm by itself does not provide explicit solutions to many problems related to future services. The first explicit attempt to extend ICN from content to services (Named Function Networking, 2013) contributes to the task of sequencing the services in a service chain, but it leaves open several problems including optimal instance selection, which require more sophisticated coordination than that needed for caching of static content.

Cloud computing has been developed to deliver applications and services in a scalable manner to cope with elasticity of demand for computing resources, exploiting economies of scale in multi-tenancy data centres. Just as with CDN services in the past, cloud resources are now being deployed in local ISPs and other distributed network locations, presenting a much more complex problem than can be solved with generalised resource assignment algorithms in individual data centres or cloud infrastructures with only a handful of geographical locations. While new networking paradigms for intra-data centre communications have been developed to facilitate the distribution of data-processing intensive applications over a flexible number of computing devices within the same data centre, these techniques and technologies are limited to specific data centres and services, and have not been rolled out to the wider-area Internet. Although cloud federation has received a lot of attention in recent years the techniques have been aimed at improving scalability for cloud-based applications and they do not address the problem of fine-grained localisation of processing nodes in the network between the federated clouds. Similar conclusions apply also to the converged use of NaaS and cloud technologies despite a lot of research that has been done in this domain (Qiang, Yuhong, Vasilakos, 2012.). The latter refers in particular to related activities undertaken recently in the context of NFV (ETSI, 2013) where the joint use of network and data centre resources is key for the realisation of distributed network services.

Several distributed service management architectures have been proposed with IRMOS, NGSON and PADIS being representative recent examples, discussed in the following paragraphs.

The goal of Interactive Real-time Multimedia Applications on Service Oriented Infrastructures (IRMOS) (Menychtas, 2010) is to enhance SLAs in a grid/cloud computing platforms with providing strict quality guarantees in the transport network. Automatic deployment and instantiation of a service using resources distributed in a network is based on an abstract description of all the execution environment requirements of the service (given in the form of Virtual Service Network, VSN), including the description of the connectivity requirements between service components and their individual QoS demands. To this end IRMOS integrates the orchestration of network resource management and allocation functions for cloud services based on VSN specification. IRMOS relies on strict QoS guarantees so it fits best to managed networks and needs adoptions for wide area Internet.

Next Generation Service Oriented Network (NGSON) (IEEE, 2011) identifies several individual architectural components and functionalities, however, restricted basically to service routing and composition. NGSON provides capabilities for service composition/orchestration which take the form of ordering the invocation of possibly multiple basic services in response to a single request. It also adopts the concept of centralised controller for network resource and QoS control which conceptually corresponds to traditional resource managers and can easily be extended to the form of an SDN controller. NGSON does not cover resource management in data centres and service placement. Extensions to NGSON should thus include service orchestration capable of allocating and load balancing among service instances through active cooperation with distributed execution environment. As of today, only the functional architecture of NGSON has been standardised, but no interface specifications are available. A proof-of-concept implementation was based on the RESTful protocol for service routing and the use of Business Process Execution Language (BPEL) notation for composite service orchestration (Lee & Kang, 2012).

PaDIS (Provider-Aided Distance Information System) (Poese at al., 2012a) is designed as an ISP-operated system to improve server selection for users' requests in the context of CDNs. PaDIS works at the level of local DNS. It intercepts DNS responses for client queries from the authoritative CDN DNS server and rewrites the CDN surrogate server address provided in the A/AAAA record with the address of the surrogate considered optimal for this query. Optimal surrogate selection by PaDIS is based on the local knowledge of ISP about network conditions and topological diversity of CDN surrogate servers learned through sniffing DNS traffic. Conceptually, PaDIS thus allows ISPs to enter the request routing loop of a CDN in order to improve delivery performance based only on server selection without explicitly changing routing in the network. This general idea of cooperation between ISPs and CDNs has subsequently been enhanced by allowing ISPs to (1) rank CDN surrogate servers pre-selected by the CDN instead of rewriting DNS responses on its own (Poese at al., 2012b) and (2) get involved in the process of allocating CDN surrogate servers by automated on-demand negotiation and deployment of new CDN surrogates based on an IaaS model (Frank et al., 2013).

One of the requirements for service-centric networking is the ability of the platform to take network state information into account for the purposes of both the orchestration and service routing. In this context we note that in addition to using raw monitoring data, service-centric networking may potentially benefit from concepts originally developed for overlay applications. A notable example of such a solution is the concept of Application Layer Traffic Optimization (ALTO) (Seedorf & Burger, 2009) which provides network information in the form of abstractions like network map and cost map based on modelling the network as a set of equivalence classes known as Provider-defined Identifier (PID) being collections of end-point addresses. Moreover, ALTO extensions to cover data centre information have also been proposed recently (Lee, Bernstein, Dhody, & Choi, 2014). Despite known proof-of-concept implementations of ALTO (Scharf et al., 2012) and a first commercial product being available (Dharwadkar, 2011), practical adoption of ALTO has been slow. Considering the successful use of (partially) similar services like Radar offered by Cedexis (Cedexis, 2014) one can expect that ALTO additionally needs extensions to multi-domain environments.

Summarising the above discussion, we conclude that while the integration of CDNs, ADNs, NGSON and other known solutions like ALTO is possible at a conceptual level, it is hard to just take existing technologies in order to achieve the goals of SCN. The most important missing parts are network-wide service orchestration and support for the implementation and propagation of network policies to allow service resolution taking account of server load, data centre resources and network conditions. The SCN approach is holistic in addressing these problems as outlined in the remainder of this chapter.

REQUIREMENTS FOR SERVICE-CENTRIC NETWORKING

In this section, the requirements for orchestrating and managing demanding interactive services and execution resources across a distributed set of heterogeneous execution environments are introduced, covering the high-level non-functional service and business requirements that impact SCN.

Service-related requirements

The services that could benefit from a service-centric networking infrastructure share a number of key properties and requirements that potentially have a huge impact on the overall SCN architecture.

Network sensitivity

By network sensitivity, we imply that the functional behaviour of these services is sensitive with respect to the network bandwidth, latency and/or jitter characteristics . Placing these services too far from the end users (e.g., in distant centralised cloud environments) can result in bad QoS, eventually bad QoE, or increased service cost. This is a key necessary property for all distributed SCN architectures, since non network-sensitive services can easily be deployed on classic cloud infrastructures.

Real-time services

Along with the network sensitive nature of services comes the real-time nature of the services, as these envisioned services need to deliver data or a data stream within a specific deadline or at a particular rate. Real-time services will also be sensitive to factors such as computational capacity or storage resources within a data centre (DC).

User session longevity

We envision that services with possibly long active sessions can benefit from service-centric networks. For example, a personalised video transcoding service or game rendering service can be active for several minutes to hours. During this period, potentially large amounts of compute and networking resources may be consumed. Proper service placement, deployment and selection strategies are needed to meet these service requirements.

Resource intensity

The combination of the above service requirements on real-time behaviour and long user sessions necessitates that demanding services must be deployed and managed very efficiently. Many of these services will have very specific resource requirements (e.g., compute-bound, memory-bound, I/O-bound, network-bound, etc.) and may rely on particular accelerators for efficient and effective operation (e.g., a 3D live rendering service typically requires a GPU). Careful selection of appropriate cloud environments and execution nodes is essential for the intended service classes.

