Future Service Oriented Networks

www.fusion-project.eu

Deliverable D2.1

Service-centric networking architecture, security issues and initial interface specifications

Public Report, Version 1.1, 22 December 2013

Authors

UCL	David Griffin, Miguel Rio, Raul Landa
ALUB	Frederik Vandeputte, Luc Vermoesen
TPSA	Dariusz Bursztynowski
SPINOR	Folker Schamel, Michael Franke
IMINDS	Pieter Simoens, Piet Smet

Abstract: This document presents an architecture for service-oriented networking as envisioned by the FUSION project. Large numbers of service nodes are 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. Given this rich set of resources, FUSION will enable services to be flexibly deployed over this distributed service-execution platform and will optimise the placement of service instances according to the performance requirements of the application, the location of its users and according to the experienced demand. At the network level a service-anycast capability is being developed so that service instance selection can be optimised on the grounds of proximity and network/server load.

The document introduces the advantages and benefits of adopting a service-oriented approach for a range of applications/service use cases, drawing the set of requirements for the system behaviour. The overall system architecture is defined, identifying the key functional blocks and the interfaces between them. A range of possible business models are introduced highlighting the interactions between the different actors and roles involved in the provision and operation of services. A set of security threat models are analysed to identify security issues beyond those applicable to general cloud computing systems. Finally, related architectures, technologies and projects are reviewed highlighting the relevance to the FUSION system and identifying the key differences of the FUSION approach to service-oriented networking.

Project funded by the European Union under the
Information and Communication Technologies FP7 Cooperation Programme
Grant Agreement number 318205

EXECUTIVE SUMMARY

This document presents the FUSION architecture for service-oriented networking. Large numbers of service nodes are 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. Given this rich set of resources, FUSION will enable services to be flexibly deployed over this distributed service-execution platform and will aim to optimise the placement of service instances according to the performance requirements of the application, the location of its users and according to the experienced demand. At the network level a service-anycast capability is being developed so that service instance selection can be optimised on the grounds of proximity and network/server load.

The document begins by introduces the advantages and benefits of adopting a service-oriented networking approach from business perspectives as well as on technical grounds including optimality, dynamicity, flexibility and scalability for a range of applications/service use cases, drawing the set of requirements for the system behaviour. The defined service use cases include highly demanding, high performance and personalised applications that would be difficult to deploy in today's cloud computing infrastructures.

The overall FUSION system architecture is specified, identifying key functional blocks and the interfaces between them. Three layers are defined: the execution plane, the service routing plane and an underlying IP networking infrastructure. Three main architectural components are specified: the ***FUSION orchestrator*** responsible for managing FUSION services and their fine grained placement in a set of distributed ***FUSION execution zones***, and the ***FUSION service routers*** which resolve and route queries and service invocation requests from users to service instances. The three main architectural components are elaborated through an analysis of service and execution layer functions and a decomposition of the FUSION orchestration architecture. The service networking architecture is elaborated, covering naming and addressing issues and service routing deployment options. An overlay-based service routing architecture is identified as the best candidate on the grounds of meeting the requirements of serviceID-based routing in a scalable and cost-effective manner. The document specifies its initial software architecture for implementation and defines the interfaces and APIs for the FUSION system.

A range of candidate business models for the FUSION ecosystem are introduced highlighting interactions between different actors and roles involved in the provision and operation of services. Two business model options are selected as the basis for the next phase of project work: an ISP-centric model and an OTT model with a common service routing plane.

Security threat models are analysed to identify security issues to be addressed by the FUSION service-oriented networking system beyond those applicable to general cloud computing systems.

Finally, related architectures, technologies and projects are reviewed highlighting the relevance to the FUSION system and identifying the key differences of the FUSION approach to service-oriented networking. The approach of the project is positioned with respect to grid and cloud computing systems, specific cloud platforms such as OpenStack, NaaS for cloud and networking convergence and other architectures for service-aware networks including IRMOS, NGSON, IMS/SIP-based architectures and information/content centric networking approaches such as NDN, PSIRP/PURSUIT. NetInf, Serval and XIA.

This deliverable presents the initial specification of the FUSION system – the service architecture and service networking are being studied in WP3 and WP4 respectively and a more detailed description of the functions, algorithms and protocols are available in deliverables D3.1 [D3.1] and D4.1 [D4.1]. The initial overview of the interfaces and interactions specified in this document will be refined in deliverable D2.2 in year 2 of the project. The final system architecture will be documented together with an evaluation of the cross-layer models in deliverable D2.3 in year 3 of the project.

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Document overview

This document presents an architecture for service-oriented networking as envisioned by the FUSION project. The document introduces the advantages and benefits of adopting a service-oriented approach for a range of applications/service use cases, drawing the set of requirements for the system behaviour. The overall system architecture is defined, identifying the key functional blocks and the interfaces between them. A range of possible business models are introduced highlighting the interactions between the different actors and roles involved in the provision and operation of services. A set of security threat models are analysed to identify security issues beyond those applicable to general cloud computing systems. Finally, related architectures, technologies and projects are reviewed highlighting the relevance to the FUSION system and identifying the key differences of the FUSION approach to service-oriented networking.

This deliverable presents the initial specification of the FUSION system – the service architecture and service networking are being studied in WP3 and WP4 respectively and more a more detailed description of the functions, algorithms and protocols are available in deliverables D3.1 [D3.1] and D4.1 [D4.1]. The initial overview of the interfaces and interactions specified in this document will be refined in deliverable D2.2 in year 2 of the project. The final system architecture will be documented together with an evaluation of the cross-layer models in deliverable D2.3 in year 3 of the project.

1.2 Problem Statement

FUSION will enable highly-demanding and personalised services to be flexibly deployed across the Internet; services that depend upon real-time processing of high-bandwidth streams with very low-latency to large numbers of geographically distributed users. Data centres and cloud-computing infrastructures have not been designed with such decentralised, bandwidth/processing-intensive, real-time applications in mind. In-network processing nodes need to be strategically positioned closer to the users of the services they deliver to provide faster application responsiveness and to reduce traffic within and between ISPs. A new networking paradigm is required to break down the barriers between data centres/server farms and the wide-area networks that interconnect them. We envision a fusion of service deployment and execution technologies with native service-centric routing capabilities throughout the network to provide a service-oriented network ecosystem for delivering a wide range of novel data- and processing-intensive services that have so far been impossible to deploy at large scale over the Internet.

1.2.1 Service-centric networking

The Internet was originally conceived as a data communications network to interconnect end-hosts: user terminals and servers. The focus has always been 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: text, images, audio and video. 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 the quality of experience for users 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 has taken matters a stage further by routing requests for named content to caches which 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 enormous attention recently, the approach, like CDNs, is limited to non-interactive content where identical copies are distributed to multiple consumers. Cloud computing on the other hand 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 does not address the problem of fine grained localisation of processing nodes in the network between the federated clouds.

Many resource-demanding services – including personalised real-time video, games or processing of high bandwidth streams for safety and health-care monitoring – are not suited to being centralised in a relatively small number of remote data centres where high network delays and low throughput can have a serious impact of the QoE experienced by many users. Application/service providers do not have access to an infrastructure to position application logic close enough to users/data sources to meet tight QoS constraints or to avoid congested or expensive network paths for high bandwidth flows to centralised locations. Media organisation logic, media/UI rendering services and media interaction mechanisms remain centralised, hard to query, impossible to mix and inflexible to provision.

Up until now service deployment and data networking issues have always been treated in isolation by the research community as well as in the commercial reality of deployed products and services. The vision of the FUSION project is to develop a new approach of service-centric networking where the boundary between service and network layers, between cloud-computing and wide-area data communications functions is broken down to deliver seamless service-oriented networking. In the context of the FUSION project services are seen as chains of distributed data processing elements interleaved by network transmission, potentially over a wide-area. FUSION will develop a new networking architecture and protocols that are designed to natively support efficient service provisioning and execution at the service-layer combined with native service-centric routing and load balancing within the network layer for the most demanding multimedia services. Solving these issues involves a combination of service and network engineering disciplines. Dynamically constructed service execution topologies must be mapped to the underlying network and processing nodes.

The FUSION project allows an operator to make the evolution from a content-driven network architecture towards a service-centric network. It will provide for efficient service provisioning and service orchestration which in turn will enable for faster development cycles and time-to-market. An important dimension of the solution is how to decouple responsibility between applications and the underlying infrastructure to control service instantiation, routing and execution such that service requirements are met while network and infrastructure utilisation is efficient.

1.2.2 FUSION approach

FUSION foresees a situation where large numbers of service nodes are 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. Given this rich set of resources, FUSION will enable services to be flexibly deployed over this distributed service-execution platform and will 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.

To meet the performance targets mentioned above 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 and the users, the service providers or the network itself must be able to select an appropriate one. FUSION adopts a service-centric networking approach and will deploy a service-anycast capability in the network so that service instance selection can be optimised on the grounds of proximity and network load, maximising their availability. Furthermore, FUSION will develop lightweight protocols to also allow server load to be taken into account so that load balancing algorithms running in service-centric routers can discover the best service instances to route user request towards.

Cooperation between the service and network layers is key to the success of the FUSION vision and the project intends to innovate in this area too. Network-driven instantiation and replication of service components to reduce network congestion and adapt to highly dynamic fluctuations in usage patterns will be developed. In addition, FUSION will research and develop a service-node execution platform leveraging the state-of-the art in novel, container-based virtualisation technologies that simplify network-driven session instantiation, replication and migration.

1.2.3 Objectives

Objective 1: Define a combined service and network architecture that exploits the synergies between both and which provides significant improvements in latency and scalability. This will enable a new class of decentralised, high-bandwidth-and-processing, real-time applications. In addition to taking into account the roles and needs of content/service- and network/infrastructure providers, this architecture will constitute one more step towards ISPs as content distributors. This can be broken down in four technical goals:

- Design an elastic, network-driven service architecture that supports distributed processing elements interconnected by the public Internet, rather than the tightly controlled environments typical of current data centre architectures. This architecture will support dynamic service instantiation according to detected/monitored user demands (including flash crowds, service urgency and interactivity). In addition, this architecture will be versatile enough to scale elastically with heterogeneous resources (networking, servers and specialised hardware for service processing acceleration).
- Design interfaces between the application and the platform provider which are rich enough to accurately convey application requirements, yet simple and abstract enough to apply to all applications and be usable by all application providers.
- Design and evaluate a scalable service naming scheme that allows intelligent optimisation of routing at both the network and service management layers.
- Design a migration path from current cloud architectures and towards our proposed network-driven service architecture.

The metrics by which the success of the project can be assessed against this objective and technical goals are: whether the architecture can deal with dynamically growing or shrinking service component instances; the scalability of the solution with respect to the number of component instances and the number of attributes needed to convey application requirements.

Objective 2: Provide new network protocols that improve the performance of the aforementioned architecture compared to current state-of-the-art infrastructure for demanding applications requiring large amounts of computation and bandwidth resources over a wide geographic area. These protocols will operate at both network and service layers and within the requirements set by

the application, and will be designed to be cost-effective and respond to changing network and/or infrastructure availability conditions. This objective can be broken down in three technical goals:

- Design a routing protocol that selects the best available instance for a user session. This protocol will perform load balancing between user requests, and ensure that a user session is consistently routed to the same service instance.
- Research the design of hybrid link-state/distance-vector routing protocols. To this end, we will carry out a comprehensive evaluation of compact routing algorithms for service routing. The resulting trade-offs will drive the design of novel, hybrid routing schemes using complementary routing algorithms for different endpoints.
- Design network layer protocols to enable the efficient registration and aggregation of service instance status updates distributed over the network.

Verifiable metrics for this objective: the accuracy of selecting optimal service instances; the load dispersion across multiple instances of the same component; the routing overhead and stretch; and the amount of service data that needs to be maintained according to the frequency and quantity of routing updates.

Objective 3: In order to support our network-driven service architecture, computation elements themselves will have to be significantly improved. In addition to markedly reducing the network impact of its distributed computations, our service architecture will be able to flexibly manipulate virtual computation units, provide personalised services and deal with complex service definitions. This can be broken down in four technical goals:

- Design intelligent service placement algorithms that dynamically adapt to make efficient use of resources at both the network and service layers, i.e. service instantiation will be optimised by considering network and computational resource availability. This will be achieved by taking workflow representations of all required services and mapping them to the available network and computation resources.
- Design lightweight, container-based resource isolation techniques for the instantiation of services with appropriate CPU, RAM, storage, and network resource guarantees. These container-based isolation techniques will apply to heterogeneous hardware systems and provide low start-up and tear-down overhead and application response times.
- Define techniques to provide increased flexibility and efficient network support for personalised services.
- Design a service description language that includes non-functional service constraints and an orchestration language that is able to describe an application in terms of service components and interactions.

Verifiable metrics for this objective: the network and service performance metrics (e.g. response time) delivered by servers/service instances at the selected locations; the overhead (memory, CPU load, etc.) of the selected lightweight virtualisation techniques; the time to establish and tear-down service instances; the number of parameters required to capture personalisation profiles.

1.2.4 Expected results

In order to meet the aforementioned objectives, FUSION will concentrate on research advances in the areas mentioned below to produce the following results:

- ***A new service-centric paradigm for the future Internet architecture***, meeting head-on the challenges beyond content caching and enabling a fundamental migration from content to services, facilitating the deployment and delivery of low-latency, data- and processing intensive,

interactive and personalised services to be deployed at a large scale over a public service-centric Internet.

- ***Native network support for service routing protocols*** based on service names rather than locators, with ***service location- and instance-independent naming and addressing schemes*** as a side result.
- A ***service-oriented anycast capability inherently supported by the networking infrastructure*** so that service invocation is no longer tied to addressing specific servers. This will enable replication and load-balancing, and will provide the flexibility of dynamic instantiation of service instances, therefore providing support for elasticity-on-demand directly in the network substrate.
- ***Algorithms to optimise service instance placement***, allowing processing capabilities to be placed where they are needed: close to users and in strategic network locations, rather than being limited by the extremely coarse-grained localisation afforded by today's cloud computing infrastructures. Fine-grained targeting of service instance placement reduces latency, improves bandwidth, and reduces network load, ultimately benefiting both service providers and end users.
- ***An architecture, platform and APIs to enable service node resources across multiple infrastructure providers*** to be seamlessly made available to application developers through a platform provider who supports the FUSION dynamic service deployment and localisation algorithms.
- Innovative approaches to ***bridge the gap between data-centre and Internet interworking***. Internal data-centre routing and resource allocation strategies cannot be simply applied in the wide-area Internet due to constrained resources and a lack of knowledge of inter-data-centre capabilities. Furthermore, since their only tool at the present is a limited form of limited cloud federation for virtual machine replication, current solutions are unable to directly address network capacity shortages by dynamically deploying service processing resources.
- A framework for ***cooperation between network and service layer algorithms and functions***, including the definition of specific lightweight protocols to inform the network layer of server capabilities and dynamic load, and allowing network-driven hotspot detection and service-component instantiation.
- Protocols and algorithms to enable ***interworking across provider boundaries***, including scalable approaches based on compact routing to allow service instance state to be conveyed between provider domains.
- Service execution platform technologies based upon ***simple and lightweight container-based virtualisation means running over heterogeneous hardware***. This will reduce the overhead for dynamic instantiation of processor- and data-intensive service-component instances, facilitating a move from the cloud hosted web applications of today to processing and bandwidth intensive real-time services.
- ***Service deployment mechanisms across widely distributed nodes***, including programming interfaces and APIs to convey complex service manifests in terms of service deployment graphs that capture service constraints and performance targets.
- ***Innovative use cases deployed in real testbed environments*** to evaluate and demonstrate the interoperation of service localisation algorithms, dynamic service-component instantiation and native service-centric routing for representative services.

1.2.5 Benefits of the FUSION approach to service-oriented networking

1.2.5.1 Business aspects

- Operators are no longer “dumb pipe” providers, but offer service hosting capacity with additional advantages: geolocality, low-latency.
- Lower costs for operation and maintenance since this can be done centralised by larger service provider and distributed devices, not a central server at the service provider's office basement (lowers barrier for smaller players to roll out their own services without having to invest in their own infrastructure). While this is a general feature of cloud computing, FUSION also adds more flexibility, locality, reduced latency, etc.

1.2.5.2 Optimality/efficiency

- Bandwidth reduction (by smarter placement). FUSION increases degree of b/w reduction compared to today's coarser-granularity DCs.
- Response time/latency (and jitter) reduction (for better QoE with respect to interactive applications). FUSION increases degree of latency reduction compared to today's coarser-granularity DCs.
- Fine-grained server instantiation/selection taking into account the clients' location (and the service instance if it is a stateful service).
- Automated optimisation of b/w, latency, etc. (at deployment and run-time).
- Capability to exploit appropriate accelerators (GPUs, encoders, etc.) FUSION’s added value in this area includes more types of hardware accelerator, smarter awareness and selection of accelerators, resource description, APIs etc.
- Capability to efficiently share sources and resources (stored 3D objects, textures, decoded video frames, GPU buffers, transcoding function, subtitling service, etc.). These are new FUSION features - through service manifest definition, orchestration techniques. Current APIs and software packages are too abstract, high-level or low-level to provide the right handles to the resources.

1.2.5.3 Dynamicity, flexibility and scalability

- Flexible deployment infrastructure (more suitable/fine grained for real-time, interactive applications than today's clouds). FUSION increases the degree of flexibility compared to today's coarser-granularity DCs.
- Automatic service migration (or load balancing between instances) to rebalance performance/processing depending on the number of concurrent players or users connected to the game or conference call, and their location.
- Scaling running service instances up or down, based on the demand (short-term) and success (long-term) of the application. While this is also applicable of today’s cloud infrastructures FUSION orchestration algorithms will enable service scaling across multiple execution zones for service scaling on a global scale.
- Client/user mobility (migrate service as users move). It should be noted that FUSION will work on algorithms/heuristics and not migration mechanics/technology.
- JIT deployment (cf. situation-based services in FRV, games), even triggered at service-request time. FUSION will work on algorithms/heuristics and evaluation/measurements of dynamic deployment technology.

- Dynamic application behaviour: rapidly changing interactions between sources, clients and services.
- Resilience/robustness: to congestion, failures, etc. at both network and service levels. Through server replication and the use of a network anycast (and/or late binding) paradigm for server selection/routing. FUSION will provide scalability, load balancing etc. compared to today's solutions, e.g. CloudFlare.

1.2.5.4 Distributed aspect

- Enable smarter placement: reduce roundtrip times, balance load, save bandwidth, etc. These are new FUSION features, thanks to finer-grained execution zones, smart deployment, scalability and load balancing.

1.2.5.5 Offer performance/QoE guarantees/SLAs to application developers

- Finer grained description of deployment constraints, including dynamic aspects.
- Guarantees covering network as well as the execution platform performance.

2. USE CASES

In this section, we will introduce the types of applications we want to support with the FUSION architecture and platform, and describe a number of key use cases we will focus on in this project. We will use these use cases to extract a number of key attributes and requirements that should be taken into account when designing the FUSION architecture.

2.1 Personal Media Experience

2.1.1 Media Dashboard

A core trend in digital media consumption is that PCs, DVD players and consoles replaced more and more by mobile devices like tablets or smart phones.

Another trend in digital media consumption is the growing importance of unified access to different media, for example accessing Video on Demand (VoD), music, TV, a private photo library or games in one place. For example, unified UIs like the Xbox 360 dashboard[1] where many media types are in one place are today's standard and expected especially from the younger generation of users. Also the visual quality of such dashboards will continue to improve. For example 3D graphics using 3D rotations and high-quality rendering effects like particle effects are today's quality standard both on consoles and mobile devices.

Medias are often consumed together: Families, couples and friends enjoy watching movies or browsing private photos together or play games together. For the future we expect that shared live media experience expands also to people being at different locations. Already today many people play multi-player games together remotely. It is natural that people also want to watch movies together or browse a private photo library when not being at the same location.

For example, a girl-and boyfriend are either sitting on their sofa together at home, or they may be at different locations but want to share their evening together. In the latter case they may start not only a via video chat, but also enter a shared virtual 3D room dashboard:

- She can put her today's photos into that shared 3D room.
- They browse together the photos and talk about them via video chat.
- She can point out particular photos or persons on these photos by highlighting them in the shared 3D room.
- They browse together the episodes of their favourite series and choose one.
- They watch a streamed series episode together while video chatting about the episode.
- They play a cooperative multi-player game together.
- They watch a friend playing a game.

The following image is a mock-up of such a media consumption dashboard:

[1] https://www.youtube.com/watch?v=3zOX7_67dIQ

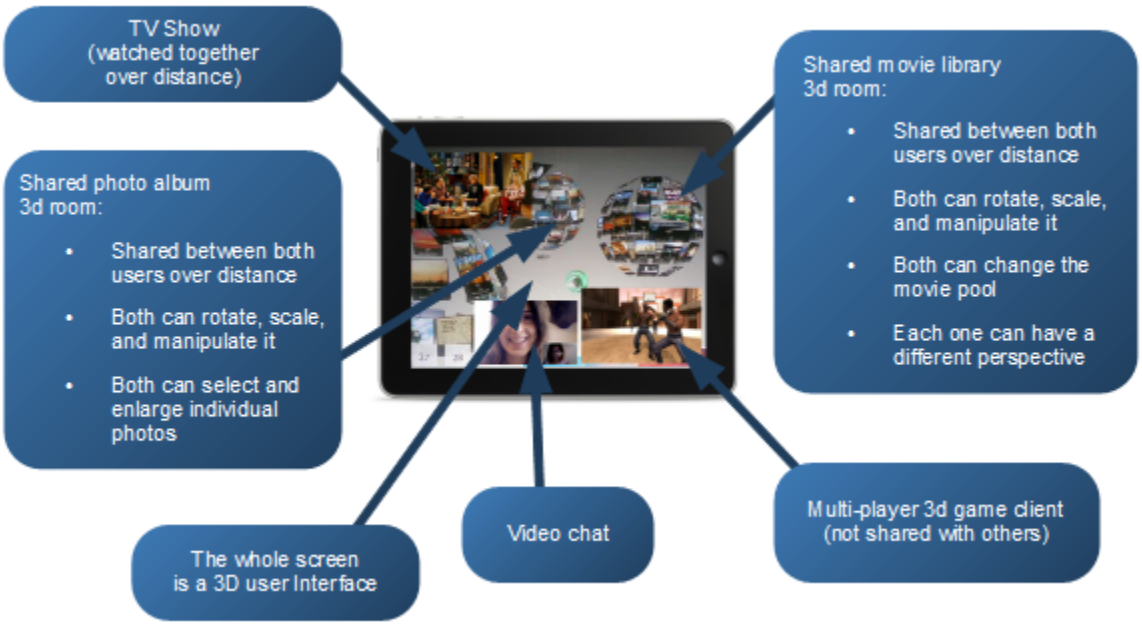


Figure 1: Media Dashboard Example

FUSION can help in multiple ways to let these scenarios become reality:

- First, running a virtual 3D room application displaying multiple different video streams at an end user's device may be impossible or at least inefficient because these streams have to be delivered all the way to the device and therefore allocate a lot of bandwidth. By moving the rendering of the UI nearer to the sources and only sending the result over the network, bandwidth consumption can be reduced and also devices with lower bandwidth connection would become reachable. Furthermore, depending on the application or service used, parts of the computations and rendering steps could be reused by other users (simple example would be a high definition to low definition conversion service of a popular TV channel), so that the amount of resources can be significantly reduced.
- Secondly, many of these device (smart phone, tablet, TV, STB, etc.) are limited in terms of compute and rendering capabilities. Having many complex interactive (3D) applications running smoothly on the device itself is in some cases infeasible.
- The above mentioned benefits require on the other hand to intelligently choose an appropriate location to perform the processing; the more naïve approach to deploy these services on a centralised data centre or cloud will typically result in unacceptable response times as well as jitter, depending on the distance, the network configuration and contention.. So the key will be to find the optimal location in the network in order to have a good balance between response times and compute capabilities.
- By offering the possibility of dynamic service orchestration, the required services can easily be installed and instantiated on demand. So if for example one user wants to watch a show on a less powerful mobile device with a low screen resolution, an automatic downscaling service can be inserted in the network between the video source service and the user's device.
- Common standardised interfaces allow for easier development of different kinds of services, be it a simple video streaming service, a request and response news ticker or a complex game. By having a common architecture the interoperability of these services is significantly improved, providing easier integration and therefore higher efficiency in development thus shorter time-to-market.

- Another benefit on business side is the dynamic allocation of already existing resources: A service operator does not have to buy or rent server infrastructure front up for a certain time but can directly deploy to the network which itself chooses the best location to do the work. It is even possible to only tell FUSION where to find the resources and the orchestration decides when to start up new servers or scaling is required. So the distribution and maintenance costs may be much lower, allowing also small and medium size enterprises to distribute their services (similar to Apple's iPhone where the entrance hurdles are also very low).

2.1.2 Omni-view

In the omni-view scenario, a user can virtually look around and walk through a particular event (e.g., the Olympics, a soccer game, cycling, a music festival, etc.), allowing the user to completely immerse himself in the experience. High-resolution and ultra-high resolution videos from multiple angles and locations are seamlessly stitched together and augmented with additional information in real-time. Each user can personalise his experience of the event by either manually controlling the scene, or by letting a virtual director create an experience specifically for that user, based on his preferences. This may include tracking a specific player or athlete in detail, overlaying the scene with specific background information, etc.

Figure 2: Omni-view example [FASCINATE]

Streaming all relevant high-resolution video feeds to all end devices (smart phones, tablets, TVs, etc.) and seamlessly blending them into an augmented virtual 3D environment across multiple screens on these devices themselves is clearly infeasible. By doing all this blending and stitching in the network, the high-resolution feeds as well as several rendering steps can be shared across many users, saving huge amounts of bandwidth as well as compute resources. Running these applications in the network also enables significantly more complex rendering operations, resulting in a highly immersive experience. A crucial aspect regarding this immersive experience however is smooth interactivity; the end user should be able to interact with the virtual scene as if it was running locally.

FUSION contributes to this scenario in several ways, including the following:

- By optimally placing the services in the network with respect to the video feeds, the other services as well as the end users, both the bandwidth as well as the latency of these high-resolution feeds can be drastically reduced.
- By placing the rendering service close to the end user in combination with the FUSION routing capabilities, FUSION enables a smooth interactivity by creating a low latency communication channel between the rendering service and the devices of the end user. Also, if the user travels to another location, he will continue enjoying the same smooth experience.
- As this scenario requires a large of advanced dedicated functionality (3D rendering, stitching, video encoding and decoding, etc.), FUSION can easily take these requirements into account and map these services on execution zones that provide these advanced functions through specific accelerators (GPUs, hardware encoders, etc.).

- Because of the temporary nature of the events to which this scenario applies, it is crucial that the corresponding services can be deployed and removed quickly in large amounts. Indeed, the services are only required for the duration of the event, which could range from a few hours once a day, to a few days once every few years, but when they are active, they could easily need to serve several millions of users (e.g., the Olympics) at the same time.
- The resource-demanding nature of this scenario in combination with this extreme time-varying behaviour would make it extremely expensive for individual organisations to each support and provide the necessary infrastructure to support this use case. However, many of these events will basically use the same set of resources and share the same set of requirements. Consequently, having a common deployment infrastructure that is capable of handling these events would be extremely beneficial to these parties, as they no longer have to worry anymore about building and managing such an infrastructure themselves.

2.1.3 Immersive Communication

Similarly to the omni-view scenario, the goal of this scenario is to provide a more immersive communication experience between multiple parties, which may be spread all across the world. Whereas classic video conferences, including the telepresence conferences, are typically very static and limited, the goal of this scenario is to blend the (3D) camera feeds from all parties into a very dynamic virtual environment, possibly augmented with additional information (e.g., a slideshow is integrated in the virtual scene). The virtual environment is constantly modified, for example according to who is talking, non-verbal expressions as well as gestures. Additionally, the entire scene can be fully personalised either manually or automatically for each individual user, including users that only watch the conference without actively participating.

Figure 3: Immersive communication example

Although, this scenario shares many commonalities with the omni-view scenario, there are a few key differences. First, though the camera feeds may also be plentiful and in high-resolution, the requirements will typically be an order of magnitude lower than in a full-fletched omni-view scenario.

On the other hand, the synchronisation of all feeds and all rendered scenes across all parties will likely be much more important in this scenario than in the omni-view scenario, especially as the feeds will be coming from remote locations across the globe, rather than from one particular event.

The contributions of FUSION with respect to this scenario are similar to those from the omni-view scenario, though they differ in a number of ways:

- Although this scenario has similar on-demand requirements, the scales are completely different. In the omni-view scenario, there will typically be a limited number of medium-sized to big events

where up to millions of users are connected to, whereas in this scenario, there will typically be many sessions running in parallel with each other, but each involving only a handful of users, which may still be spread across the globe.

- As the real-time feeds from each user needs to be blended seamlessly in one virtual environment, and as these feeds may come from all across the globe, placement of each service involved in a particular session, as well as the routing and sharing of these feeds is of extreme importance. This differs fundamentally from WebRTC for example, where all feeds need to be sent to all other parties, limiting the quality and amount of people that can participate in one (let alone more) conference session(s).

2.1.3.1 Multi-Screen

The multi-screen scenario can relatively easily be applied to almost all of the other scenarios where all the rendering is done by FUSION services in the network. The multi-screen scenario means that one or more end users can connect to the same service via multiple screen devices. There exist two basic models for doing multi-screen. The simplest model is the screen replication model, where the same output is send to multiple screen devices, possibly downscaled and in a different format. The second model is the side-channel model, in which the service has more than one rendering output on to which different screen devices can be attached. Although this has similarities with the omni-view scenario, the sharing here is often done more on a personal level and each rendered output may contain very different information. For example, the main output may be overall soccer game, whereas the second output may be background information regarding one or more players.

2.1.3.2 Real-World Tagging

In this use case tourists use an app that overlays touristic information about the observed monuments and events on their head-mounted device. The video feed from the front camera on the wearable device is streamed to object detection algorithms running on cloud infrastructure at the network edge, where the appropriate content and position of the overlay items is calculated and returned to the wearable device (see Figure 4). The service is extremely personalised, and tailors the overlay content to the user's profile. For example, it shows hints and tips of friends who have visited this location in the past and added tags to monuments.

Figure 4: A Real-Life Tagging Application [PAJA11]

Network operators have anticipated on the massive demand for mobile bandwidth by WhatsApp messages, tweets, Facebook sharing and the video upload by deploying additional mobile 4G antenna towers. Previous experience has however learned that adding towers alone won't meet the data demand, so they have further increased capacity through local Wi-Fi hotspots and femtocells deployed in residential set top-boxes [QSC13]. Network management automatically redirects the user to the most appropriate interface. Switching between these different technologies, user movement patterns and wireless channel contentions result in continuously varying latency and bandwidth of the device-cloud connection.

The FUSION platform monitors these network statistics. As the cloud infrastructure of the tourist guide has limited capacity, new service instances are deployed according to the demand. For fast deployment, the service instances are not identical replicas, but only contain the data of nearby tourist attractions. The FUSION routing ensures that each user is routed to the closest replica to meet stringent latency requirements.

2.2 Public Safety

2.2.1 First Responder Video

The public safety ecosystem is currently be revolutionised by the introduction of LTE for first responders [MOBI11, NETW11]. The use of broadband mobile connectivity allows connecting more video feeds into the public safety information system, and allows connecting more users to those feeds (control rooms, local command centres and police/fire-fighting/ambulance first responders). Some feeds can even be deployed for a shorter period of time (e.g. during a major sports event or demonstration) and other users could be connected as well (e.g. volunteer stewards for crowd control).

With the connectivity being solved, the next challenge is to translate all those raw video feeds and interaction into relevant information. A policeman doesn't have the time in the midst of a critical situation to browse through some menus to get access to the right camera feed showing the right tactical information to assess the situation, nor is it possible to constantly monitor all feeds manually. We thus need a range of analytics that can analyse these feeds, identify and signal issues to the relevant parties when they occur.

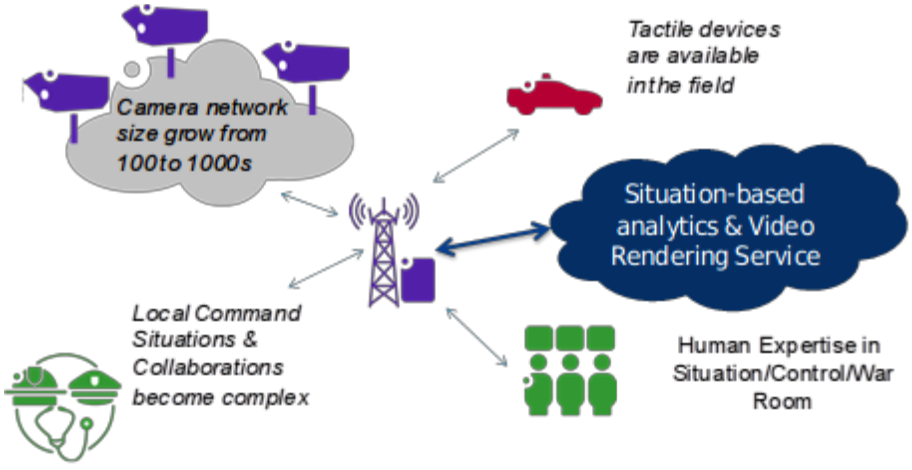


Figure 5: First Responder Video Scenario

For this reason it is important to be able to deploy 'situation-based' services. These situation-based services combine the deployment of video analytics functionality (crowd/traffic analysis, license plate recognition, smoke detection etc.) with video rendering and content augmentation, to optimally reduce the cognitive load for first responders and present them with the relevant information.