Distributed service graphs

Complex services typically consist of a graph of service components that together perform complex functions. Each of these service components in a service-centric network can be deployed in one or more execution environments depending on the various interconnection and execution requirements and constraints. Such services require a more complex orchestration and service selection/routing considering the overall end-to-end performance of the service across multiple domains.

Instant on-the-fly deployment

Accurately predicting service demand patterns is not trivial, especially combined with the fact that services need to be deployed across a potentially large number of execution environments, resulting in a fragmented deployment of services across the Internet. In case of unanticipated load patterns, services may need to be deployed instantly to be able to serve new incoming requests with similar QoS. Secondly, in a universal service-centric networking system where tens of thousands of services can be managed, there will be a long tail of services for which pre-deployment of some instances in all locations is not cost-effective. Under these circumstances, the service-centric networking system should be able to immediately deploy new instances on-the-fly in order to handle these infrequent or unpredictable service requests. As a result, a service-oriented networking system should incorporate on-demand service placement and deployment mechanisms in addition to service selection mechanisms.

Security

Due to the dynamic management and orchestration mechanisms, security (including integrity) regarding service management and service selection are crucial. For example, a service-centric networking system should be able to guarantee that misbehaving entities cannot pretend to be other services and that requests do not arrive at the wrong service instance. Proper service authorisation and authentication mechanisms should be in place.

Business-related requirements

Business-related requirements can also drive and constrain service-centric networks for a number of reasons:

- First, service-centric networks must simplify service deployment without having to deal with the complexities of a highly distributed infrastructure.
- Secondly, service oriented networks should allow for ISPs to provide improved service and network QoS/QoE by leveraging their detailed network information as well as reduce their

network bandwidth costs thanks to smart placement and service selection.

SERVICE LEVEL MANAGEMENT AND ORCHESTRATION

This section discusses a number of candidate high-level architectures for Management and Orchestration (MANO) for service-centric networking, followed by an enumeration of the key service management and orchestration functions. We discuss how these functions are impacted by the specific service requirements and how they can be implemented in a flexible and scalable manner in a distributed execution environment.

Challenges

The requirements outlined in the previous section impose a number of key challenges for any service-centric networking MANO architecture. We briefly elaborate on these challenges.

Scalability and flexibility

SCN architectures need to manage large amounts of service instances of many different service types across numerous execution environments of various types. This means that on one hand, the overall SCN architecture should be able to efficiently scale with increasing amounts of services that are deployed in such architecture. On the other hand, due to the specific requirements and capabilities of the services and available infrastructures, the SCN architecture should also be able to provide enough flexibility so that the various services can be deployed efficiently on the various execution environments. In the next section, we will describe various high-level MANO architectures and discuss their effectiveness with respect to scalability and flexibility.

Heterogeneity

As previously indicated, the services of interest could have drastically different requirements with respect to resources, operations and orchestration. Similarly, in a completely distributed service-centric networking architecture, the various execution and networking environments can be quite heterogeneous in nature as well, from a resource, infrastructure and management platform point of view. More specifically, execution environments will range from standard centralised general purpose cloud environments to highly distributed, small size and specialised execution environments that are located very close to the edge or even within the home, with tight resource and management constraints.

Scalable distributed high-level MANO architecture

Mapping the service-centric networking requirements as specified in the previous section to possible architectures, we see a number of candidate architectures.

- One centralised cloud, possibly including a number of distributed execution environments or zones that are all fully managed by the central cloud orchestrator.
- Collaborative or federated clouds, where each cloud environment operates completely independently but can interact with other cloud environments without one central coordinator.
- Hierarchical cloud, incorporating a global orchestrator which has a high-level view and control of multiple decentralised lower-level cloud environments. Each of these lower-level cloud environments are treated mostly as a black box with respect to the high-level orchestrator and provide their own service management and orchestration functions.

Each of these solutions has a number of advantages and disadvantages regarding scalability, level of control, multi-cloud-provider support, etc. For example, a single centralised cloud architecture has the advantage of ultimate level of visibility and control, as all services and execution nodes are under the immediate control of a centralised orchestrator who has full visibility. However, this solution is prone to scalability issues in case tens or hundreds of thousands of services must be managed across hundreds or thousands of distributed execution environments. A single centralised cloud environment also typically implies only a single-cloud-provider model, which can constrain the possible geographical deployment locations for the services.

With a completely decentralised approach, scalability can be handled more easily. However, this model implies that service providers need to register, deploy and manage their services with multiple cloud providers and in multiple locations, making the configuration and management of widely deployed services more complex.

A hierarchical architecture tries to combine the best of both worlds. In this approach, the advantages of the decentralised cloud architecture (i.e., independent MANO operations and multi-cloud-provider support) can be combined with global service orchestration and management functions. The reduced level of control can be largely mitigated using special techniques such as the concept of evaluator services as proposed in the FUSION architecture, discussed later in the chapter.

In the next sections, we focus on describing the challenges and requirements for managing demanding network-sensitive services in a hierarchical cloud architecture. Two examples of hierarchical cloud architectures are IRMOS (Menychtas, 2010) and FUSION (discussed later in this chapter), where the concept of orchestration domain and execution zones (or nodes) is introduced. Note that both approaches allow inter-domain service orchestration and management that can be modelled either as a decentralised set of clouds or as an extended form of hierarchical clouds with two or more levels.

Key service management and orchestration functions

This section discusses the role and expectations of the key service MANO functions in a service-centric network system.

Service registration

A key service management function is the service registration that handles all registrations of new service types, provides updating with respect to their deployment parameters, and ultimately decommissioning of a particular service in the orchestration domain. Registering a new service may involve a subsequent automatic deployment of a number of service component instances within an orchestration domain, as described in a service manifest. The service manifest identifies the components of a service, the requirements for provisioning in terms of the computational requirements of the infrastructure and the network performance targets for interconnecting users to the service endpoints and between the service components forming the service graph. Service names are assigned at registration time and form part of the service manifest, with the domain orchestrator being responsible for ensuring that names are globally unique. The service manifest is of crucial importance, as a service-centric networking system needs to be able to automatically deploy, manage and interconnect service instances across a distributed set of heterogeneous execution environments. The service manifest needs to capture all necessary information regarding the service graph, its deployment and platform dependencies, lifecycle management, monitoring, automatic scaling as well as various security, business and other policies.

Service placement

Due to the various service requirements, load patterns and heterogeneous execution environments, it is essential to find the optimal execution environments when deploying new instances of particular services. In a hierarchical cloud environment, the service placement problem can be divided in two sub-problems, namely finding an optimal execution environment for hosting the service components, and finding an optimal host within the selected execution environment for effectively deploying and running each service component, not necessarily in this order. A key requirement for this mechanism to work efficiently is that at the domain level the service placement function should have enough information to appropriately assess and rank different execution environments. To avoid scalability or even confidentiality issues, this should not require every execution environment to expose their full resource capabilities and system information to the domain level. Secondly, service requirements can be very application and hardware specific, making it extremely difficult for capturing this in a set of static requirements that need to be specified and understood by a service placement function. Thirdly, as some of the execution environments could be close to the edge, they will likely be less powerful in terms of processing and storage capabilities and therefore only have a constrained set of potentially very specific resources. These resources need to be distributed across the services in a cost-efficient manner. Key trade-offs to be made here will be QoS/QoE and profitability.

One way of assessing the suitability of an execution environment to run a specific service is to make use of an associated evaluator service that can assess and score the efficiency of deploying a service in that environment. Execution environments can apply local policies and preferences by wrapping these scores into offers that remain valid for a period of time. This technique/solution allows for very service specific requirements (potentially incorporating historical data), execution environment capabilities such as GPU support as well as execution environment specific policies to be incorporated at the expense of additional overhead in the execution environments for hosting these service evaluators and preparing the corresponding offers.