Some important behavioural aspects of this use case are (i) that the set of video streams, first responders, control centres, required analytics, access to archived data, etc. constantly change from incident to incident, (ii) that the occurrence and size of these incidents is unknown beforehand, and (iii) that the network capacity (including the bandwidth available in the mobile cells), plays a crucial role in selecting the most optimal set of streams.

Evidently, this is an environment where a scalable, dynamic FUSION platform will play a vital role. As situations can change quite rapidly, different teams need to collaborate on diverse video feeds, analytic functions need to be inserted quickly in the processing pipeline and different compositions and augmentations of the videos are needed. This cannot be solved with the traditional coarse-grained service platforms currently available. Also, the network-aware routing functionality of FUSION will be key to ensure smooth interactivity across multiple mobile networks. Furthermore, as networks in this market are often private and dedicated, this will allow for an easier migration path to include FUSION functionality.

2.2.2 Surveillance

This scenario involves the continuous monitoring of key public buildings and places, squares, roads, etc. and involves a continuous monitoring of a fixed number of locations in near real-time. Although by itself this class of use cases has limited need for many FUSION-specific features it could take advantage of the accelerated hardware functionality for low-power and high-efficiency processing of the multitude of surveillance cameras and they could trigger the first-responder scenario, for example when a hazard is detected.

2.2.3 Face Recognition

In this scenario, the face of a person is recognised automatically when that person wants to enter or leave a building or a room. A picture of the person is taken and uploaded to the instance that is handling the request, in combination with the identification of the building or room. To keep the time to recognise the service short, FUSION will automatically route the request to the closest available instance running in the network. The service running in the network has direct access to the entire database that is relevant for that location, resulting in a fast processing of the request. The service sends back the result of the request to the face recognition device, which then either grants or forbids access to the location.

Some key benefits that FUSION will provide is the smart routing to the best instance that is capable of handling the request, and scale up and down when needed. This could include predeploying instances during periods of high activity (e.g., in the morning, when people enter the building when they arrive for work). FUSION will take care of all the management and infrastructure needed to deploy this type of service.

2.2.4 Major Sports Event

This use case is similar to the first-responder video scenario, but in a different setting. Example sports events include soccer games, the Olympics or other big championships. The scenario enables security and first responders to quickly identify and act on hazardous situations. Some key aspects of these scenarios are that they are very event-based (ad hoc situations), where the service only needs to be activated when the event occurs, and extra services need to be deployed very quickly and seamlessly when a hazard occurs.

2.2.5 Key Public Places

These scenarios capture the more classical surveillance use cases. However, deploying these services on the FUSION infrastructure would enable a flexible deployment of more advanced surveillance services on-the-fly, based on the analytics produced by a number of background surveillance

services. The first-responder scenario could even be triggered automatically based on automatically detected hazards.

2.3 Gaming

Typical overall characteristics include potentially a lot of heavy rendering, which requires (powerful) GPU for efficient rendering, and extremely low latency depending on the type of game (first-person shooter versus a turn-based game).

2.4 Additional Service Examples

Additional examples of possible services are presented in Appendix A: Service Components in page 109. Examples of how to compose these services in media usage scenarios can be found in Appendix B: 7 Service Configuration Scenarios in page 113.

2.5 Summary of example use cases

The table below analyses the example use cases described in the previous sections against the FUSION benefits introduced in section 1.2.5. Each use case is marked from 1 to 5 against each of the benefits/features of the FUSION approach to service oriented networking.

	Media Dashboard (EPG)	Omni-view	Immersive Comms	Multi-Screen	Real-World Tagging	Surveillance	First-Responder Video	Client-based Gaming	Cloud Gaming
* ***Business aspects***									
* Operators not "dumb pipe" providers	5	4	3	5	5	1	3	3	5
* Lowers barrier for smaller players	4	1	4	4	5	3	2	4	5
**** Optimality/efficiency***									
* Bandwidth reduction (by smarter placement)	5	5	4	5	5	3	4	2	3
* Response time/latency (and jitter) reduction	5	5	5	2-3	5	3	4	4	5
* Fine-grained server instantiation/selection	5	2	4	3-4	3	4	2	3	5
* Automated optimisation of b/w, latency, etc.	5	5	5	4	5	4	3	4	5
* Exploit appropriate accelerators	5	5	4	5	5		no	1	5
* Efficiently share sources and resources	5	3	4	5	5	2	1	2	5
**** Dynamicity, flexibility and scalability***									
* Flexible deployment infrastructure	3	4	5	5	4	3	3	4	5
* Automatic service migration (or load balancing between instances)	4	1	5	3-4	5	3	5	5	5
* Scalability	4	1	5	4-5	5	3	4	4	5
* Client/user mobility (migrate service as users move)	4	2	2	2	5	4	no	2	1
* JIT deployment	4	4	4	4	4		no	3	5
* Dynamic application behaviour	5	5	2	4	1		2	2	1
* Resilience/robustness	3	4	2	5	5	3	2	2	5
**** Distributed aspect***									
* Smarter placement	5	4	5	4	5	4	4	3	5
*Performance/QoE guarantees/SLAs to application developers	3	5	3	3-4	5	4	3	3	5
* Combination of FUSION benefits	5	4	5	4	5	3	3	4	5

Table 1: Summary of FUSION example use cases

2.6 Key Attributes and Requirements

The goal of this section is to identify a number of key attributes and requirements from the set of use case scenarios we described in the previous section, and to highlight why current infrastructures and platforms are insufficient for many of these use case scenarios.

We now derive attributes and requirements from the use cases. The Media Dashboard service is an example of a streaming service that is stateful for the duration of the session. The state could also partially be made permanent outside of FUSION so that the next time the end user reconnects to (another instance of) the EPG service, his last state is restored and the user can continue where he had left off. The sessions will typically last for minutes to hours, and in the simple case, one can expect that there is no real inter-service interaction and that all user sessions are independent from each other (so there is no real coupling apart from the various sources).

In terms of the input and output channels, there will typically be a large number of various kinds of input channels and sources (typically video streams), which can change quite dynamically and frequently, for example when the end users is flicking through the video broadcast channels. The sources themselves can be either coming from other FUSION services or come from outside of FUSION (e.g., a YouTube video). The number of output channels will typically be small (even in the multi-screen or shared viewing scenario), and will typically be a video stream, which remains active as long as the end user is connected via the corresponding device. The output is typically a non-FUSION destination (e.g., a thin client installed on a tablet or STB). The feedback channel is important to mention, it is a message-based communication channel with very strict latency requirements for sending the user interaction to the EPG service running in the network.

The key functional blocks include lots of video decoding, video encoding and (3D) rendering. One of the key requirements is a low round-trip latency between pressing a button on the end device and seeing the interaction on the screen. Depending on the type of interaction, noticeable latencies will not be tolerated by the end user.

In the same way, we can explore the key attributes and requirements of our other use cases, and extract those elements common to all of them. These are reported below.

2.6.1 Key Attributes

The subsections below identify the key attributes that define the scenarios above and reflect various aspects that are important for FUSION.

2.6.1.1 Service Type

In FUSION, we will support both the classical request-response as well as streaming services. With request-response services, the service sessions (see deliverable D3.1 [D3.1]) are often extremely short-lived and typically involve one or more request messages sent from the client to the service, the service computing a response, and subsequently sending back the response to the client.

2.6.1.1.1 Interactivity

Service sessions, especially involving streaming instances, can be either interactive or non-interactive. In the former case, the client can interact in real-time with the service instance running in a FUSION domain, thereby changing the content that is streamed back to the client. An example is the EPG scenario. In the latter case, the client cannot (directly) interact with the service instance that is running in a FUSION domain; once the request is made, the instance starts streaming the content back to the client, and the client cannot change what is streamed. An example is live streaming or adaptive streaming.

2.6.1.1.2 State

Service instances can be either stateful or stateless. With stateless instances, the client can connect to another service instance at any time, even with streaming services. With stateful instances, the maintains some session state that is specific for the session it is handling, and the client cannot seamlessly be transferred to another instance without first transferring the session state as well. Interactive streaming services will typically be stateful, whereas request-response as well as non-interactive streaming services in many cases can be made stateless. Stateless service instances are preferred, as this increases the overall service request routing and simplifies migration.

While FUSION should be as flexible as possible and avoid any strict classification of service types, it may be still useful to classify services pragmatically for analysing typical use-cases. A basic classification may be stateful versus stateless services:

Kind of service	Minimal number of service instances required for N > 0 service users	Service Replication	Service Instantiation	First request response time optimisation
Stateless	1	Technically Identical to service instantiation	Optional for better performance	Instantiate additional services
Stateful	N	Usually not needed	Required for each service user	Pre-instantiate service instances

Table 2. Stateful vs. Stateless Services

2.6.1.1.3 Session Duration

The duration of the service sessions can vary from extremely short-lived sessions to long-lived sessions that can last for minutes, hours, or even days. Request-response services with typically be short-lived, whereas streaming services will typically be long-lived, though this does not always have to be the case.

2.6.1.2 Key Functional Blocks

Many use case scenarios share one or more common key functional blocks, for which often specialised hardware can be used for higher efficiency. Examples of these functional blocks include video encoding and decoding blocks, 3D rendering, etc. Providing the necessary infrastructure for leveraging these functional blocks is an important aspect of FUSION.

2.6.1.2.1 Input and Output Channels

Services may have rely on other (FUSION or non-FUSION) services or sources for providing their services. To facilitate an optimal placement of this service with respect to its sources (or destinations), it is important that FUSION should be made aware of them. These sources (or destinations) can/should be characterised in a number of ways:

- Cardinality: how many input/output channels does this service (potentially) have;
- Type: is the inter-service communication rather message based (sporadic events, bursty, etc.), or does it involve a continuous data stream;

- Requirements: what are the inter-service communication requirements, for example in terms of latency and bandwidth;
- Dynamicity: how often do you want to (re/dis)connect to each source, or how often can other services (dis)connect;
- Scope: are the sources coming from outside FUSION or generated by other FUSION service instances (similarly for the output channels);
- Coupling: how loosely or tightly are the different services coupled to each other.

Each of these may have an impact on the placement and routing in between these services. One of the main difficulties is that in many cases, it is not known how many and where these input or output services are located.

2.6.1.2.2 Feedback Channels

A special input channel is the feedback channel, which is a communication channel in the case of interactive services between the client and the service instances, allowing the client to interact in real-time with the service instance running in a FUSION domain.

2.6.2 Key Requirements

This section discusses the key requirements FUSION should be able to support considering the use case scenarios discussed previously. The requirements have been grouped into three main categories:

2.6.2.1 User Requirements

Based on the scenarios discussed above, we drew the following conclusions:

- Digital media consumption on various devices can be implemented by connected distributed service components.
- Service components may run on same server or close servers (e.g. game client output to video encoder), or at distant locations (e.g. game client in cloud → 3D room client at ISP).
- Media views may be nested, for example a private 3D UI may contain shared 3D room, which may contain a VoD library, which may contain many live streams of different videos.
- Rendering may be done either
    - on a powerful end-user devices (e.g. PC), especially final composition or game clients, or,
    - only displayed on a less powerful end-user device (e.g. mobile devices, TV), while rendering is done on a server, and the video is streamed to the end-user device. The disadvantage of this approach is a worse responsiveness, but may provide better quality for low-performance end-user device.

Users want to use digital media as easily as possible: they prefer getting it from a single business partner (see the success of stores like iTunes). They especially don't want to care about technical details and about which others business parties are involved.

2.6.2.2 System Requirements

Many services will have both a number of specific software-related as well as hardware-related requirements. The former set includes requirements in terms of OS, drivers, libraries, etc. The latter set includes requirements like CPUs, memory, and specific accelerators (GPUs, hardware encoders, etc.).

2.6.2.2.1 Mobility

Today basically everything is mobile. People watch movies, play games, share images, are first responders, basically do everything also mobile. Mobility is not a use-case itself, but a core feature of many use-cases.

Mobility can benefit from connection handover. This may be outside the scope of FUSION but FUSION should be open for service components choosing existing solutions at own will. If FUSION would integrate such features, it may offer it as separate optional module.

Mobility can also benefit from stateful service instance migration by keeping service instances close to the end-user device, for example a game server, game client or dashboard service following the user while moving. This is not strictly required for FUSION to work, but for flexibility FUSION should not be restrictive, and therefore allow services to support migration and make it easy for service implementations to support it.

2.6.2.2.2 Service Instance Replication and Pre-Allocation

For both stateless and stateful services it makes sense to support replication (in case of stateless services) and preallocation (in case of stateful services). The following diagrams show scenarios without and with preallocation:

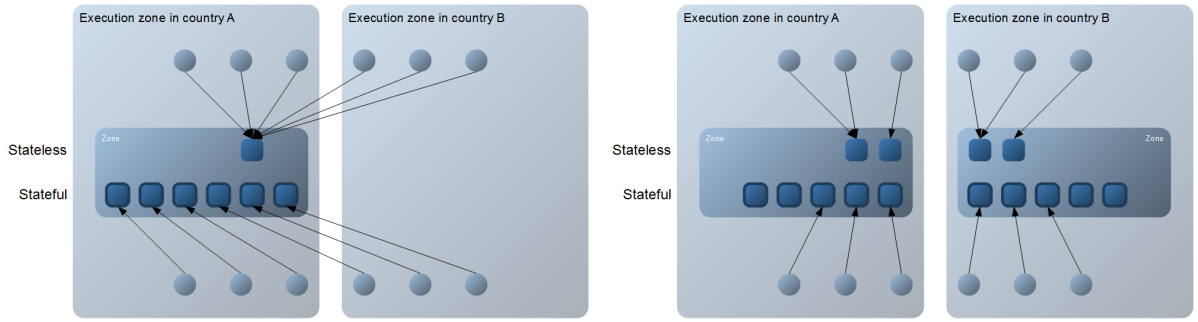


Figure 6: Preallocation Scenarios

2.6.2.2.3 Service Deployment

Deployment requires data from user-defined source (e.g. https URL to a package). At simplest the package is at exactly one location. Deployment itself is probably not performance critical for most services. Pre-deployment of possibly used services is possible.

2.6.2.2.4 Service Instantiation

Instantiation (including replication) is probably performance critical and will likely require local deployment. Pre-instantiation of likely used services is possible.

2.6.2.2.5 Communication between services

We have identified three aspects of how services interact:

1. Who requests a connection from one service to another service?
2. Dataflow directions like forward, backwards or bidirectional. For example, a VoD client may demand a video data stream, and a video chat client may provide a video data stream.
3. Live-time management: who owns the service instance? Ownership is usually the same direction as the request direction, because whoever requested a service usually also owns the service instance[2]. However, there may be exceptions, for example for utility service components like video stream encoder and decoder.

[2] Reference may be useful, for example like in DCOM?

2.6.2.3 Execution Requirements

The execution requirements include both high-level attributes like availability, reliability, etc., as well as more low-level attributes like bandwidth, latency, throughput, jitter, etc.

2.6.2.4 Business Requirements

Services will typically also have a set of business requirements that constrain the deployment of that service in a FUSION domain. This includes the ability to define a number of policies concerning security, privacy, cost, etc.

In the today's fast changing market, not only end-users but also service and application providers prefer software platforms which make it easy to create new applications or provide new services. Ideally the workflow for creating a new service (e.g. for example a dashboard extension or an avatar-based community application) is simple:

1. You put together a new 3D application.
2. You press the "share" button.
3. You choose QoS, pricing and payment options.
4. You publish the URL (or send an access URL to others)
5. Any number of other persons can join.

Today application creators do not have the time to deal too much with technical aspects like programming, cloud services, distribution, load balancing, storage, configurations etc. This is the job of software platform providers[3]. Core principle in today's software industry are Rapid Application Development (RAD) and Agile Development, which are more and more a requirement for business survival and keeping a competitive edge.

[3] An analogy may be today's approach for creating applications like “google docs” or a shop: As application developer you don't worry about where the data is stored, how the data is synchronized, how scaling is implemented etc., but use software packages like GFS.

3. FUSION HIGH LEVEL ARCHITECTURE

3.1 Definitions

3.1.1 Service

A ***service*** is a set of functions provided by software or a system, usually accessible through an application.

3.1.2 Atomic service

An ***atomic service*** is a service that cannot be further decomposed, and does not contain other services. Base services are the finest level of granularity that is managed in FUSION [NGSO11].

3.1.3 Composite service

A ***composite service*** is composed of more than one service that can be a base or composite service. It contains an execution sequence of the composed services [NGSO11].

3.1.4 Service instance

A single instantiation of a service (atomic or composite) running in an execution zone and identified by a service identifier (serviceID). In FUSION there will usually be multiple instances of the same service running in the same execution zone and across many execution zones, all identified by the same serviceID.

3.1.5 Domain

A FUSION domain is a collection of execution zones and/or service routers that are managed by a single authority. Domains may peer with other domains.

3.1.6 Orchestration domain

An orchestration domain consists of a FUSION orchestration entity and one or more execution zones where the orchestrator may deploy and execute service instances forming either atomic or composite services. An orchestration domain may own the computing resources forming the execution zone or it may contract resources from a third party, e.g. a public cloud provider.

3.1.7 Service routing domain

A service routing domain consists of one or more service routers owned by a single administration. This could be an ISP, an orchestration domain or a third party. See section 6 on Business Models for a discussion of the various options. Service routing domains will usually peer with other service routing domains to provide a global service routing plane for all FUSION services. However in some business models a service routing domain may be associated with a single orchestrator which is also an ISP providing services to local ISP customers only. In this case peering between service routing domains may not be applicable.

3.1.8 Execution zone

A collection of computing resources at a physical location (e.g. data centre). The computing resources may be owned by third parties but are managed by the FUSION system – specifically by a FUSION orchestrator – and are available for deployment of FUSION service instances. The internal network of an execution zone is assumed to be highly connected and extremely reliable (low-latency, no packet loss). Execution zones are under the control of a single orchestration domain. A data centre may support multiple execution zones belonging to multiple orchestration domains.

3.1.9 Execution point

The specific physical or virtual environment in which a FUSION service instance is deployed within an execution zone.

3.1.10 Service router

The entity responsible for forwarding service requests. The requests will be forwarded to other routers until they reach the desired service instance in an execution zone.

3.1.11 Evaluator service

An evaluation service is a computational entity running in an execution zone that is able to score and rate the execution zone and the execution points within that execution zone on various aspects of the service manifest – for example availability of specialised hardware, network and computational metrics for QoS estimation, the performance measured to nearby service instances. See Deliverable D3.1 [D3.1] for more details.

3.2 Overview of the FUSION system

The FUSION framework can be seen in Figure 7. Functionality is divided into 3 layers. At the lower level IP routing forwards packets using traditional end-to-end protocols. At the top layer the execution plane consists of all the execution zones where the services' instances will run. In the middle the service router layer will forward request 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 7. Service routing domains, consisting of one 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 7. Service routing is anycast in nature – the user simply requests a service and it is the 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 7. Note that the physical data centres are depicted in the lower IP routing layer as the data-plane communications will be directly between users and service instances running in physical data centres, while the abstract representation of execution zones – a logical partition of a data centre – are shown in the upper execution plane.

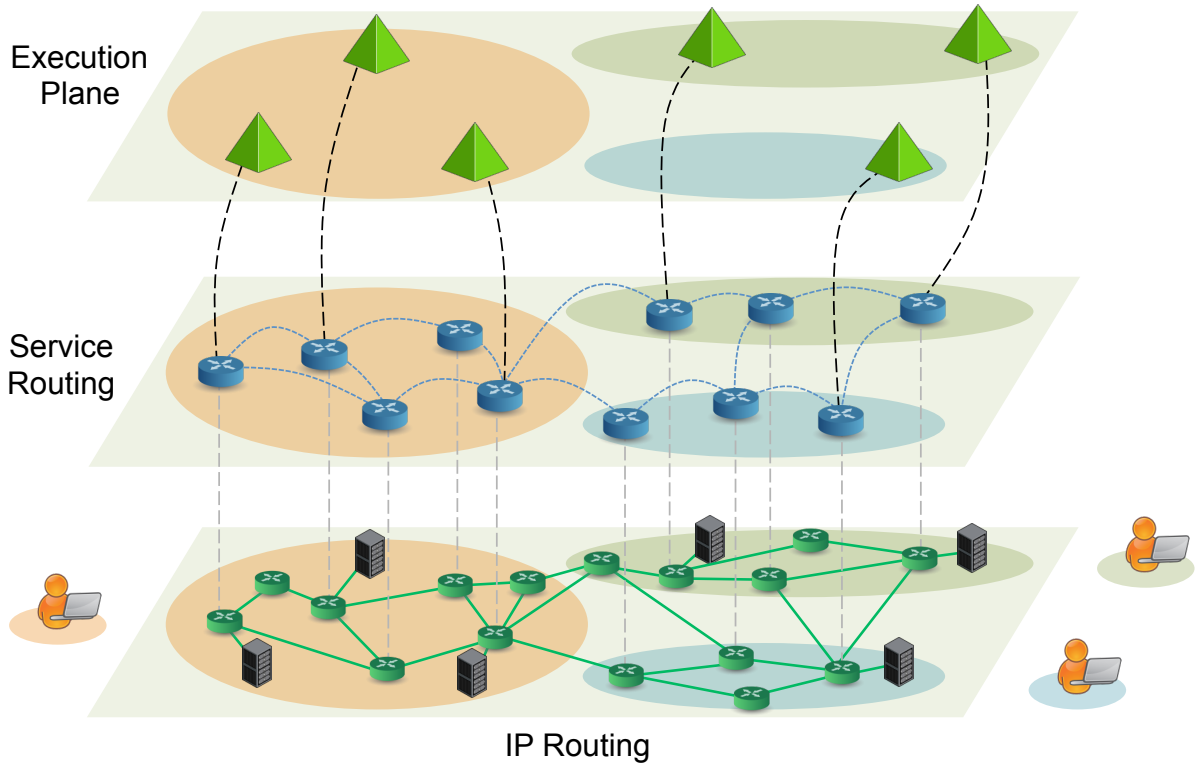


Figure 7 - FUSION Framework

In the following section the functional entities required to implement the above system are defined at a high level – the details are developed further in deliverables D3.1 and D4.1 [D3.1, D4.1].

A description of the FUSION service layer, including lifecycle and state management and user interaction is described further in section 3.5. An overview of the execution plane consisting of execution zones managed by a zone manager is described in section 3.6. The orchestration and

execution functionality including service placement algorithms is introduced in section 3.7. All three aspects are dealt with in more detail in deliverable D3.1 [D3.1].

The service networking architecture, covering the role of service routers, is discussed in section 3.8. A range of options for service oriented networking are covered, ranging from DNS-like name resolution approaches to clean-slate deployment of native service-oriented networking. Deliverable D4.1 [D4.1] analyses the service networking functions in more detail drawing conclusions on specific routing and forwarding protocol design options which will be developed in detail in the second year of the project.

The interaction between the service execution and orchestration layer and the service networking layer is a key aspects of the FUSION approach to service-oriented networking. Section 4 investigates and compares two possible modes of operation: integrated vs disjoint orchestration and routing. These options and further models for lightweight interaction between service and network layers will be analysed and evaluated during the course of the project and the results will be presented in the final deliverable of this workpackage, D2.3.

Figure 7 models the layers of the FUSION system but so far the mapping of roles to business entities has not been discussed. These aspects are discussed in section 6.

3.3 FUSION architecture overview

The main functional entities in the FUSION architecture are depicted in Figure 8. The three main entities 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 role and functionality of the orchestrator is elaborated in section 3.7.

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. The execution layer functionality is elaborated in section 3.6.

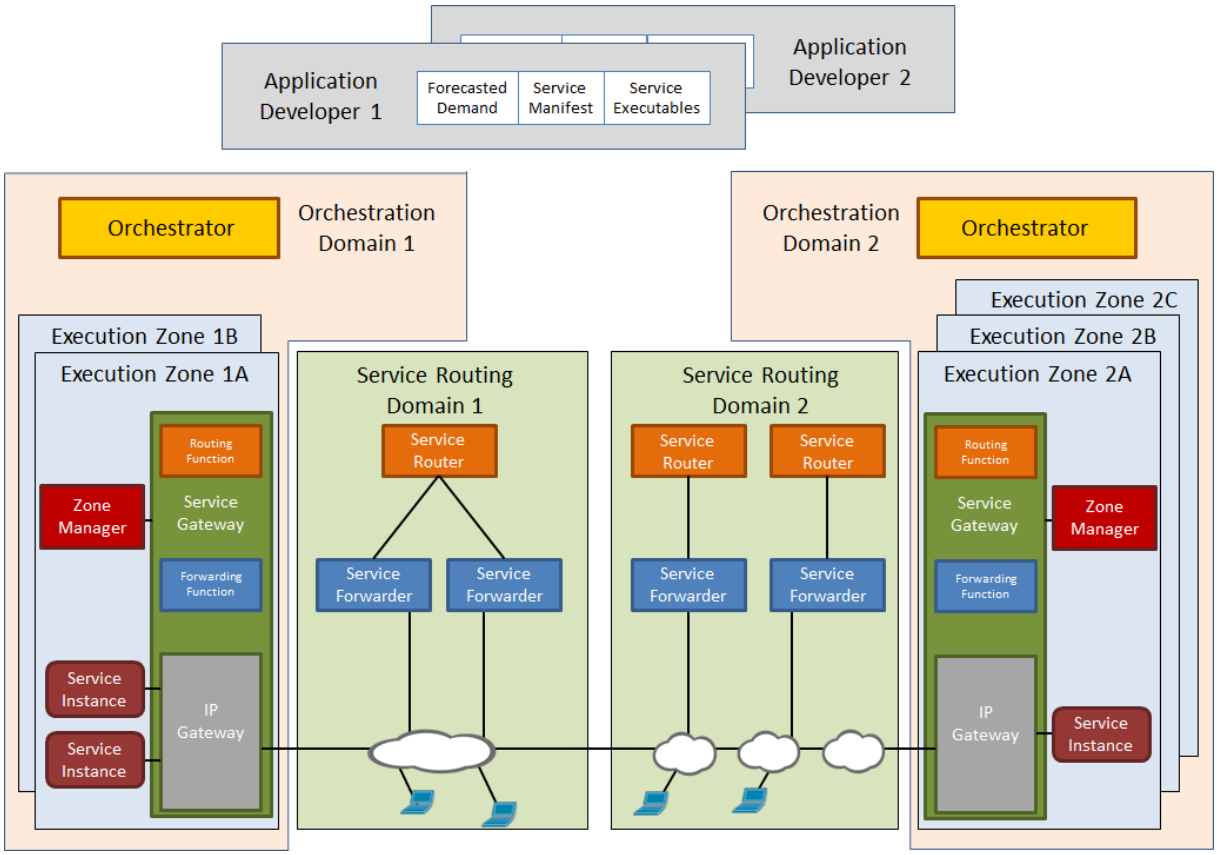


Figure 8: FUSION High Level Functional Architecture

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[4] in Figure 8. The service router 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 received 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 – see Figure 10 and Figure 12 – 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 8 – in this architectural option 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 (see also section 4.1 on integrated orchestration and routing). 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 8 (see also section 4.2 on disjoint orchestration and routing). The service networking architecture is elaborated further in section 3.8).

[4] 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 document.

3.4 FUSION interface specifications

The FUSION interface will be fully specified in the Deliverable 2.2. Here we briefly outline what the interfaces will achieve.

3.4.1 Service Publishing and Deployment

The first set of interfaces (included in Figure 9) deal with publishing and service deployment. The publishing interface is used between the application developer and the orchestrator in order to initialise a new service and grow/shrink it. The orchestrator then uses the deployment interface to install/remove service instances.

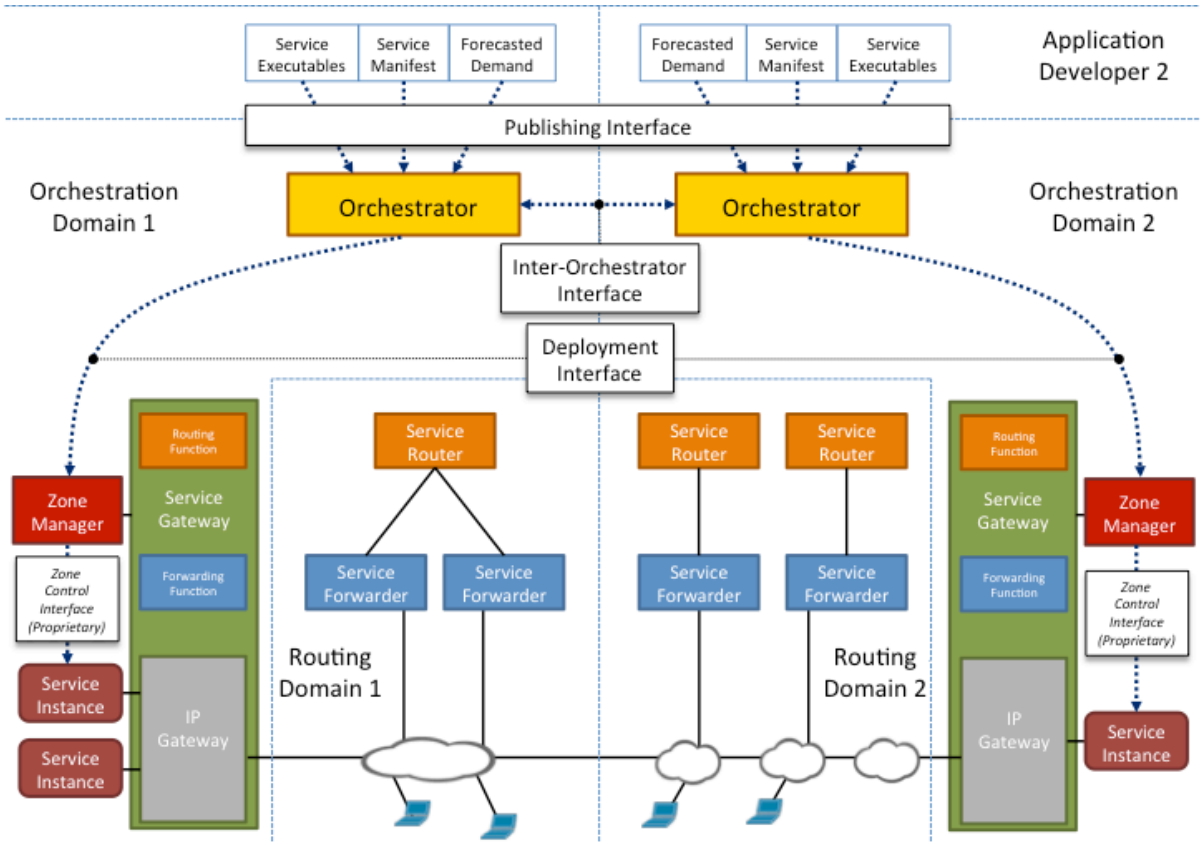


Figure 9: Service Deployment and Publishing interfaces

Deliverable D3.1 contains detailed descriptions for these interfaces [D3.1].

3.4.2 Service Routing Plane

Service routing needs the definition of three interfaces. The execution zone reporting interface communicates with the service gateway and quantifies the availability of a service instance. This gets then exposed through the execution announcement interface to the service routing plane. Finally the inter-domain announcement interface allows this information to get propagated to all the service routers.

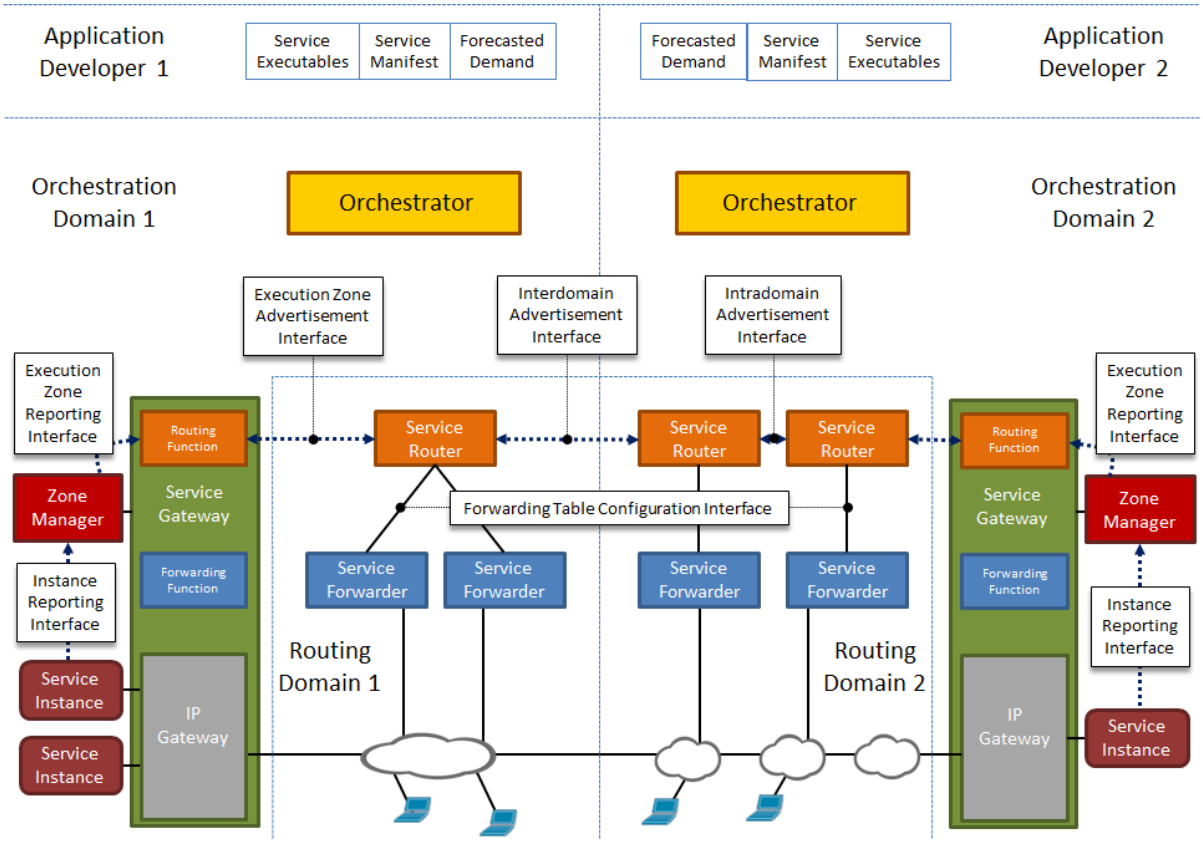


Figure 10: Service Routing interfaces

These interfaces are detailed in Deliverable D4.1 [D4.1].