To effectively deal with environments that are resource constrained and for which there are multiple independent alternatives, the concept of auctioning has been proposed, in which orchestrators representing different services can bid for specific resources in the execution environments for their service components based on the estimated QoS/QoE their users will receive.

Service placement algorithms operate at several epochs: firstly in a static fashion for the initial service deployment and secondly in a more dynamic mode whenever there is a significant change in user demand patterns or the availability of execution environments as they are added or removed. Dynamic service scaling is discussed further in the following subsection.

Service scaling

Service scaling goes beyond service placement. Whereas service placement mainly involves the selection of what execution environments should host a service, service scaling refers to the number of instances that should actually be deployed in each of the selected execution environments.

Instance-scaling algorithms typically balance resource usage with application performance. Upscaling the number of service instances will improve application-level performance metrics such as response delay, but incurs additional costs (energy, renting). Downscaling is needed to avoid unnecessary capacity costs when the service demand is low.

In a hierarchical architecture, service scaling can occur at the various layers, both at the domain orchestration layer as well as within each execution environment, each implementing their own scaling algorithms based on a combination of service requirements and internal scaling policies. We will refer to these scaling functions as inter-zone and intra-zone scaling, respectively. By zone we refer to a logical representation of a data centre where a service can be deployed.

Regarding intra-zone scaling, a number of options are available. A first option is to leave the intra-zone scaling decision authority to the service itself. In this case, the intra-zone scaling is completely opaque to the orchestration layers. The advantage of this scenario is that application-specific elasticity rules can be applied without exposing Key Performance Indicators to other components. Two main drawbacks are (i) that a standardised scaling interface to the intra-zone orchestrator is needed, and (ii) that each service needs to provide and implement its own scaling mechanism.

In a second intra-zone scenario, all scaling decisions are made by the intra-zone orchestrator. Service instances report Key Performance Indicators (KPIs) to this entity, which will then automatically scale up and down, based on service and other scaling rules and policies. In case this intra-zone orchestrator cannot further upscale, it may notify the domain-level inter-zone orchestrator, which can subsequently take necessary actions.

At the domain level, the inter-zone orchestrator can implement global scaling decisions to ensure resources for a particular service are available across multiple zones. As an example there may be insufficient resources in a specific zone to meet the predicted load of anticipated user demand which require inter-zone orchestration algorithms to identify that additional service component instances should be deployed in alternative zones that meet the required user QoE for the service. Inter-zone scaling can take into account service-specific elasticity rules and policies, load pattern predictions, internal elasticity rules and policies as well as potential explicit triggers from either the lower-layer intra-zone orchestrators or even manual triggers (e.g., from a service provider to identify expected demand spikes or to modify over/under-provisioning policies). These inter-zone scaling decisions will subsequently trigger placement and corresponding deployment decisions.

Service deployment

In a hierarchical architecture, service deployment takes place at the level of the orchestration domain and that of individual execution zones. The domain-level service deployment function involves triggering and coordinating the deployment functions at the selected execution environments to start deploying new instances within their execution environments. How the latter is implemented and enforced is up to the execution environment. This decoupling improves both scalability as well as flexibility, as each execution environment can customise and optimise its service deployment function with respect to the execution environment specifications and policies. A crucial factor here is that the service placement and scaling functions at the domain and execution environment levels need to provide all necessary service and resource deployment specifications so that the execution environment can efficiently and automatically deploy the new service component instances within the execution environment.

Service and resource monitoring

Capturing, aggregating and propagating state and runtime information for both resources and services is a key MANO function in a service-centric networking architecture. It provides valuable feedback information to various parties, enables real-time or longer automated or manual feedback and control loops to improve the system and its services. Monitoring functions typically incorporate at least the following set of metrics:

- Data centre or execution infrastructure related metrics
- Networking related metrics
- Service execution related metrics
- Application related metrics
- Service-oriented networking architectural related metrics

Given the potential scale of service-centric networking architectures, a crucial trade-off is how much information should be captured at the lowest layers and how much information should or could be propagated and aggregated to higher layers to keep the amount of monitoring information manageable. Security, privacy or business incentives could restrict the amount of information that can be exposed or interpreted to higher orchestration layers.

SERVICE RESOLUTION AND ROUTING

The SCN routing sub-system needs to provide a new *anycast primitive* that combines network and service metrics. Names need to be converted to the best instance and the service routing system needs to be able to trade-off network quality with server quality. Whilst implementing this decision the service routing needs to try to accommodate service specific requirements to provide good quality of experience to the user at the same time that it load balances the service load amongst all the replicas. Obviously, service instance quality needs to be conveyed to the routing sub-system accurately and efficiently.

An intuitive way of implementing this sub-system of SCN, would be to retrofit the Domain Name System (DNS) with a small set of features that would enable the finding of the best closest replica of a given service. This however would have limitations on what could be achieved, namely:

- DNS is tied to a particular name space which is organised hierarchically. Different applications may want to use different name spaces, either flat (e.g., Magnet links), different hierarchies (e.g., ISBN), more expressive name spaces, randomly allocated names, full URLs, etc. Retrofitting these in DNS would only be possible by completely changing IP-level information routing and the client access protocol.
- Service Anycasting could not be retrofitted in DNS without a major overhaul. One needs a significant amount of service routing information to be propagated, including both network and service metrics. This will need a new protocol and is important to tie it to the resolution/invocation component so that a feedback cycle can be achieved.
- The only result returned by a standard DNS query for a service is a single IP address. However, a composite service could be constructed from different components executed on different servers or make use of multiple transport sessions, using different ports and possibly different transport protocols. A service-centric network will need to be able to return more complex structured results.
- DNS does not allow for clear implementation and propagation of network policies. ISPs can “hijack” DNS queries and return preferential results but this is not done in a formal way and does not allow for collaboration implementing traffic engineering policies. In addition, network providers may want to prioritise some services to some clients. A SCN control plane should have a well-designed intersection with traffic engineering and traffic prioritisation. These capabilities can be significantly increased with recent developments in software defined networking (SDN).
- The ability for services to restrict access to their services is becoming crucial in an increasing adversarial Internet. Denial of service is a top concern for the Internet ecosystem and a SCN should be able to restrict access at the edge of the network. This would enable the deployment of more secure services. Exposing IP addresses to the outside world limits this capability.
- DNS allows for very little client parameterisation when a query is issued. A SCN will enable parameters to be passed at query/invocation type where the client can indirectly control the response it receives. For example, a client might only be interested in service instances in a radius of 5ms. Another important option is for the client to request for more than one instance and then choose which one to contact. This can be used for parallelism and resilience purposes.

Another dimension to be explored is the amount of disruption a SCN may and should cause to the Internet architecture. In particular a choice needs to be made between a clean slate architecture or an overlay solution that works over an unmodified IPv4/IPv6 Internet. Although a clean slate approach would potentially provide a slightly more efficient control plane (since the queries/request would follow a more optimal path) this is not a sufficient argument to propose a complete overhaul of the Internet fabric. Changing network layer functionality has proven incredibly difficult in the last decades. IPv6 transition is the classical example but there are lots of good ideas proposed by the research community that stumble due to the ossification of the Internet.