3.4.3 Monitoring

Monitoring, as illustrated in Figure 11, plays an important role in Service oriented networking since routing decisions need data to inform their decisions. The service monitoring interface allows for clients and servers to report end-to-end quality of experience. Network monitoring allows network providers to use network information to inform the service routing. Finally the execution zone monitoring interface allows the status reporting to the orchestrator in order to inform grow/shrink decisions for a given service.

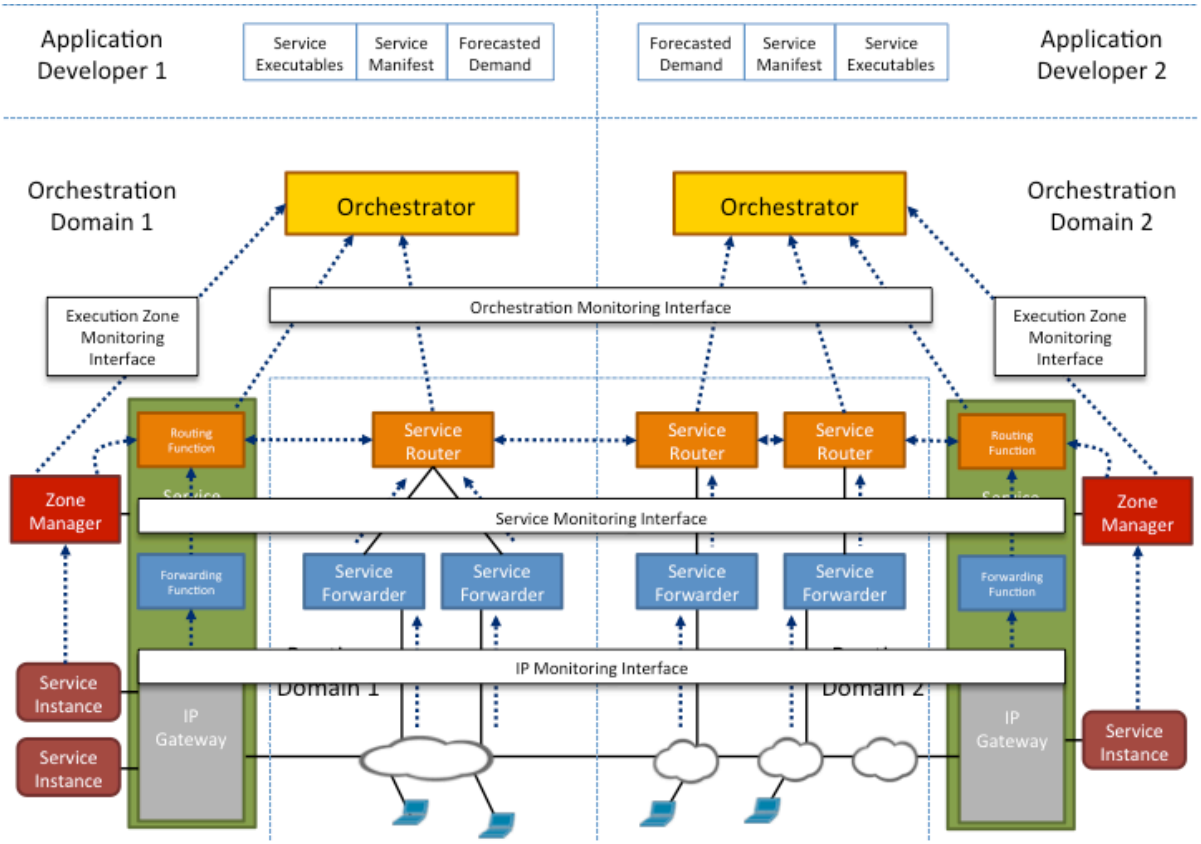


Figure 11: Monitoring interface

3.4.4 Service Querying/Invocation and Data Forwarding

After routing information has been propagated, service requests need to be routed to the right service instances. This is illustrated in Figure 12 and consists of two parts: the execution zone query interface and the inter-domain query interface.

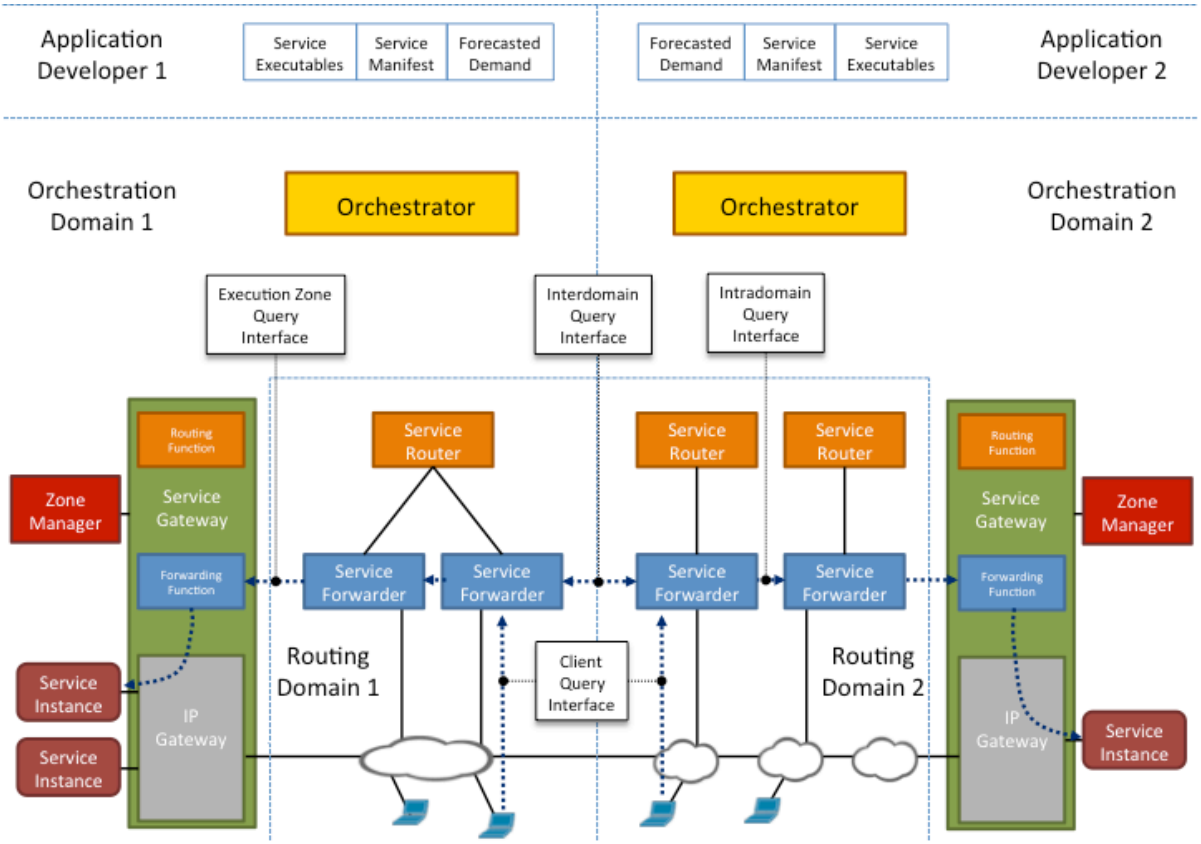


Figure 12: Query/invocation interface

The final stage of the FUSION workflow is the data exchange which takes the unchanged Internet and can be seen in Figure 13.

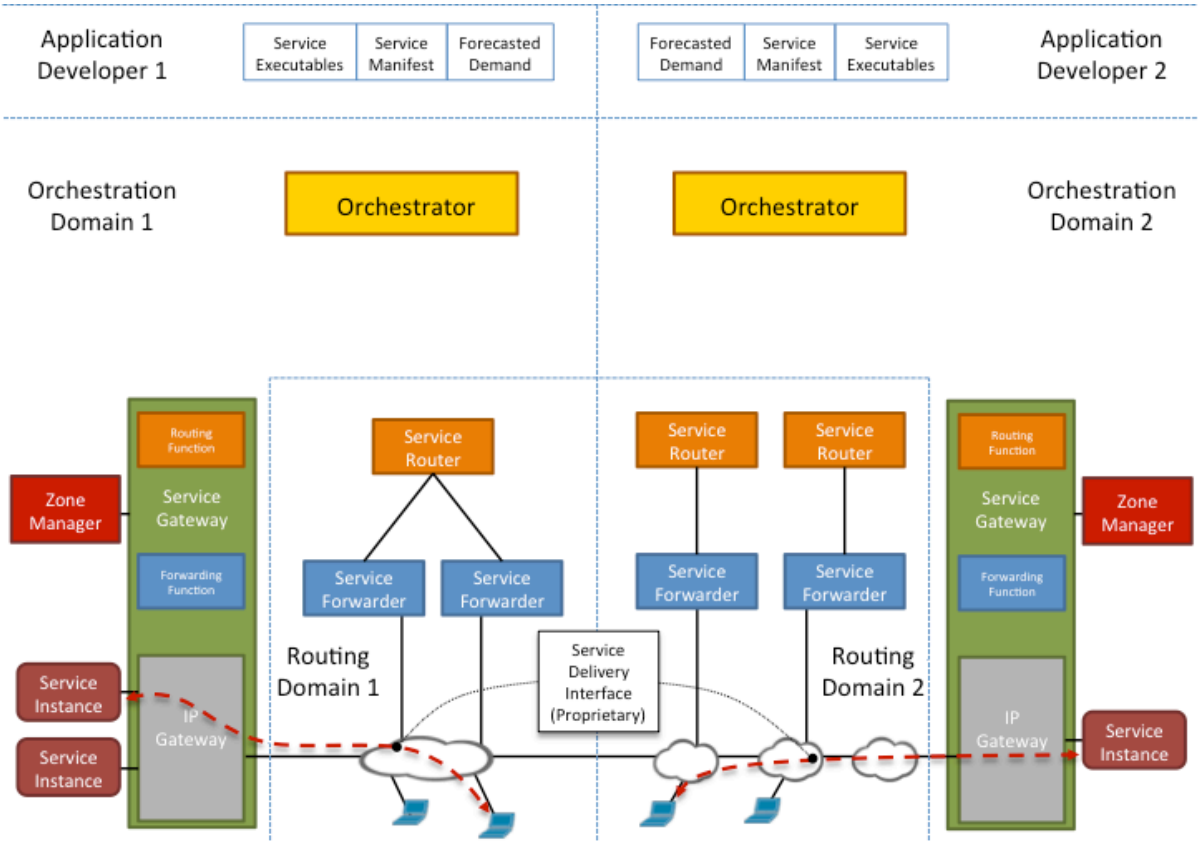


Figure 13: Data Forwarding

Deliverable D4.1 describes in more detail these interfaces [D4.1].

3.5 Service Layer

In this section, we will describe a number of key elements and functions of the service layer of the FUSION architecture.

3.5.1 Basic service management functions

All FUSION services will have to provide and implement a number of basic functions to be able to run on top of the FUSION infrastructure. This includes the following functions:

- Lifecycle management

  Each service should provide the means for FUSION to properly start and stop a particular service instance. This could be implemented for example by providing proper scripts or by providing proper call-back functions.

- State management

  When migrating services, the service could assist FUSION by implementing functions that capture the state of an instance, or that can resume execution based on a particular state. However, it is assumed that researching mechanisms for storing or restoring the state of a service instance are outside the scope of FUSION.

- Monitoring

  A service can implement particular monitoring functions that allow the FUSION execution layer to monitor a particular instance, provide a service with particular monitoring information or ask the service what action to take when things go wrong.

Services can choose to implement one or more of these basic functions. Execution zones may only accept services that implement at least a particular subset of functions to ensure proper execution inside that zone. More details on the interfaces will be provided when discussing the core FUSION service APIs, see section 5.2.4.

3.5.2 Specific FUSION service functions

Some services will implement specific functionality that is important for some aspect of FUSION. Each of these services will have to implement a particular API or implement particular communication protocols to provide that functionality. Examples include the orchestration services, monitoring services, service routing services, the evaluator services or probes that help service mapping, etc. More details of these APIs and communication protocols can be found in section 5.2.

3.5.3 Service communication functions

Services will typically provide one or more communication channels, implementing particular protocols.

3.5.4 Service Manifest

A service manifest is a static description of a service, including all its dependencies, requirements, constraints, business aspects (cost, etc.), parameters, etc. It contains all functional and non-functional requirements to be able to automatically and optimally deploy and create instances of this service in execution zones in FUSION domains. The service manifest should be described in a format that is understandable by FUSION to enable fully automatic deployment and instantiation.

The manifest likely consists of several parts, each of which could be provided in separate documents, and each of which is only relevant (or perhaps even visible) to the relevant parts of FUSION or its stakeholders. For example, there will be sections about deployment, how many resources to allocate, how to monitor, etc. Other parts of the manifest should also be made public so that other

services, potentially from other service providers, can communicate with that service or even integrate that service into a more complex composite service. For example, sections that describe the public API of that service, including the communication API, the service functionality, etc. should be visible to all other services or providers who are authorised to interact with this service type.

The service manifest can only contain the static aspects that are known in advance, and that can easily be described in a manifest file. In many cases however, this is insufficient or too complex. As a result, we also rely on the concept of evaluator services, which is a dynamic, executable version of a service manifest and which can complement (or help interpreting) the static service manifest file. Deliverable D3.1 [D3.1] contains more details on these evaluator services.

3.5.5 Service Request

When a client (or another service instance) makes a service request, FUSION has to decide to what service instance it will forward the request. We will first shortly discuss the potential structure of a service request, followed by two key scenarios for FUSION when routing and handling the request.

3.5.5.1 Structure

A service request will typically consist of a service name and a number of service parameters, some of which can have an impact on the routing towards an appropriate instance. For example, different instances of a video transcoding service could be running in the network. The request could specify the desired resolution, which might impact the routing decision, because higher resolution output streams will stress the network more. As part of the request, the client may also optionally include some payload. An example of this payload is an image that is passed to the service instance on which that instance will for example perform face recognition. When the service instance returns a response, this response could be a return code in combination with an optional payload or even a stream of data, as is the case with HTTP requests.

3.5.5.2 Session Slots

We introduce the notion of FUSION session slots as a service independent way of advertising service instance availability to the service routing plane and to orchestration. Each service instance can typically handle a particular number of service requests in parallel. When a new service request is issued by a client (or another service), the service routers need to decide to what service instance to route that particular request. For this, it needs to take into account many factors, including the overall load of a particular service instance. For many reasons, including scalability and privacy, it may not be practical or feasible to expose high-level or low-level resource utilisation data. We therefore introduce the concept of FUSION session slots.

During service instantiation, a service instance may allocate sufficient resources for handling N FUSION sessions in parallel. We thus say that the service instance has N available session slots. At execution time a service instance will identify how many session slots are available for new sessions depending on the current occupancy of the instance. This information will be collected by the zone service gateway and announced to the service routers so that future queries/invocation requests from clients can be routed to execution zones with available resources. The zone service gateway will aggregate the number of available session slots across all instances of a service running in an execution zone to be announced to the service routing plane. This can be averaged over time to avoid instabilities in routing due to highly dynamic changes in session slot announcements.

It should be noted that the service gateway is under the control of the local zone manager and policies specifying the frequency of route update announcements or how to aggregate session slots across instances may be specified and managed by the zone manager. As an example of such a policy: an execution zone may implement automatic service scaling described in section 5.4.3 of deliverable D3.1 [D3.1]. The orchestrator will have defined a maximum number of service instances or session slots that an execution zone should provide and it is the zone manager who manages how

many instances are running at any one time according to observed demand. Provided that the execution zone can instantiate additional instances quickly enough (as specified in the service parameters defined by the orchestrator) it may choose to announce more session slots than are currently available by running service instances. For example an orchestrator may request that an execution zone provides up to 3 instances, each of which can support N session slots. Even though only one instance is running the zone manager may set a policy of announcing 3N-(number of occupied slots) session slots to the service routing plane. In this way the execution zone can make itself available for serving up its maximum quantity of session slots without predeploying idle service instances that consume computational resources.

The session slot concept is elaborated in more detail in deliverable D3.1 [D3.1].

3.5.5.3 The Available-Instance Scenario

In this scenario, the FUSION service routers discover at least one available instance that matches the service and request parameters and that still has available session slots. In that case, the service routers can very easily forward and route the service request to the corresponding instance in its corresponding execution zone, which will handle the request and typically send back a response over the appropriate reply channel. Note that to be able to service the request, the selected instance in its turn may also perform one or more service requests to other services.

3.5.5.4 The No-Available-Instance (On-Demand) Scenario

In this scenario, the FUSION service routers detect that there is no instance available that matches the service and request parameters or that still has available session slots. It may also be the case that there is simply no instance running at all yet. There may also be a grey zone in which there is still an instance available, but it is not an optimal instance, for example because it is located too far from the client to deliver the best service. On the other hand, starting a new instance on-demand will definitely take some time. The question then raises what option to choose.

There are a number of reasons, which in combination of each other make the on-demand scenario equally important. First of all, a cloud does not come for free. As you typically pay per use, running an instance all the time will consume a particular number of resources and thus has a price. Second, the types of services that we want to support are typically rather demanding in terms of compute and network resources. This means that the amount of resources that has to be reserved per instance will typically be much higher than a classic web service. Third, in FUSION, we target a distributed execution model, where there can be many execution zones spread across many FUSION domains to be able to deliver an excellent QoE towards the clients. This however means that, in order to avoid the on-demand scenario, each service type would have to be predeployed on all relevant execution zones. Accurately predicting which execution zones will be relevant at each point in time for all services is nontrivial, and flooding all execution zones with instances is simply too expensive if the actual resource utilisation turns out to be much lower. Fourth, it is likely that FUSION could eventually host a huge amount of smaller and sparsely used interactive applications (cf., the long-tail), for which predeployment at a specific execution point does not make much sense. An example can be a 3D game renderer client for a specific game, which needs to be close to the end user for optimal interactivity, but which is almost impossible predeploy successfully in a cost-effective manner, unless when it is a very popular game for which statistic multiplexing applies and load prediction is effective.

The on-demand scenario can be handled in two ways, either in-band or out-of-band. In the in-band approach, the router forwards the request to the orchestration while requesting to create a new instance, all in one step. In the out-of-band approach, an appropriate error code is returned to the client and the client needs to explicitly contact the orchestration layer and forward the request there, implicitly or explicitly asking to create a new instance along the way.

There are a number of complications that come along with the on-demand scenario. First, the entire chain needs to be made aware that there might be cases where there is no instance available, and implement ways to deal with this case. Second, the on-demand scenario inherently results in an increased start-up latency when making the request, as FUSION may have to download, install, configure and start a new service type, which can take a while for non-trivial services. Moreover, because this process should be fast, this will also impact the overall network and execution points as the packages will likely be transferred as soon as possible, and the execution point needs to be installed and prepared as quickly as possible as well, temporarily creating spikes in bandwidth, compute and I/O resources.

3.5.6 Late Binding

Depending on the service placement, two communicating services may have been deployed on the same (physical or virtual) machine or another, either inside the same execution zone or across multiple execution zones. Depending on what option was selected (for whatever reason), the optimal communication protocol and mechanism may vary. For example, if two communicating services are co-located on the same machine, they may pass raw data directly in shared memory. On the other hand, if the same two services are located on different machines, potentially on different execution zones, then it may be necessary to add one or more transformation functions, for example to save bandwidth (H.264 encoding) or to increase overall security (SSL encryption) at the expense of increased compute resources (and additional end-to-end latency).

As a result, FUSION should support a late binding mechanism for services, to enable to use the optimal communication channel depending on the chosen service placement distribution and physical mapping. Next to this, FUSION should also allow services to implement their own communication channels, to be able to support custom and dedicated communication mechanisms. One example is the efficient sharing of GPU buffer pointers across multiple service instances running on the same machine to avoid expensive copying and wasting huge amounts of bandwidth. As this is a very dedicated protocol, it is impossible for FUSION to support this out-of-the-box. However, FUSION should make it easy for services to detect these scenarios and deploy their own communication mechanisms via custom libraries in a reliable and secure way.

3.6 Execution Layer

In a FUSION domain, the FUSION execution layer is responsible for executing and managing the execution of all service instances that are active in a domain. The FUSION execution layer is managed by the execution zones, which are completely responsible for the runtime management of all its service instances.

From a FUSION domain-level, each execution zone is considered to be a black box, allowing for each execution zone provide its own implementation of the execution layer, optimised for the set of resources it contains and the set of services it wants to support. Each execution zone only needs to implement the zone API so that the orchestration layer at the FUSION domain level can interact with the execution zone.

Concretely, the execution layer consists of a number of execution zones, each of which contain a zone manager that interfaces with the orchestration layer, the other execution zones as well as the (domain-level and zone-level) networking layer.

3.6.1 Zone Manager

The zone manager is responsible for interfacing with the domain-level orchestration layer, the network layer, managing its internal resources (compute, network, storage, etc.) as well as managing the services that are deployed inside its execution zone. It is also responsible for providing an execution and management platform for executing and managing FUSION services on these resources. This includes managing the underlying heterogeneous hardware resources, managing VMs, containers, etc.

To provide this low-level functionality, a zone manager will typically interface with existing application and infrastructure management platforms (e.g., OpenStack, CloudStack, PlanetLab, Globus, etc.). Consequently, the zone manager will typically act as the glue between the FUSION infrastructure and existing cloud infrastructures on which FUSION services are eventually deployed.

3.6.1.1 Service management

The zone manager is responsible for managing all FUSION services and instances that are deployed and running inside its execution zone. It will handle requests coming from the orchestration layer and translate them onto the existing cloud infrastructures, taking into account the service requirements. It will also provide the appropriate monitoring and management services to ensure that the instances are running properly.

3.6.1.2 Resource management

We envision that a FUSION execution zone will consist of a heterogeneous set of resources. This includes different types of compute, network and storage resources as well as dedicated or specialised accelerators. The zone manager will be responsible for managing and monitoring these resources, ensuring that FUSION services use them appropriately.

3.6.1.3 Load balancing

Because of the demanding and sensitive nature of many of the FUSION services, properly balancing these services across the available (network, compute, accelerator and storage) resources will be extremely important. As a result, a key function of the zone manager will be to dispatch and balance all its instances appropriately across its available resources. In case the zone manager builds upon an existing cloud infrastructure platform, a key task of the zone manager will be to make sure that the cloud infrastructure is configured in such a way that the appropriate QoS can be guaranteed to the FUSION service instances.

3.7 Orchestration Architecture

In a FUSION domain, the orchestration layer is responsible for managing the domain-level resources as well as all services that are registered within the domain. Resource management within a domain includes key functions as resource registration and resource monitoring; service management includes key functions as service registration, placement, instantiation, and monitoring.

3.7.1 Service Placement

Service placement or service mapping is the act of finding an optimal location for instantiating a new instance of a particular service type. Service placement is closely related with resource monitoring and service deployment, as services should only be mapped onto execution zones or hosts with enough available and appropriate resources, and is triggered by a service instantiation request.

The input for the service placement operation consists of a number of constraints coming from multiple locations, including the service manifest, the service request parameters, the resource availability, cost constraints, time budget for doing the placement, etc. The output of the operation is the optimal execution zone for creating a new service instance with the appropriate constraints.

3.7.1.1 Service Placement Heuristics

The communication channel between service instances is a good sample for discussing the two situations of a standard implementation provided by FUSION versus a custom implementation by FUSION users. Standard Media Data flow may include for example:

- FUSION network data channels using FUSION routing (and possibly optimised internally to use shared memory when possible).
- Standard shared memory API

Custom Media Data flows are for example:

- Communication via local files.
- GPU object handles
- Cooperating service instances share a single address space for communication and access shared C++ objects via pointers.
- Services run fundamentally differently (e.g. game server and game clients use a shared scene graph or physics state)

3.7.2 Static Orchestration

We now describe the steps in which we envisage service orchestration taking place for the initial static case. The text will be written as if it pertained to a centralised orchestrator intra-domain case; decentralisation of orchestration functions and inter-domain generalisations will be addressed later.

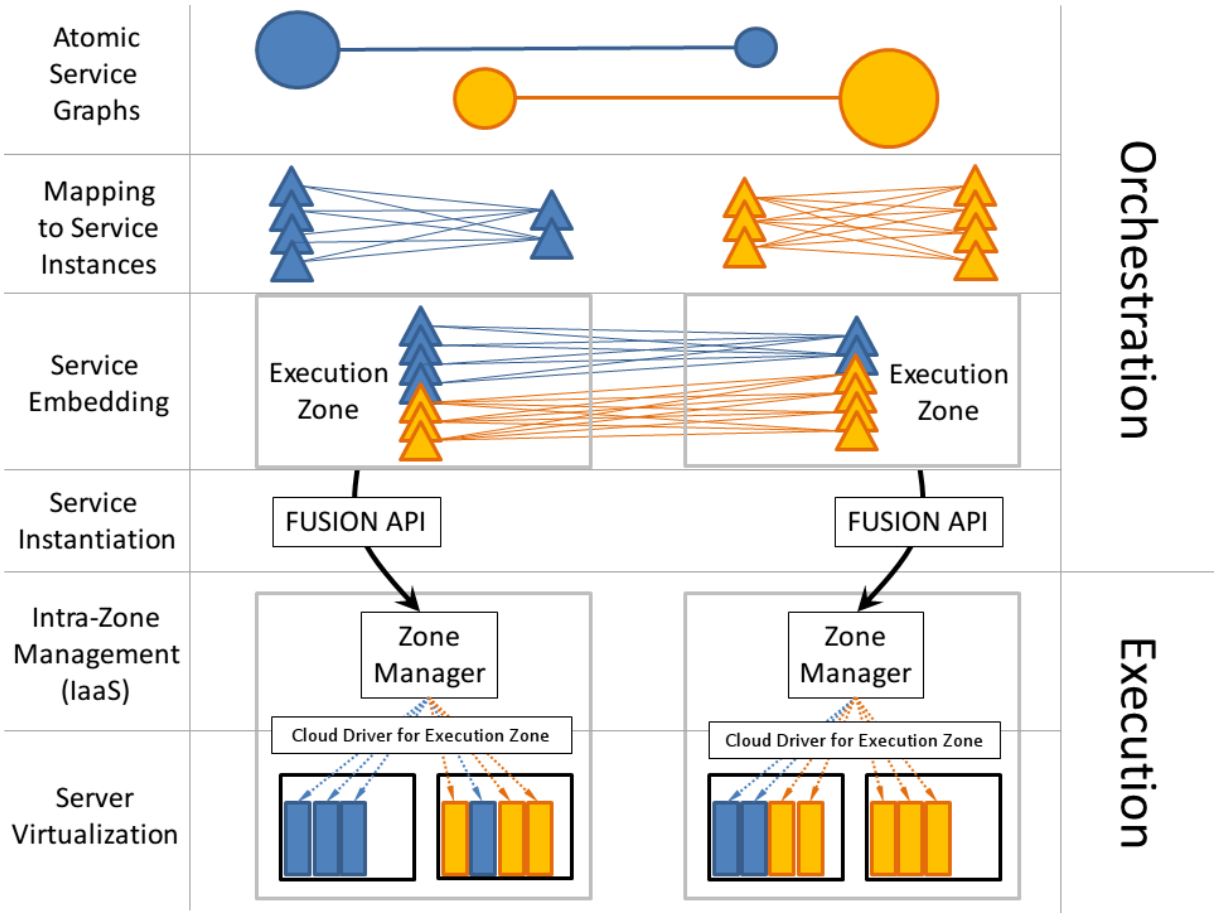


Figure 14: Static Service Orchestration

Phase	Actions
Service Graph Request	The input to the optimiser is a set of atomic service graphs; that is, a graph where nodes are atomic services and the links represent data flow between them. Both nodes and links are labelled with minimum/expected/maximum demand information and performance constraints. If non-atomic services are required, these need to be converted into atomic services first.
Mapping to Service Instances	Using the graph labelling information provided as part of the service graph request, the orchestrator determines the number of atomic service instances needed to satisfy a given level of service, as well as the network capacity and QoS requirements between them. We call this the instance graph request.
Service Embedding	The orchestrator takes the many instance graph requests that it has received from all clients, and it maps them to the available computation and network resources. This yields an service instantiation plan for all service graph requests. Hence, service embedding in this first approximation statically maps service requests to the available execution zones taking into account the limitations imposed both by the execution environments and the network between them. Of course, if there are not enough resources to fully embed all instance graph requests, some may be rejected or downgraded in QoS. This additional complication needs to be better explored.
Service Instantiation	The orchestrator then takes the service instantiation plan and implements it on all execution zones. Execution zone functionalities are invoked using the

	specific interfaces of their corresponding providers.
Service Execution	Each zone manager starts as many instances as required by the orchestrator, configuring also any network-related options between them.

3.7.3 Multi-Domain Static Orchestration

There are two ways in which multi-domain orchestration can progress from here.

3.7.3.1 Disjoint Orchestration

In a very simple case, an orchestrator domain B simply grants permission to another domain A to issue commands to its cloud drivers, hence allowing A to instantiate services in Execution Zones with whom it has no direct contact. This implies that A performs its service embedding on its own, simply labelling some of its instances as external, and invoking the corresponding drivers on the other domain when implementing its service instantiation plan.

Unfortunately, this simple architecture implies that B has no control over the capacity that A can consume, which could hurt its own customers. Unless A has some sort of hard reservation on resources, chances are service requests from A will at some point experience some kind of admission control/QoS control by B. Since this will be invisible to the orchestrator at B because A is talking directly with the driver, it will be the responsibility of the driver to give priority to requests from B. Hence, A will have to have mechanisms to cope with the fact that some of its service embeddings may fail at the instantiation level, rather than at the embedding level.

3.7.3.2 Joint Orchestration

In this slightly more complex case, an orchestrator domain B simply grants permission to another domain A to issue partial instance graph requests that connect service instances in B with services instances in A (or other domains, for that matter). Since these requests from A are then visible to the orchestrator at B, B can implement any policies needed regarding priority (the cloud drivers are then not required to implement this functionality).

3.8 Service Networking Architecture

The FUSION network layer is responsible for interconnecting service component instances according to well-defined metrics. The network stack provides both control and data plane functionalities. The functional requirements for the network stack are as follows:

- Naming: the applied scheme must allow to identify services. This name will be used by application developers, similar to the URLs used in web applications.
- Addressing: the addressing scheme allows to deliver data through the network to the correct service instance component.
- Routing: determining the path to be followed by data, according to specific policy configured by the network operator.
- Forwarding: transferring incoming packets from one interface to one or multiple outgoing interfaces, according to the routing table policies.

3.8.1 Naming and Addressing

Naming schemes are required for specifying the entity. Naming and identification are mostly used as synonyms. Naming does not necessary mean that the identification should be human-readable. Several entities may need their own identification scheme:

- Services: service identifiers refer to the service as the abstract entity. Today's URL scheme is an example of such a service identifier: connecting to "www.fusion-project.eu".
- Service instances: multiple instances of the same service might be deployed in the network. For stateful services, it is important to forward subsequent requests to the same instance and a service instance identifier is required. Today, no explicit naming scheme is used for instance identification. Service instances are identified by the tuple (serverIP, serverPort).
- Sessions: service sessions (defined in detail in [D3.1]) are the period of time in which a (set of) communication channel(s) exists between a client and the service instance that handles the service request. Session identifiers may be used in service routers if flow-based forwarding mechanisms are used (e.g. OpenFlow), or at the endpoints for demultiplexing to the correct service instance. Today, sessions are only identified at the endpoints and transparent to the network (apart from middleboxes like NAT). If multiple threads are listening on the same port, demultiplexing is performed on the tuple (clientIP, clientPort, serverIP, serverPort).

The addressing scheme is used to route traffic to the physical location of the entity specified by the entity. Currently, IPv6 is the predominant addressing scheme that is used. Note that the IP addresses are overloaded: they have both the meaning of an identifier (possible in conjunction with the port) as of physical, routable address.

Since the address is tightly coupled to the physical location, a moving entity will need to be reassigned a new address reflecting the new location. For example, if a service instance (inside a VM) moves to another subnet, or if a client attaches to another subnet, the IP address will change. This change of IP addresses might interrupt running sessions. Network-layer mobility solutions adopt two basic approaches: routing-based and mapping-based [Chan]. Under the routing-based approach, a mobile node keeps its IP address unchanged regardless of its location changes. Thus, the IP address is used to both identify the mobile and to deliver packets to it. The routing system keeps track of the most up-to-date location of the addressed entity and updates the routing tables to deliver packets with the unchanged IP address to the new location. Under the mapping-based approach, the IP address of a mobile dynamically changes to reflect the current location of the mobile. An explicit mapping function in the system is needed to map the stable identifier of the moving entity to its changing IP address for delivery. Typically this is done by an intermediate proxy (mobility anchor) to which all traffic is directed. The anchor performs the mapping function to the current IP address of the mobile node. This approach is used in Mobile IP and Proxy Mobile IP.