An overlay solution that aims to complement the Domain Name System is much easier to deploy. It will consist of software running on commodity servers which is easy to replace, update and deploy. Memory becomes much less critical allowing for a more feature-rich service-centric architecture where meta-data for each service can contain many attributes.

Given the above context the networking requirements of a Service-centric Network are as follows:

- Each service is identifiable by a globally unique name. Apart from an overall convention preventing naming clashes, each service can choose the naming scheme that it wishes.
- Service instances may have to be authenticated.
- Services are supported by the existing IPv4/IPv6 architecture where the data plane is not changed.
- A composite service may be formed of multiple service components that may be running on more than one server, in different ports and using different transport protocols.
- The service-centric network overlay needs to resolve names to locators or to directly send messages to service instances without exposing their IP addresses to the users.
- End users' clients should be able to report Quality of Experience back to the overlay and that information should be used to inform future service resolution/routing decisions taken by the overlay.
- The service overlay should provide authentication at the edge so that services can choose to only be contacted by authorised clients to mitigate denial of service attacks without unwanted traffic being propagated far into the network.
- A network provider running a service resolver/router as part of the service overlay should be able to influence the service resolution decisions according to its preferences and policies, for example according to the network costs to reach different instances of the same service.

Although the network layer of SCN could be implemented using a variety of primitives, the following messages illustrate a possible set for meeting the requirements outlined above:

1. REGISTER: This message is used to register service component instance(s) with the local service router. It is potentially authenticated to certify that the service is legitimate.
2. UNREGISTER: This message is used to unregister service instance(s) when these stop being available.
3. SERVICE_UPDATE: Message sent from the execution zone to local service router and between service routers for conveying the known service instances and their associated metrics.
4. RESOLVE: This message is issued by a user's client, or a service component in the case of a composite service. It requests the service routing/resolution system to resolve a given name to one or more network locators.
5. INVOKE: This message is also issued by a client/service component. Rather than requesting that a name is resolved to a locator, the service routing overlay forwards the service invocation request directly to the best server instance through the overlay.
6. QOE_REPORT: Report sent by the client to the local router about a particular flow
7. ROUTING_UPDATE: Message sent between service routers to establish the overlay. It is not service specific.

In the following sub-section we present an overall architecture integrating the resolution and routing functionality as discussed in this subsection with the service management and orchestration functions as presented in the preceding subsection.

FUSION ARCHITECTURE

FUSION (Future Service Oriented Networks) is an EU FP7 project developing a new networking architecture designed to support efficient provisioning, discovery and execution of service components distributed over the Internet.

The FUSION framework can be seen in Figure 2~~Figure 2~~. In comparison with Figure 1, two layers are above the routing plane to accommodate the service routing and execution planes for enabling SCN functions. Functionality is divided into 3 layers. At the lower layer IP routing forwards packets using traditional end-to-end protocols. At the upper layer the execution plane consists of all the execution zones where the service instances will run. In the middle the service router layer will forward requests from clients to the appropriate service instances.

The basic operation of the FUSION system is that orchestration domains – consisting of a potentially large number of geographically distributed execution zones – deploy services on behalf of application developers or service providers in one or more execution zones according to the expected demand by service users. This is depicted in the upper layer of Figure 2~~Figure 2~~. Service routing domains, consisting or more service routers, are responsible for matching service requests referring to a service by serviceID to execution zones containing running instances of the requested service. This is depicted in the middle layer of Figure 2~~Figure 2~~. Service routing is anycast in nature – the user simply requests a service and it is responsibility of the service routing plane to find the “best” available instance for that request. Once a specific service instance in a specific execution zone has been selected for the user request data plane communications take place in the data forwarding plane depicted by “IP Routing” in the lower layer of Figure 2~~Figure 2~~. Note that the physical data centres are depicted in the lower IP routing layer as the data communications will be directly established between users and service instances running in physical data centres, while execution zones – logical representations of a data centre – are shown in the upper execution plane.

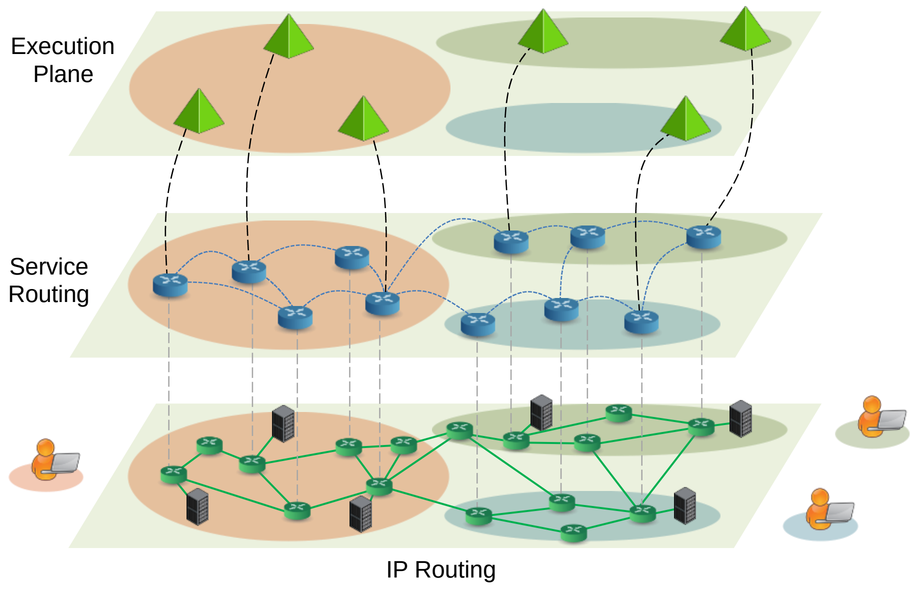

Figure 2 - FUSION Framework

FUSION architecture overview

The main functional entities in the FUSION architecture are depicted in Figure 3~~Figure 3~~. The three main are the orchestrator, execution zone and service router.

The *orchestrator* manages its orchestration domain resources including execution zones and services which it manages on behalf of application developers (or service providers). The orchestrator is responsible for service management functions including service registration, server placement (selecting appropriate execution zones to execute service instances), service lifecycle management and monitoring.

The *execution zone* is the logical representation of a collection of physical computational resources in a specific location, such as a data centre, which is managed by an orchestrator. The orchestrator has an abstract view of an execution zone and the detailed internals are managed by a *zone manager*. The zone manager is responsible for managing service instances within its zone but under the instruction of the orchestrator. It will select the specific physical location (VM, machine, rack, etc.) of individual service instances and interact with the local infrastructure management platform of the data centre/cloud node for VM lifecycle management. The execution zone interacts with the communications infrastructure of the outside world through a service gateway. The service gateway interacts at the level of the service routing and forwarding planes and IP.

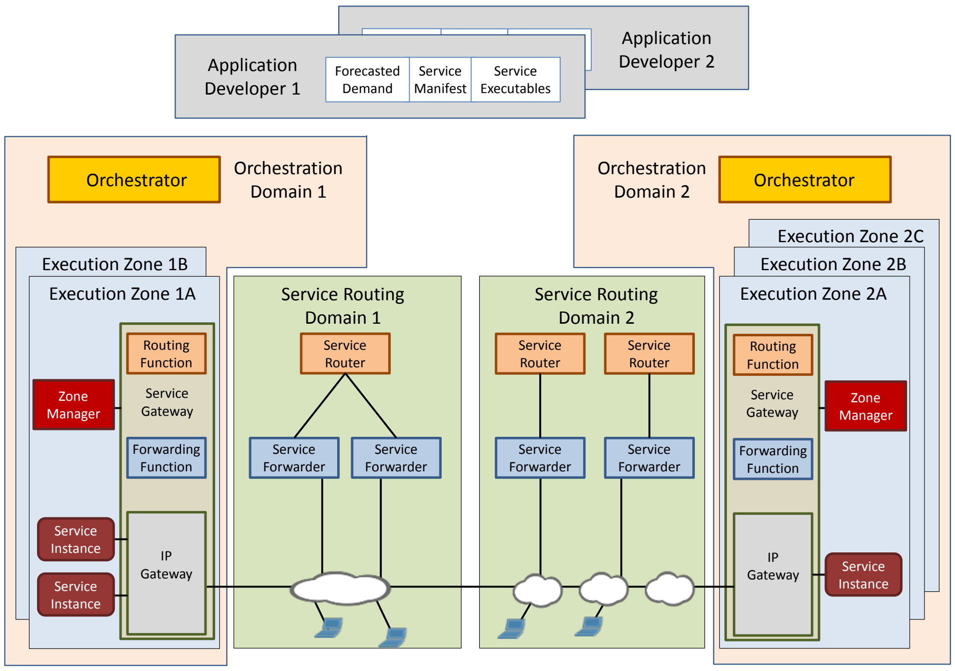

Figure 3: FUSION High Level Functional Architecture

The *service router* is responsible for maintaining and managing service routing information to create forwarding paths for queries/invocation requests from users and other service instances to be resolved or forwarded to execution zones containing available running instances of the specified serviceID. Service forwarding and service routing functions are shown separately^{iii} in Figure 3~~Figure 3~~. The service part manages the routing information injected by execution zones on available serviceIDs and runs routing algorithms to populate forwarding entries in the service forwarder. The service forwarder receives queries/invocation requests and forwards them according to the forwarding tables managed by the service router. The interfaces for routing updates and for forwarding queries are distinct but another reason for separating the functions is that we are considering two different models for implementing routing algorithms within the project. In one model the service routing functions are centralised within an orchestration domain as shown in domain 1 on the left hand side of Figure 3~~Figure 3~~ – in this the centralised routing algorithms may be co-located with a centralised orchestrator functionality in the business model case of combined orchestration and service routing domains. The second model distributes the routing functionality, co-locating it with service forwarding as shown in domain 2 on the right hand side of Figure 3~~Figure 3~~.

SERVICE DESIGN REQUIREMENTS AND PRINCIPLES

Since one goal of service-centric networking is to move services closer to the user running on distributed execution environments, interactive applications can be deployed more optimally across the network, optimising both networking and execution behaviour. These new possibilities are especially interesting for software vendors which require both high network and execution throughput as well as QoS. One example is the entertainment software industry, where similar efforts have recently been started, however without leading to major breakthroughs yet. Another example are the broadcasting, video and movie industries, where also a trend is seen to move towards more demanding and interactive applications, i.e., interactive EPGs (electronic program guides) or live online collaboration, which no longer only rely on pre-generated content but also require increasing amounts of personalised rendering.

This section looks into several key technical aspects from the perspective of these stakeholders. The three main differences compared to traditional software are:

1. Infrastructural level: This mainly focuses on how SCN-enabled software can be deployed and addresses the communication interfaces of the deployed components with the network. Changes to existing software may be required. However, with appropriate application wrappers and manifests existing services could also be adapted to the SCN environment.
2. Architectural level: increased network throughput enables new ways of separating software modules. Traditionally, synchronisation for interactive internet applications was done by exchanging small amounts of data such as position vectors of objects and synthesising this information separately on each client, which required the client to perform its own calculations. With the new possibilities provided by SCN, especially regarding better networking performance due to service replication and optimal service selection, interactive media applications can move functionality from the client to the server, e.g., perform rendering on the server resulting in very thin client applications for the users.

In the following paragraphs, the above mentioned challenges are described in more detail.

Running media software on a service-centric networking infrastructure

One important task of many media software vendors is to adapt their software to service-centric networking infrastructures. Porting software to the underlying operating system environment involves packaging it in a way it can be installed, run and terminated in a standardised way, for example by packaging it in a container or virtual machine. The virtualisation of the underlying hardware is important because the strengths of software oriented networking applications is that the exact hardware and hence its specification is not known in advance. Additionally – as today's servers are running multiple virtual machines – isolation of different applications from different vendors and customers is a requirement. One possible approach is the use of Docker or similar light-weight container technology.

Classical interactive media software usually runs as a fat client or as a stand-alone application on consumer platforms which are optimised for such applications, for example Microsoft Windows. Because of the widespread usage in the consumer sector, GPU driver support for Windows is much better than for Linux. However, today’s cloud infrastructures are often based on Linux for good reasons, since Linux-based platforms are well suited and proven for large-scale administration and tend to be more lightweight which makes deployment easier. The open source paradigm also allows faster incorporation of new technologies, like, for example, the integration of virtualisation functionalities in kernel modules. Last but not least, easier licensing and cost reduction play an important role for software, which will be deployed to a pool of compute resources where it is not always clear how many instances of this software are running at a given time.

For multi-media applications like entertainment software or any other software incorporating real time rendering, access to highly parallelised vector arithmetic hardware, such as GPUs is crucial. Additional research is required to solve challenges regarding efficient and light-weight GPU access without having to carry a full graphical windowing environment on different platforms. For example, in Linux environments, accessing a GPU without a full X environment is still an issue. Research and development in this area is currently a very active topic. Based on the above we envision that first services without specific hardware accelerator requirements will be deployed in SCN.

As more data centres become equipped with such hardware accelerators, (e.g., Amazon EC2 amongst other currently provide VM instances with GPU support), more demanding applications will be able to leverage SCN.

Input and output channels

Many applications fall into one of the following categories:

1. Standalone applications running on a device.
2. Server-client based applications with standard clients (usually a web browser).
3. Server-client based applications with proprietary clients that implement application specific logic, for example performing calculations locally before sending the results to the server or performing calculations on the data received from the server. Such clients can be considered as fat clients.
4. Server-client based applications with proprietary clients that perform application-independent functions. An example is cloud-based gaming where the client implements simple tasks such as input forwarding and video streaming the output. Such clients can be considered as thin clients.

Standalone applications and fat clients often have drawbacks with respect to mass-deployment across many different devices, which often have a wide variety of hardware and software platforms with varying capabilities and constraints. This not only results in huge porting efforts but it can also result in different QoE, as some devices may support particular features, whereas others do not. For example, mobile devices, set-top-boxes and other upcoming devices often have very restricted hardware, allowing only a subset of these applications to be deployed efficiently on these devices. Furthermore, offline software is prone to software piracy because all of the relevant data is shipped to the end user, where software is subject to possible illegal copies.

One example of a thin client model is the cloud gaming service provider Onlive, which is running its games in specialised data centres, which can be accessed by the users using custom thin client applications. Another example is Sony's PS4, which addresses the problem of the PS4 hardware not being compatible with legacy PlayStation games by integrating their services with Gaikai delivering video stream-based gaming. Another advantage of running the applications in the network is that software maintenance and upgrades are largely simplified.