In practice, most of the deployed architectures today have a small number of centralised anchors managing the traffic of millions of nodes. Currently, there is a trend from the centralised mobile management deployments to more distributed mobility management, closer to the user at the network edge [ietf-dmm]. The motivations behind this shift are relevant for FUSION:

- A shift in user traffic behaviour: there are many direct communications among peers in the same geographical area.
- Mobile nodes remain attached to the same point of attachment for considerable periods of time. Mobility management support is not required for applications that launch and complete their sessions while the mobile node is connected to the same point of attachment. As IP mobility support is designed for always-on operation, it maintains all context parameters for a mobile subscriber for as long as they are connected to the network.

When designing identifier-to-address mapping functionality in FUSION, the most important question is whether we want to support session continuity during migrations of service components (client and/or service instance in the network). This involves the following questions:

- What is a typical duration for users to stay connected to the same subnet? A more complex question is whether we support multi-interface solutions.
- Is it realistic to assume that service instances will migrate when they have sessions active? Apart from failure recovery, what would be motivations for a service instance migration? If a service instance gets overloaded, it is the session that should migrate to a (possibly only very recently instantiated) instance.
- Many application-layer techniques already exist, that support temporary disconnectivity.

When defining naming/identification schemes, the following dimensions must be taken into account:

- Globally unique: the allocation of names to services and service instances must be globally unique, or at least the probability of allocating the same name to different entities must be extremely low.
- Name integrity: avoid routing to spoofed services. Two options are proposed in literature: self-certifying (embedding the hash of the content in the name), and indirect binding: embed the public key of the publisher in the name, and sign the hash of the content with the corresponding secret key (requires 3rd party PKI)
- Location-independent: service instances (including the client component) might be migrated. After migration, their locator will have changed, but not their identifier.
- Granularity: do we only name services (service components), or also service instances?

Structure: The naming scheme may be flat or hierarchical. Flat names may be independent of organisational changes (e.g. change of ownership) and self-certifying (cryptographic hash). Flat names are used in NetInf, DONA, PSIRP. Hierarchic naming schemes are used in CCN, where the names are rooted in a prefix unique to each publisher. Even more advanced naming schemes can be considered, such as the directed acyclic graph (DAG) in the eXpressive Internet Architecture [XIA].

3.8.2 Resolution and routing

If a client (or service component) requests another service component, FUSION will serve this request by routing to the appropriate instance. This process of resolution can be done in different ways:

- Out-of-band: similar to DNS, where a name resolution service is queried by the client component before the connection is initiated. The DNS returns the locator of the selected instance. In this

case, routing in the network is done based on the locators. A middle ground between this approach and the in-band routing is where the first service router performs the name resolution.

- In-band: name-based routing. The client is unaware of the locator of the remote instance. The network routes to the most appropriate instance, based on the name.

3.8.3 Service Routing Deployment Options

Below is the list of possible scenarios for deployment of FUSION service forwarding and routing. Since we assume deployability in the current Internet, we assume that an IP routing is available and that service routing will be implemented as an application layer overlay.

Note on notation: in the figure, triangles are execution zones, squares are service routers and circles are IP routers.

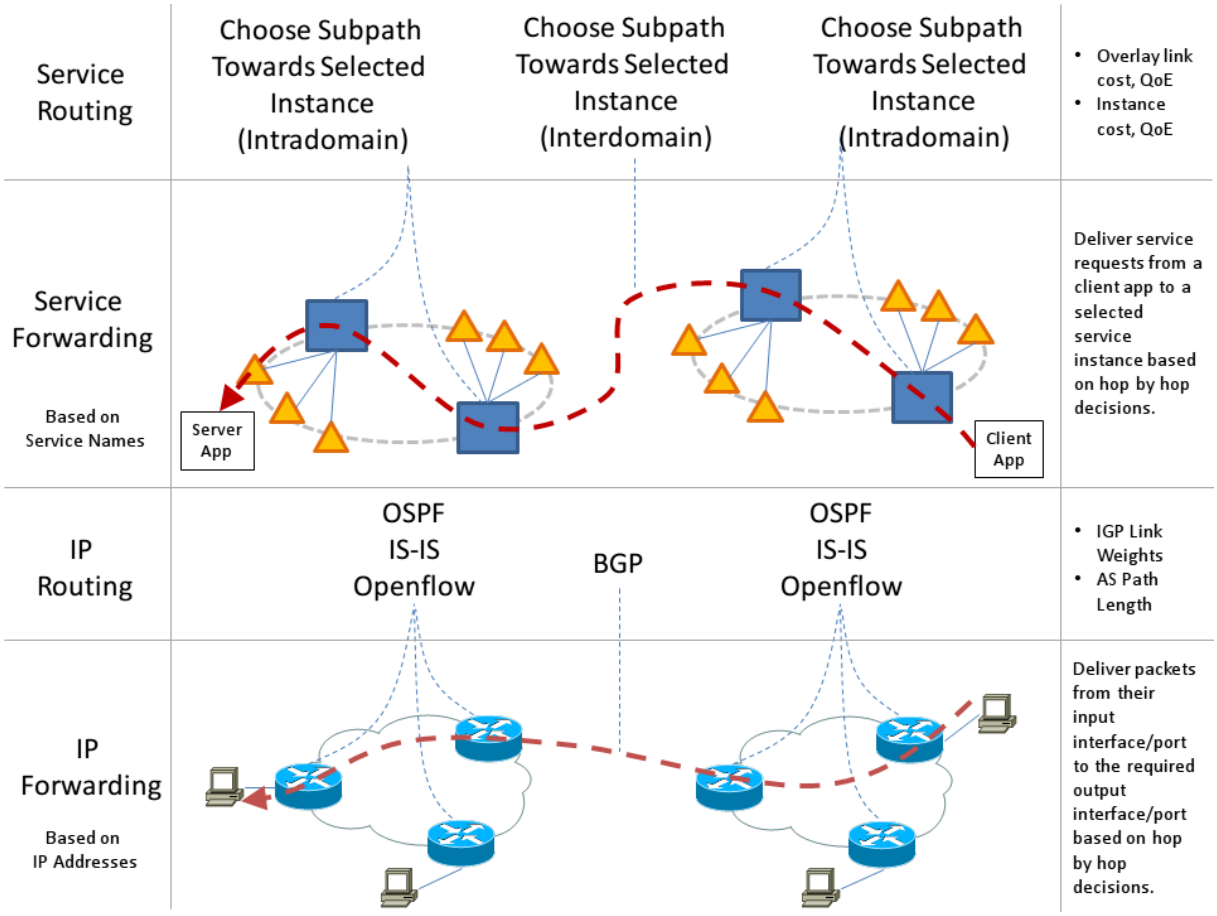


Figure 15: Service Forwarding and Routing

The following deployment scenarios are categorised in three groups. Firstly, options where service routing and forwarding is largely implemented by the IP layer – section 3.8.3.1. Secondly, scenarios where service routing and forwarding is achieved by a combination of IP layer and service routing functionality – section 3.8.3.2. Finally, in section 3.8.3.3, two scenarios where service routing and forwarding protocols are decoupled from the IP infrastructure. The benefits and challenges of each scenario are introduced and a table is introduced to compare the scenarios with regards to their approach to Service Naming, Name Resolution, Service Forwarding, Routing, Signalling, Monitoring and Service Delivery from both intra- and inter-domain perspectives.

3.8.3.1 Higher Responsibility placed on IP Routing/Forwarding

Advantages/benefits:

- Based on current Internet protocols:
  - clear evolution path
  - zero stretch
- Reduces service-routing problem to a simplified service-name resolution
- Off-line lookups (DNS, or DNS-like) with minimal changes to client applications/API/OS protocol stack.

Disadvantages/challenges:

- Increased latency - RTT for lookup/resolution per service request [but this happens only at the start of the session, so only an issue for short-lived sessions].
- Less consistent with vision of native name-based routing in the Internet.
- To get benefits of late-binding (including support for mobility/migration) needs frequent updates and look-ups or a pub/sub approach.

3.8.3.1.1 Using DNS

This scenario uses the DNS without any changes to the client-server DNS protocol. Normal DNS queries return the IP unicast address of a host running the service. This scenario implies an incremental change to the existing Internet architecture based on hierarchical naming and we consider two alternatives: reuse of existing host/domain names or the adoption of new flat or hierarchical naming schemes for services. In this scenario the client-DNS protocol does not change. Also no new DNS Resource Records are added for the first alternative, in which case service names will need to be mapped to follow the dotted hierarchical naming structure of host/domain names.

	Intra-domain	**Inter-domain**
Service Naming	First option: DNS naming (i.e. dot-separated domains, with each domain having an associated authoritative server and being able to delegate resolution to more specific servers). Alternatively, the use of the existing DNS is just a client interface/protocol and a new service naming scheme is adopted.	First option: DNS naming (i.e. dot-separated domains, with each domain having an associated authoritative server and being able to delegate resolution to more specific servers). Alternatively, the use of the existing DNS is just a client interface/protocol and a new service naming scheme is adopted.
Name Resolution	Offline	Offline
Service Forwarding	First option: Recursive or iterative DNS resolution as per today's DNS. Alternatively, a new type of iterative/recursive lookup is required (e.g. DHT) if a flat naming space is used and DNS reuse is limited to the client-server interface and protocol.	First option: Recursive or iterative DNS resolution as per today's DNS. Alternatively, a new type of iterative/recursive lookup is required (e.g. DHT) if a flat naming space is used and DNS reuse is limited to the client-server interface and protocol.
Service Control: Routing	First option: Authoritative name servers load their registers directly; DNS zone transfers can implement	Through the root servers, that associate domain names with specific commercial entities

	routing updates. Alternatively, unrestricted[5] if DNS is just a client interface.	that manage them.
Service Control: Signalling and Monitoring	Unrestricted.	Unrestricted.
Service Delivery	Directly through the IP routing/forwarding system	Directly through the IP routing/forwarding system

3.8.3.1.2 DNS inspired service resolution

This scenario offers offline, dynamic name resolution of service names to IP addresses. In that sense, it is similar to the DNS scenario, but it proposes significant changes to the ways DNS information is queried, maintained and updated: for example a client may subscribe to updates for a particular serviceID and the DNS server pushes matching resolutions as they are updated. A query for a given service name returns the IP address of a chosen instance running that service. In a first approximation, we do not assume any changes to the underlying IP routing substrate, and hence we assume that a normal IP unicast address is returned.

No constraints or naming are imposed a-priori. Hence, we assume that both hierarchical or flat naming schemes can be accommodated. This option represents an incremental change to existing architecture. Only small changes may be required client-side; mainly on the way applications use the resolution service.

	Intra-domain	**Inter-domain**
Service Naming	First option: DNS naming (i.e. dot-separated domains, with each domain having an associated authoritative server and being able to delegate resolution to more specific servers). Alternatively, unrestricted if DNS is just a client interface.	Domain orchestrators can advertise offered services. If needed, domain orchestrator redirects service request to other domain.
Name Resolution	Offline	Offline
Service Forwarding	Unrestricted; DNS is just a client interface.	DNS used as a client interface; pub/sub application-layer multicast system to communicate routing updates between servers
Service Control: Routing	Unrestricted; DNS is just a client interface.	Since name resolution and query response are decoupled, locality-aware DHT-inspired routing is well suited to the task
Service Control: Signalling and	Unrestricted	Unrestricted; Uses an underlying pub/sub application-

[5] The term “unrestricted” in these tables means that the scenario does not restrict this aspect of naming/forwarding/routing to one particular protocol or mode of operation at this level of abstraction.

Monitoring		layer multicast system
Service Delivery	Directly through the IP routing/forwarding system	Directly through the IP routing/forwarding system

3.8.3.1.3 Extensions to IP

This scenario is implemented with incremental changes to IP, preferably at the edge. Examples include: LISP, use of anycast addresses, use of unicasted anycast addresses, use of router alert option, new records to BGP, etc. It may still require DNS (e.g. to convert serviceID to anycast address). Service routing is done by IP.

	Intra-domain	**Inter-domain**
Service Naming	Unrestricted	Unrestricted
Name Resolution	Unrestricted (possibly offline)	Unrestricted (possibly offline)
Service Forwarding	Unrestricted; can involve hacks	Unrestricted; can involve hacks
Service Control: Routing	Unrestricted; can involve hacks	Unrestricted; can involve hacks
Service Control: Signalling and Monitoring	Unrestricted	Unrestricted
Service Delivery	Directly through the IP routing/forwarding system	Directly through the IP routing/forwarding system

3.8.3.2 Shared Responsibility between IP and Service Routing/Forwarding

Advantages/benefits:

- Late binding (without needing high-frequency updates or additional pub-sub mechanisms).
- Do not need per-packet service routing.
- Application sees this as similar/identical to native-service routing.

Disadvantages/challenges:

- Feedback/monitoring to service routers differs for the 3 cases below.
- The Overlay Routing: Response sent directly to the client scenario (section 3.8.3.2.1) needs monitoring information to be collected by a central monitoring system before routers are updated.
- In the Overlay Routing scenario (section 3.8.3.2.2) service routers are updated directly and can react faster but network wide summaries are more difficult.

3.8.3.2.1 Overlay Routing: Response sent directly to the client

In this scenario the client sends requests (indicating serviceID) to the service router. The service routing system resolves, routes and delivers request to the appropriate server. Finally the server sends acknowledgement directly to the client through the network. Data goes through the network without the intervention of service routers.

In this scenario the client will have to listen for replies that can be received from anywhere in the Internet. Practical considerations like NAT traversal also make this scenario of difficult deployment. However the impact on the service routing is minimised.

	Intra-domain	Inter-domain
Service Naming	Unrestricted	Unrestricted
Name Resolution	Online	Online
Service Forwarding	Overlay forwarding based on service names (e.g. DHT)	Overlay forwarding based on service names (e.g. DHT)
Service Control: Routing	Unrestricted	Unrestricted
Service Control: Signalling and Monitoring	Directly through the IP routing/forwarding system	Directly through the IP routing/forwarding system
Service Delivery	Directly through the IP routing/forwarding system	Directly through the IP routing/forwarding system

3.8.3.2.2 Overlay Routing: Response and control through the overlay

In this scenario, we assume that to invoke a service, the client sends request to its associated service router including a serviceID. Like the previous scenario, the way in which each client decides which service router it is associated with is for now left out of consideration. We assume that in this case the service routing system resolves, routes and delivers the service request to the chosen service instance. The actual setup of a service association, along with monitoring and signalling operations, are also propagated using the service overlay. The data path, however, is only routed through the IP substrate, thus reducing service forwarding load. Essentially the control plane is in the overlay and data plane is plain IP.

Like in Serval [NDGK12], the service instance will acknowledge the client request through the service overlay. Progress/status reports can also be sent through the overlay. Data, however, will come through the IP network. Note, however, that the control connection state is kept in the entire routing overlay while the data transmission takes place. That means that the routing substrate will keep state for every active connection it helps to establish.

	Intra-domain	Inter-domain
Service Naming	Although Instance names exist, only generic service names are injected into the routing plane.	Although Instance names exist, only generic service names are injected into the routing plane.
Name Resolution	Online	Online
Service Forwarding	Overlay forwarding based on service names	Overlay forwarding based on service names
Service Control: Routing	Unrestricted (e.g. Distance-Vector-Based compact routing)	Unrestricted (e.g. Distance-Vector-Based compact routing)
Service Control: Signalling and Monitoring	Through the service routing/forwarding system operating as an overlay over the IP routing/forwarding system	Through the service routing/forwarding system operating as an overlay over the IP routing/forwarding system
Service Delivery	Directly through the IP routing/forwarding system	Directly through the IP routing/forwarding system

One crucial aspect of this scenario is that each service will only be advertised once from each execution zone; hence, anycast operation is assumed natively. Moreover, this also means that when customer requests for a given service names are processed, they will be forwarded on the basis of service names and terminate on a given execution zone rather than with a particular instance. For stateless services, the assignment to instances can be performed probabilistically on the basis of load balancing considerations. For stateful services, however, a persistent association must be kept at the service layer between each session terminating at an execution zone and an specific service instance (this applies for conventional services in which state is not shared between instances. If one has an inherently scalable architecture like e.g. memcached/varnish with shared memory then the system can treat this as essentially a single instance).