This leads to a trend towards thin, general purpose clients, which mainly forward user I/O to the respective server. The question remains whether the output from the server should be performed using a higher level output description, like HTML or the X-Server protocol, or if the output is readily prepared as a video stream. While HTML or X-Server are often already considered as thin clients, these approaches have many of the disadvantages discussed above, for example requiring suitable hardware to run parts of the software on client-side, especially complex output generation operations. Service-centric networking optimises the placement and selection of service components to meet target QoS/QoE metrics for the services/users to improve the performance of client-server interactions, thereby enabling more services to be deployed in the cloud rather than as stand-alone applications. Applications can benefit from the advantages discussed with even thinner clients making this approach suitable for many advanced applications.

In addition to the above architectural changes, higher level changes and challenges have to be faced. There are already protocols for remote desktop connections (like VNC, RDP and so on), which are optimised for delivering desktop video output over large distances. The disadvantage of these technologies is that their compression algorithm is largely based on the fact that at standard graphics output on desktop PCs, only small parts of the screen change with every frame. This does not normally work as well for multimedia applications due to greater volume of picture changes to be delivered. Classical video streaming, as used for video-on-demand services, is, on the other hand, optimised for the compression of rapidly changing pictures; however, these protocols do not typically support return channels for conveying user input. Therefore a combination of desktop capturing and video streaming approaches need to be developed and possibly standardised for implementing and deploying interactive media applications. In addition, video codecs add a significant computation overhead, which often requires specialised hardware. Developers need to be able to assess the capabilities of cloud resources before deploying components that depend on specialised hardware capabilities.

Session slots

In today's highly interactive media applications, for example entertainment software, the rendering part is primarily designed to generate output for a single user, mainly because this was the main use-case when running such software on an end-user device. Therefore today the most commonly used approach for porting such applications into the cloud is to run a separate process instance or even dedicate a VM for each user. However, this approach is obviously inefficient since multiple user sessions cannot share common data and instantiating a new session can take a significant amount of time.

For cloud computing this approach may be sufficient because scaling can be achieved by increasing the number of machines instantiated. However this does not solve the problem of longer start-up times. In service-centric networking however, the hardware at the optimal location (based on network metrics) cannot be increased at short notice. To make full use of the new possibilities introduced with service-centric networking it is therefore necessary to think about possible optimisations do not dilute the advantages achieved. Therefore a special focus is placed on the question of how to serve multiple users with a single process. A single running software instance will now be able to support multiple users which are logically distinct but share the resources allocated to that software. Rather than dealing with the implementation complexity of each service a service developer will identify the quantity of *session slots* an instance can support. This is a service-independent way of identifying the resources available to serve multiple users simultaneously. Service placement and scaling algorithms, as introduced earlier in the chapter, can manage the quantity of session slots supported by the service component instances running in a zone without being concerned with the implementation details.

SYSTEM DEPLOYMENT CONSIDERATIONS

Business considerations

Today ISPs are confronted with an increasing multiplicity of services that have to be deployed, updated and managed. These services, with their compute and networking requirements, and the ever increasing speed of service deployment time have an important impact on ISPs' business models and business parameters such as target addressable market, revenues and total cost of ownership amongst others.

ISPs hosting real-time-aware services are confronted with choices whether to host these services using:

1. hardware appliances on customer basis,
2. dedicated service offerings using centralised cloud
3. a SaaS-like solution.

In the SaaS-like case applications could be offered as part of a central application store whereby services are rolled out automatically, incurring increasing costs due to hardware and software investments caused by the automation itself but carrying potential revenue increases. Each of these options implies a different compute and network architecture whereby the investments in compute platforms (hosting these services) need to be balanced against network related investments.

Along with the growing number of customer oriented applications to be supported, the fast-pacing progress in the NFV area opens a window of opportunity for ISPs and service providers to virtualise their core services and co-host these services with personalised customer services thereby optimising compute resources and operational benefits in a unified management platform.

Specifically due to the automated service deployment and service routing capabilities of SCN, ISPs are presented with a specific set of questions and trade-offs. A key question being “centralised vs. distributed” whereby investments in compute platforms need to be balanced against network related investments at the level of their backbone, aggregation or access networks.

The economical optimum for service deployment is service specific and should be determined by the geographical and time distribution of the services usage patterns along with its specific networking and QoS requirements and the possible statistical multiplexing benefits of running multiple services on the same infrastructure.

Developer considerations

In the past, developers of server-client software had two, very clearly distinguishable options: Implementing logic of their service on the server or on the client side. Either choice had some implications:

1. If the required processing was highly computation intensive, placing them on the server was a good choice if not all clients (e.g., consumer PCs or mobile devices) have the required computational power.
2. If low latency was needed (especially for immediate feedback from users), a short network distance was preferable. If this collided with the first item (heavy calculations with immediate feedback needed, e.g., physics simulations in virtual worlds), as a compromise it was possible to approximate the computation on the client side with very short feedback time and overriding these results with the final and more accurate server computation results when these were available (resulting in physics objects snapping to their correct position some milliseconds after an impact occurred, for example).

Service-centric networking however enables a smoother model, where centralisation can now be traded against network performance with fine granularity. However, this raises several considerations for the service orchestration functions of placement and scaling algorithms and on the request resolution/service routing functions at service access time. These include:

1. Which computation power is available at a given execution environment? With distributed, localised execution environments it may no longer always be the case that a server with the required resources is located in a central location remote from the client.
2. Are there specific hardware requirements that must be fulfilled? For example, GPU support is a common requirement in media applications.
3. What are the costs of running a service instance at a given execution environment?
4. What are the network performance metrics from the client to the server and between service components running in a distributed fashion?

The above questions have to be answered by the orchestration logic which decides where to place a service and the service resolution/routing functions which select dynamically between running service component instances. Service providers do not necessarily have to care about the specific deployment decisions made by the orchestration functions. However, if they are aware of these questions, this can help developing the service software in a way that optimal placement can be facilitated.

ISP considerations of technical aspects of system deployment

There are several recent trends that indicate possible deployment scenarios of SCN. Of particular importance is the growing interest of ISPs in adopting virtualisation techniques in order to optimise their infrastructure and broaden business opportunities. Some ISPs have been running relatively small data centres for several years to offer services like hosting, utility computing or cloud computing to their customers (AT&T, 2014; Deutsche Telekom, 2012). Such data centres have also hosted appliances used for internal purposes of the ISP like, e.g., CGNAT and DPI. Recently, several ISPs have deployed CDN infrastructures on their own to reduce network traffic and improve QoE for their customers (CDN Planet, 2011; Telecompetitor, 2011; Deutsche Telekom, 2014), and some of them have subsequently established alliances with major CDN providers (Akamai, 2012, 2013a, 2013b, 2014a; Orange, 2012), often in order to extend their service offerings, e.g., (Bartley, 2014; Campbell, 2014). Obviously, existing data centres of the ISPs are natural candidates for the placement of CDN servers inside the ISP domain. The active role of major ISPs in multiple NFV proof-of-concepts (ETSI, 2014) confirms their growing interest in migrating their infrastructures assuming the use of virtualisation techniques for the implementation of many network functionalities. Migration strategies to be adopted by individual ISPs will of course depend on their unique preferences. However, there are good reasons to expect that options of particular interest to many ISPs will be based on aggregating current access and edge functions and collocating them together with moderate-size (mini) data centres given sufficient degree of geographical distribution of such new points-of-presence. In fact, the number of such locations in a single ISP domain may be quite large (for example, AT&T claimed to have 38 data centres around the globe with 23 of them located in North America, AT&T, 2008) which gives an ISP a lot of flexibility in configuring its virtualised infrastructure.

The above facts, together with business considerations provided previously, suggest that ISPs are important candidates able to host SCN-enabled execution zones in direct proximity of users and efficiently operate the service routing plane based on their knowledge of the network. The deployment facilities for SCN functions can be based on the mini-data centres of the ISPs meaning that dedicated large data centre infrastructures may not need to be deployed. We note also that in this scenario, service routers of a given ISP can additionally route requests to external data centres not hosted by the ISP. Under these assumptions, the details of deployment scenarios for the service routing plane in a given ISP domain may depend on specific requirements of future services with regard to delays of the resolution process. A preferred option is to have a central service router (or cluster of routers) handling all queries in a given ISP domain. However, given stringent requirements for delays and using similar arguments as those provided in (Poese at al., 2012b) regarding the deployment of Content-aware Traffic Engineering (CaTE) resolvers, the ISP can decide to distribute service routers among its points-of-presence. Such distributed service routers still could constitute a logically centralised entity, i.e. there would be no forwarding of messages between them. Yet, we admit that explicit use of forwarding capability of service routers would be justified in case of attaching technologically closed subdomains such as the Evolved Packet Core (EPC) part of the LTE (Long-term Evolution) architecture. The latter option fits well scenarios for large ISPs operating both fixed and mobile domains.

CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

Today’s centralised cloud infrastructure does not provide the required flexibility for fine-grained deployment of real-time, interactive applications. In this chapter we presented a novel service-centric networking paradigm that aims to optimise bandwidth and response time of such services by combining advanced service placement, replication and in-network selection at fine granularity. We described several aspects of the architecture; the orchestration, service routing, service design and service deployment. For each aspect we listed the building blocks, component interactions and considerations for future development.

Users will benefit from the bandwidth and response time optimisation potential of balancing service loads at the network level. Moreover, the suggested anycast routing and late binding network primitives increase resilience to congestion and failures as service requests are forwarded to geographically distributed instances at run-time.

Software service providers and software vendors face new challenges but also new opportunities by service-centric networking infrastructures. The main challenge is to port their software to these new deployment platforms, which often requires not only to port to different operating systems and support interaction with the network (infrastructural changes), but also to change the distribution of application components between servers and clients (architectural changes). A main opportunity for software service providers and software vendors is access to a service-centric infrastructure (formed by orchestration, placement and scaling algorithms, service replication and dynamic instance selection on a combination of server and network metrics) which uses resources much more efficiently provides a much higher QoS to the end user.

ISPs will no longer serve as “dumb pipe” providers to over-the-top services, but may directly offer service hosting capacity with additional advantages like flexibility, geolocality and low-latency access compared to today’s cloud providers. As service-oriented-networking enables on-demand deployment, scaling and load balancing, ISPs can lower their costs for service operation and maintenance while guaranteeing network and execution platform performance. In turn, this will lower the barrier for smaller application developers to roll out advanced services with tight networking constraints. Application developers will be able to describe complex deployment constraints including dynamic aspects.

The service routing plane goes hand in hand with the development of a novel service orchestration layer. Appropriate service placement and scaling mechanics must take into account both short-term (current demand) and long-term (cost, policy) metrics. Given the heterogeneity and stringent requirements of the targeted services, traditional techniques used in centralised clouds must be augmented with novel capabilities like just-in-time deployment and provisioning (even triggered at service-request time). Service placement within a zone must exploit appropriate accelerators (GPUs, encoders, etc.) and efficiently share sources and resources (stored 3D objects, textures, decoded video frames, GPU buffers, transcoding function, subtitling service, etc.)

To realise this new service-centric networking paradigm, many research challenges lie ahead. Future work includes detailing the service routing plane to enable routing based on a number of metrics including network characteristics, server load and operational costs for a multitude of services. The forwarding tables are managed by network components to be able to redirect requests to the best instances based on changing server load and network characteristics. Selection agility is required, as server and network characteristics may rapidly change over time and space. A major research topic is how the in-network selection can quickly adapt to these changing conditions. Here we must find a trade-off between frequent monitoring to allow accurate predictions and the bandwidth which this background traffic consumes. A further consideration is how routing across multiple service routing domains is achieved, considering the trade-off between the granularity of announcements of the availability and load of service instances/execution zones versus the overhead and complexity of maintaining large amounts of state information in service routers. Another avenue of future research is the development of on-demand service deployment. Service management functions can detect the need for additional service instances to be deployed or for instances to be migrated between execution zones. At the domain level, orchestrators might wish to sub-contract execution zones of other domains. Developing an inter-domain orchestration protocol, considering the complexities of dynamic service placement and scaling is one of the research challenges that is still open. Finally, the requirements and designs must be validated in targeted test cases and large-scale prototypes.

REFERENCES

Aiscaler. (2014), Product overview. Retrieved from http://aiscaler.com/product-overview#ADN

Akamai. (2012). Akamai and AT&T Forge Global Strategic Alliance to Provide Content Delivery Network Solutions. Retrieved from http://www.akamai.com/html/about/press/releases/2012/press_120612.html

Akamai. (2013a). KT and Akamai Expand Strategic Partnership. Retrieved from http://www.akamai.com/html/about/press/releases/2013/press_032713.html

Akamai. (2013b). Swisscom and Akamai Enter Into a Strategic Partnership. Retrieved from http://www.akamai.com/html/about/press/releases/2013/press_031413.html

Akamai. (2014a). Akamai and Telefonica Enter into Global Content Delivery Alliance. Retrieved from http://www.akamai.com/html/about/press/releases/2014/press-032514.html

Akamai. (2014b). Web Application Accelerator. Retrieved from http://www.akamai.com/html/solutions/web_application_accelerator.html

Aranda Gutiérrez, P., & Zitterbart, M. (Eds.). (2010). Final Architectural Framework. 4WARD. In *Architecture and Design for the Future Internet FP7-ICT-2007-1-216041-4WARD Deliverable D-2.3.1*. Retrieved from http://www.4ward-project.eu/

AT&T. (2014). Products & services. Retrieved from http://www.business.att.com/enterprise/business-solutions/

Bartley T. (2014). Orange is Glad they Chose Akamai for Live Video & Events. Retrieved from https://blogs.akamai.com/2014/06/orange-is-glad-they-chose-akamai-for-live-video-events.html?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+TheAkamaiBlog+%28The+Akamai+Blog%29

Buyya, R., Pathan, M., & Vakali, A. (Eds.). (2008). *Content Delivery Networks*. Berlin Heidelberg: Springer-Verlang.

Campbell T. (2014). Orange: "OTT services bring new revenue growth opportunities", Retrieved from http://www.iptv-news.com/2014/06/orange-ott-services-bring-new-revenue-growth-opportunities/

CDN Planet. (2011). Telefonica. Retrieved from http://www.cdnplanet.com/cdns/telefonica/

Cedexis. (2014). Cedexis Radar. http://www.cedexis.com/radar/

Cisco. (2014). Cisco Technology Radar Trends. Retrieved from http://www.cisco.com/c/dam/en/us/solutions/collateral/trends/tech-radar/tech-radar-trends-infographics.pdf.

Conboy, C. (2014). Front End Optimization for Developers. Retrieved from https://developer.akamai.com/stuff/Optimization/Front_End_Optimization.html

Deutsche Telekom. (2012). Strategic partnership for Cloud Computing: T-Systems to offer customers

Formatted: Dutch (Belgium)

VMware vCloud Datacenter Services. Retrieved from http://www.telekom.com/media/enterprise-solutions/129772

Deutsche Telekom. (2014). CDN solution. Retrieved from http://www.telekom-icss.com/cdnsolution

Dharwadkar, P. (2011). Network Positioning System, Cisco on-line presentation. Retrieved from http://www.ausnog.net/sites/default/files/ausnog-05/presentations/ausnog-05-d02p01-pranav-dharwdkar-cisco.pdf

Edgecast. (2014). Application delivery network. Retrieved from http://www.edgecast.com/services/adn/

ETSI. (2013). Network Function Virtualisation; Architectural Framework. *ETSI GS NFV 002, V1.1.1*. Sophia Antipolis, France: ETSI.

ETSI NFV Wiki. (2014). Ongoing PoCs. Retrieved from http://nfvwiki.etsi.org/index.php?title=On-going_PoCs

Fettweis, G.P. (2014). The Tactile Internet: Applications and Challenges. *IEEE Vehicular Technology Magazine*, 9(1), 64-70.

Frank, B., Poese, I., Lin, Y., Smaragdakis, G., Feldmann, A., Maggs, B., Rake, J., Uhlig, S., & Weber, R. (2013). Pushing CDN-ISP Collaboration to the Limit. *SIGCOMM Computer Communications Review*. 43(3), 34-44.

Ghodsi, A., Shenker, S., Koponen, T., Singla, A., Raghavan, B., & James Wilcox. (2011). Information-centric networking: seeing the forest for the trees. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks (HotNets-X)* (pp. 1-6). New York, NY, USA: ACM.

GreenICN. (2013). Architecture and Applications of Green Information Centric Networking. Retrieved from http://www.greenicn.org/.

Huawei . (2013). 5G: A Technology Vision. Retrieved from http://www.huawei.com/5gwhitepaper/.

ICNRG. (2014). Information-Centric Networking Research Group (ICNRG). Retrieved from https://irtf.org/icnrg.

IEEE. (2011). IEEE Standard for the Functional Architecture of Next Generation Service Overlay Networks. In *IEEE Std. 1903-2011*, New York, NY: IEEE.

Incapsula, (2014). Application delivery from the cloud. Retrieved from http://www.incapsula.com/cloud-based-application-delivery.html

Lee, S., & Kang S. (2012). NGSON: features, state of the art, and realization, *IEEE Communications Magazine*, 50(1), 54-61.

Lee, Y., Bernstein, G., Dhody, D., & Choi, T. (2014). ALTO Extensions for Collecting Data Center Resource Information. In *IETF draft-lee-alto-ext-dc-resource-03*.

Limelight. (2014). Orchestrate performance. Retrieved from http://www.limelight.com/services/orchestrate-performance.html

Formatted: Dutch (Belgium)

Menychtas, A. (Ed.). (2010). Updated Final version of IRMOS Overall Architecture. In *Interactive Realtime Multimedia Applications on Service Oriented Infrastructures Deliverable D3.1.4*. Retrieved from http://www.irmosproject.eu/Files/IRMOS_WP3_D3_1_4_NTUA_v1_0.pdf.

Named Data Networking. (2013). What is NDN? Retrieved from http://named-data.net/2013/07/03/what-is-ndn/.

Named Function Networking. (2013). Retrieved from http://named-function.net.

Orange. (2012). Orange and Akamai form Content Delivery Strategic Alliance. Retrieved from http://www.orange.com/en/press/press-releases/press-releases-2012/Orange-and-Akamai-form-Content-Delivery-Strategic-Alliance

Poese, I., Frank, B., Ager, B., Smaragdakis, G., Uhlig, S., & Feldmann, A. (2012). Improving Content Delivery with PaDIS. *IEEE Internet Computing*, 16(3), 46-52.

Poese, I., Frank, B., Smaragdakis, G., Uhlig, S., & Feldmann, A. (2012). Enabling content-aware traffic engineering. *SIGCOMM Computer Communications Review*, 42(5), 21-28.

Qiang, D., Yuhong, Y., & Vasilakos, A.V. (2012). A Survey on Service-Oriented Network Virtualization Toward Convergence of Networking and Cloud Computing. *IEEE Transactions On Network and Service Management*, 9(4), 373 - 392.

SAIL - Scalable and Adaptive Internet Solutions. (2011). Network of Information. Retrieved from http://www.sail-project.eu/about-sail/netinf/

Scharf, M., Voith, T., Roome, W., Gaglianello, B., Steiner, M., Hilt, V., & Gurbani, V.K. (2012). Monitoring and Abstraction for Networked Clouds. In *Proceedings of the 16th International Conference on Intelligence in Next Generation Networks (ICIN)* (pp. 80-85). Berlin: IEEE.

Seedorf, J., & Burger, E. (2009). Application Layer Traffic Optimization (ALTO) Problem Statement. In *IETF RFC 5693*.

Telecompetitor. (2011). AT&T Intros New Cloud Based CDN Services. Retrieved from http://www.telecompetitor.com/att-intros-new-cloud-based-cdn-services/

Trossen, D. (Ed.). (2011). Conceptual Architecture: Principles, Patterns and Sub-components Descriptions. In *Publish Subscribe Internet Technology FP7-INFSO-ICT-257217 Deliverable D2.2*. Retrieved from http://www.fp7-pursuit.eu/PursuitWeb/.

Xylomenos, G., Ververidis, C., Siris, V., Fotiou, N., Tsilopoulos, C., Vasilakos, X., Katsaros, K., & Polyzos, G. (2013). A Survey of Information-Centric Networking Research. *IEEE Communications Surveys & Tutorials*, 16(2), 1024 -1049.

ACRONYMS

5G	Fifth Generation
ADC	Application Delivery Controller
ADN	Application Delivery Network

ALTO	Application Layer Traffic Optimization
AS	Autonomous System
BPEL	Business Process Execution Language
CCN	Content Centric Network
CDN	Content Distribution Network
CGNAT	Carrier Grade Network Address Translation
DNS	Domain Name System
DPI	Deep Packet Inspection
EC2	(Amazon) Elastic Compute Cloud
EPC	Evolved Packet Core
EPG	Electronic Program Guide
FUSION	Future Service Oriented Networks
GPU	Graphics Processing Unit
ICN	Information Centric Networking
IMS	IP Multimedia Subsystem
I/O	Input/Output
IRMOS	Interactive Real-time Multimedia Applications on Service Oriented Infrastructures
IRTF	Internet Research Task Force
ISP	Internet Service Provider
LAN	Local Area Network
LTE	Long Term Evolution
MANO	Management and Orchestration
NaaS	Network as a Service
NFV	Network Function Virtualisation
NGSON	Next Generation Service Oriented Network
PaDIS	Provider-Aided Distance Information System
PID	Provider-defined Identifier
QoE	Quality of Experience
QoS	Quality of Service
RDP	Remote Desktop Protocol
SaaS	Software as a Service
SCN	Service-centric Networking
SDN	Software Defined Network
ServiceID	Service Identifier
SLA	Service Level Agreement
VM	Virtual Machine
VNC	Virtual Network Computing
VSN	Virtual Service Network

[i] Accounting function is also typically employed by CDNs, but we omit this block being that it is secondary in our context.

[ii] Other ICN designs have also been proposed, e.g., DONA and Curling, however, we omit them here for being interested in the main principles of ICN rather than in reviewing the whole domain.

[iii] Routing and forwarding are two distinct functions although, informally, they are often grouped together and the unit is referred collectively as a *router*. This convention is used throughout this chapter.