An entity is hence required that operates similarly to a network-based HTTP load balancer. This entity will maintain a table that can associate each client with a running instance on the basis of the destination service ID and client information (e.g. the source IP Address and Port) . Since each packet at the service level will require multiple packets at the IP layer, service routers will require segmentation and reassembly and hence, per-service-packet state (at least for a limited amount of time). We will call this entity the zone service gateway. Of course, this gateway does not need to identify the associated service instance on the basis of an IP address, even though that example is the most direct one. Alternatively, this identification could be based on layer 2 labels or even other service-based names. This gateway can also provide other services that can aid with mobility and migration. In particular, it can operate like the VLR/HLR pairs that have traditionally used for roaming; alternatively, it may provide services akin to LISP.

3.8.3.2.3 SDN inspired

This scenario is inspired by recent developments in software defined networks. Client sends request (serviceID) to Service routers. The forwarding entries in service routers are configured by a centralised routing system – potentially co-located with the service orchestrator – see section 4.1 for a more comprehensive description of an integrated orchestration and routing system. Service routers deliver requests to the appropriate instance. Server sends acknowledgement directly to client through either the service routing/forwarding system or the IP routing/forwarding system. Data goes through the network without the intervention of service routers.

	Intra-domain	**Inter-domain**
Service Naming	Unrestricted	Unrestricted
Name Resolution	Online	Online
Service Forwarding	Overlay forwarding based on service names (e.g. DHT.)	Unrestricted
Service Control: Routing	Centralised algorithm	Unrestricted
Service Control: Signalling and Monitoring	Directly through the IP routing/forwarding system to a centralised control point	Unrestricted
Service Delivery	Directly through the IP routing/forwarding system	Unrestricted

3.8.3.3 Higher Responsibility placed on Service Routing/Forwarding

Advantages/benefits:

- More consistent with vision of native name-based routing in the Internet.
- Extremely late binding.
- Solving server and client mobility/migration.
- Dynamic mid-session load balancing.
- Service routers could perform other service-layer middle-box functions.
- Migration to clean-slate is more straightforward - no-need to piggy-back on IP routing layer.

Disadvantages/challenges:

- Inefficiency.
- Service layer routers process *every* packet at wire-speed (FIBs are larger hence more complex operations per packet).
- Increased stretch.
- Full deployment in the Post-IP scenario (section 3.8.3.3.2) with all routers in the Internet being upgraded/replaced is hard to imagine.

3.8.3.3.1 Overlay Routing: Data comes through overlay

Client sends request to Service router. Service routing system delivers request to server. Server sends acknowledgement through the overlay. Data also comes through the overlay.

	Intra-domain	**Inter-domain**
Service Naming	Unrestricted	Unrestricted
Name Resolution	Online	Online
Service Forwarding	Overlay forwarding based on service names (e.g. DHT.)	Overlay forwarding based on service names (e.g. DHT.)
Service Control: Routing	Unrestricted	Unrestricted
Service Control: Signalling and Monitoring	Through the service routing/forwarding system operating as an overlay over the IP routing/forwarding system.	Through the service routing/forwarding system operating as an overlay over the IP routing/forwarding system.
Service Delivery	Through the service routing/forwarding system operating as an overlay over the IP routing/forwarding system.	Through the service routing/forwarding system operating as an overlay over the IP routing/forwarding system.

3.8.3.3.2 Post-IP

Clean slate addressing “CCN-like” service oriented layer 3. Queries forwarded by service routers. Layers 3 and above replaced in end hosts. Layer 3 in all routers also changes. This is CCN inspired but with service specific differences. Could be hierarchical or flat naming (although flat naming causes scalability problems since every router might have to be aware of every instance of every service).

	Intra-domain	**Inter-domain**
Service Naming	Unrestricted	Unrestricted
Name Resolution	Online	Online

Service Forwarding	Direct forwarding based on service names (e.g. CCN)	Unrestricted
Service Control: Routing	Unrestricted	Unrestricted
Service Control: Signalling and Monitoring	Through the native service routing/forwarding system. This option does not require an IP routing/forwarding system.	Through the native service routing/forwarding system. This option does not require an IP routing/forwarding system.
Service Delivery	Through the native service routing/forwarding system. This option does not require an IP routing/forwarding system.	Through the native service routing/forwarding system. This option does not require an IP routing/forwarding system.

3.8.3.4 Discussion on Service Routing Deployment Options

The post-IP option, section 3.8.3.3.2, is considered unrealistic for deployment in the short to medium term due to it being highly disruptive at a high cost to ISPs. The option of using DNS as currently deployed, section 3.8.3.1.1, has the disadvantage of not easily fitting the requirement for resolving serviceIDs without compromising the service naming scheme to fit existing DNS resource records. Even if new resource records are defined for FUSION-compatible serviceIDs and the client-DNS server protocol could remain intact, new functionality for resolving and forwarding queries is required for DNS servers to act as service routers and the benefits of retaining the existing DNS protocol are limited. Overlay Routing: Response sent directly to the client, section 3.8.3.2.1, has an asymmetry of communications with the client receiving invocation acknowledgements from an IP address it has not directly solicited which causes potential problems with NAT traversal which would require inelegant solutions. Overlay Routing, Monitoring and Signalling, section 3.8.3.2.2, has advantages of allowing the overlay to be aware of status reporting but has the disadvantage of requiring state to be maintained in the overlay for all service sessions.

For the above reasons our initial analysis of the networking options has resulted in the decision to keep features from the following scenario options:

- DNS-inspired service resolution, section 3.8.3.1.2, for use in two cases:
    - Firstly, where the FUSION service routing plane resolves serviceID to a specific execution zone/service instance and returns one or more IP addresses to the client for subsequent selection and invocation.
    - Secondly, where the FUSION service routing plane forwards a client's invocation request to a selected execution zone and the service instance is invoked directly, with the service response being routed back via the service routing overlay. However, we do not foresee this option being used for either long-lived sessions or request-response sessions where the quantity of returned data is excessively large. This is due to the inefficiencies and performance degradations of routing such quantities of data through an overlay or for maintaining session state throughout the overlay for a lengthy amount of time. In such cases the query resolution mode of operation is preferred with subsequent data-plane exchange being undertaken directly through IP.
- SDN-inspired, section 3.8.3.2.3, for the case when there are no running service instances and the service routing plane should invoke orchestration (or a zone manager) to deploy an instance on demand.

- An exploratory native layer solution based on the Extension to IP, section 3.8.3.1.3, using the control plane of IPv6 anycast to implement a subset of the FUSION service networking requirements with a minimal modification to today's Internet fabric. This can also be used as a comparison of cost/benefit with the main overlay-based solutions.

The outcome of a more detailed investigation of the options for the FUSION service networking protocols is discussed in D4.1 [D4.1].

3.9 Software Architecture

Different aspects of FUSION (e.g. FUSION orchestration versus communication/routing) will be implemented as independent software modules:

1. The software interfaces are independent (except for using small optional utility libraries).
2. The implementation of one module may use another module, but may only use public interfaces (“Eat your own dog food”) of other modules.

Of course the interfaces of different modules can and should be designed to work together smoothly, but without specifically designing an interface to work primarily with a particular other module.

The following diagrams give an overview for example about decoupling the orchestration from communication and routing at the level of execution points.

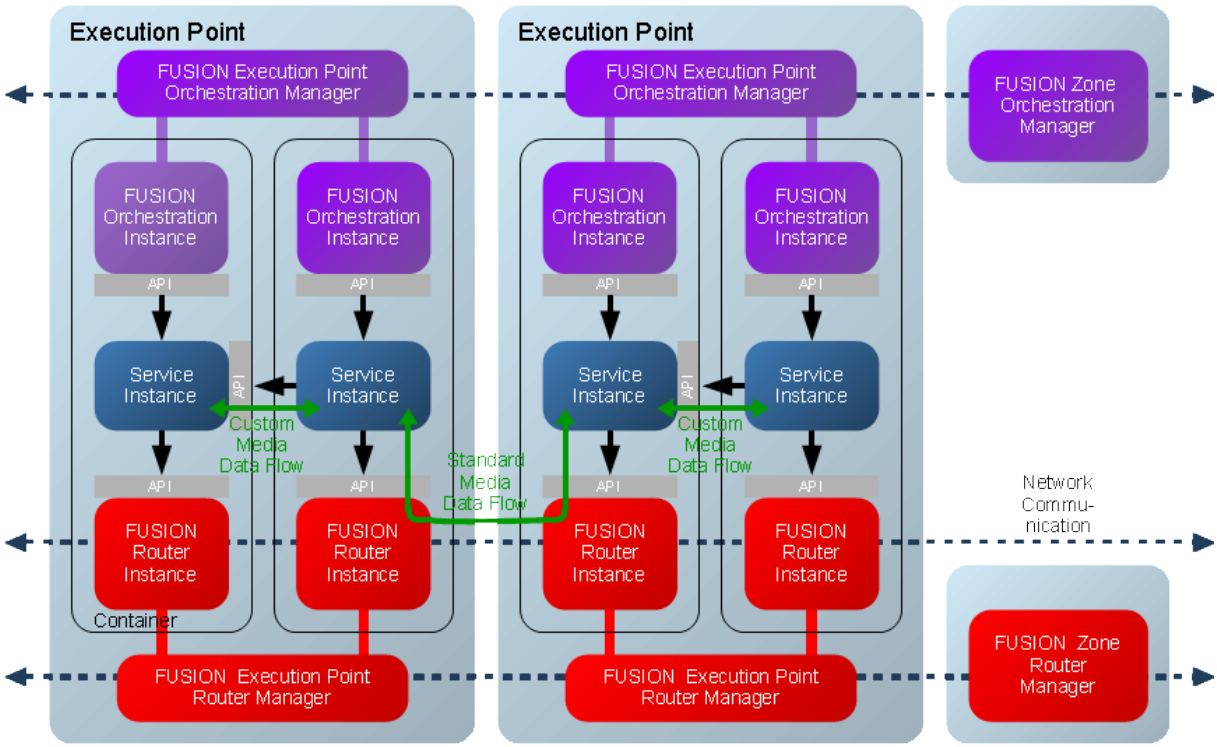


Figure 16: FUSION Software Module Architecture

The core aspects are:

- While service are managed by the orchestration part of FUSION, services should have free choice of using completely proprietary communication (e.g. passing GPU object handles for most efficient communication if the involved service instances run on the same machine, labelled "custom media dataflow" in the diagram), or using a communication and routing module of FUSION (labelled "standard media dataflow" in the diagram).
- The communication and routing module of FUSION communicate with the orchestration part of FUSION only via public interfaces which are available to any user of FUSION. For example, the proprietary communication implementation can use the same interfaces as the FUSION standard communication implementation. The FUSION standard communication implementation does not have any “unfair advantage” by using any internal interface of the FUSION orchestration layer, which in turn would mean disadvantage for any proprietary communication channel which does not have that advantage.

- As consequence this means that a strict decoupling forces the development of FUSION itself to be open for future users of FUSION which want or have to implement proprietary functionality on top of FUSION.

The corresponding overall view of such a decoupled architecture in case of orchestration versus communication may look like the following:

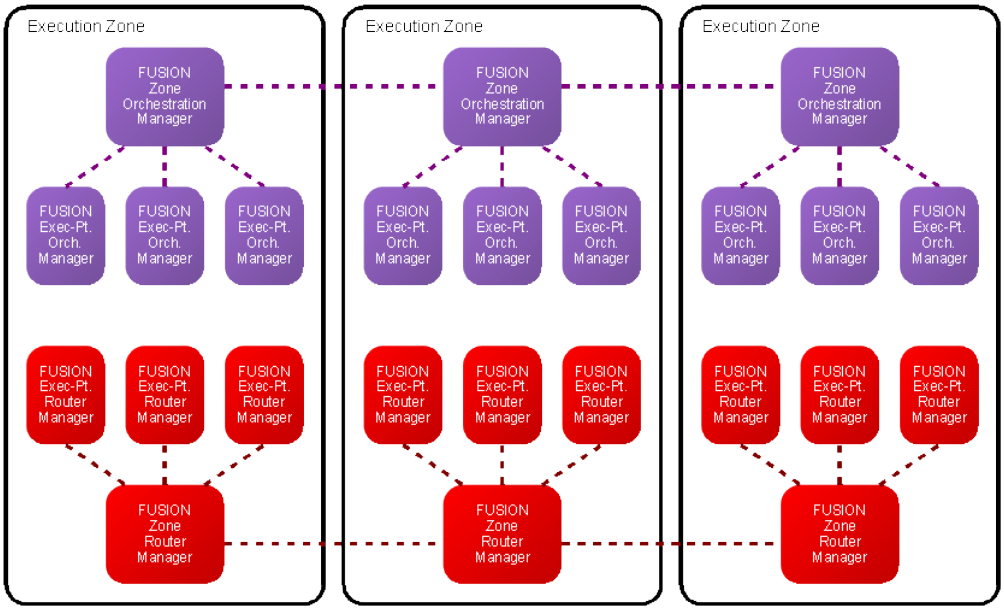


Figure 17: Routing and Orchestration Software Module Architecture

Implementing FUSION in decoupled components means from the software architecture perspective that there should be no functionality in a core module which is required to be used by multiple other modules as visualised by the following diagram:

Figure 18: Coupled vs. Decoupled Software Architectures

The decoupled approach is more industry-friendly and future-proofed, since FUSION is supposed to be not a closed package, but a foundation for many commercial users having different demands and requiring proprietary custom extensions. User of FUSION can best selectively replace functionality (e.g. direct communication channel or own routing) with lowest probability of hitting walls (because the implementer had to design the public APIs powerful enough for all own purposes). The decoupled approach also benefits from all standard advantages of modular software development. The disadvantage of a strict decoupled approach is that it causes more work for developing FUSION by enforcing very cleanly decoupled interfaces[6].

[6] For example, FUSION partner Spinor used a modular, plugin-based software architecture in v4 of their Shark 3D software, resulting into a growing number of difficulties for customers who wanted to implement certain custom features which Spinor couldn't anticipate. Also the software development process suffered

4. INTEGRATED AND DISJOINT ORCHESTRATION AND ROUTING: A COMPARATIVE ANALYSIS

4.1 Integrated Orchestration and Routing (IOR)

In this section we explore the approach where the *FUSION orchestration* both manages the service deployment (service instance placement) and the population of the routing tables of each service router in a domain. We study related work on service routing and describe the FUSION architecture and operation flow in detail. We will show that this FUSION approach is able to solve the problems discovered in related work. As a discussion point, we also take a look at an alternative approach where the population of the service forwarding tables is managed by service routers, rather than by the orchestration.

Consider a user requesting a service hosted in a FUSION domain through its service identifier. One of the key challenges in FUSION is to forward this request to the best service instance while maintaining the desired QoS. Related research brings forward two approaches: in-band forwarding of identifiers, or out-of-band name resolving.

The first approach is used in CCNx [JSTP09] amongst others. In-band routing allows users to express requests for content (*called* interests) by name, rather than by locator, straight onto the network, while the service routers are responsible for selecting the most suitable service instance. This in-band forwarding is done by mapping the service name to the next hop address on the path. In CCNx, the routing tables are gradually constructed by adding forwarding entries for a given prefix. If no entry is found in the routing table for a particular content identifier, the request is dropped. Replies for pending interests are cached in the router, so subsequent requests can be served by the router itself. In the current CCNx implementation these forwarding entries must be added manually through configuration files. As a result, there is no real routing mechanism in CCNx: either the request is served by the router, either the request is dropped.

The current CCNx approach for routing table construction cannot be applied to FUSION. There is no intelligent mapping mechanism: content requests are simply flooded on all interfaces. In FUSION, we seek an intelligent resolution of service names to the best instance, based on context parameters such as client profile, device characteristics, network statistics, etc.

The second approach, out-of-band name resolving or also called publish-subscribe, is used in PSIRP [JZEA09]. This approach resembles that of DNS: requests are first sent to a broker who selects the most appropriate destination based on multi-dimensional metrics. This approach introduces an additional delay during the lookup phase, resulting in higher latency for each service request.

In the next section we will sketch the architecture of FUSION, after which we describe how FUSION solves the above mentioned problems.

4.1.1 Intra-Domain Architecture

Consider a FUSION overlay network consisting of *execution zones*, services routers and a FUSION orchestration. To make our suggested approach scalable we introduce a FUSION domain, which is the collection of execution zones and service routers managed by one FUSION orchestration. In each FUSION domain the assigned orchestration monitors traffic on the overlay network, deploys service instances in execution zones, and updates the forwarding tables of service routers.

---

from more and more difficulties adding certain larger new features. With v5 (2005) Spinor switched to the modular-decoupled philosophy, which stands the test for eight years now without suffering from limited/legacy architectures without customers or Spinor itself hitting architectural limits.

The orchestration is a logically centralised component: it monitors execution zones in its domain and forwards instructions to zone managers and service routers. The orchestration can be implemented as a centralised entity, e.g. a zone with enough computing resources to manage its domain, or in a distributed manner as a set of cooperating zones and service routers. If the orchestration is implemented in a distributed manner, each zone or service router operating as part of the orchestration listens to the same multicast but only forwards instructions to nearby zones or service routers that it manages. In the remainder of this section we will refer to the orchestration as a logical centralised entity, as shown in Figure 19.

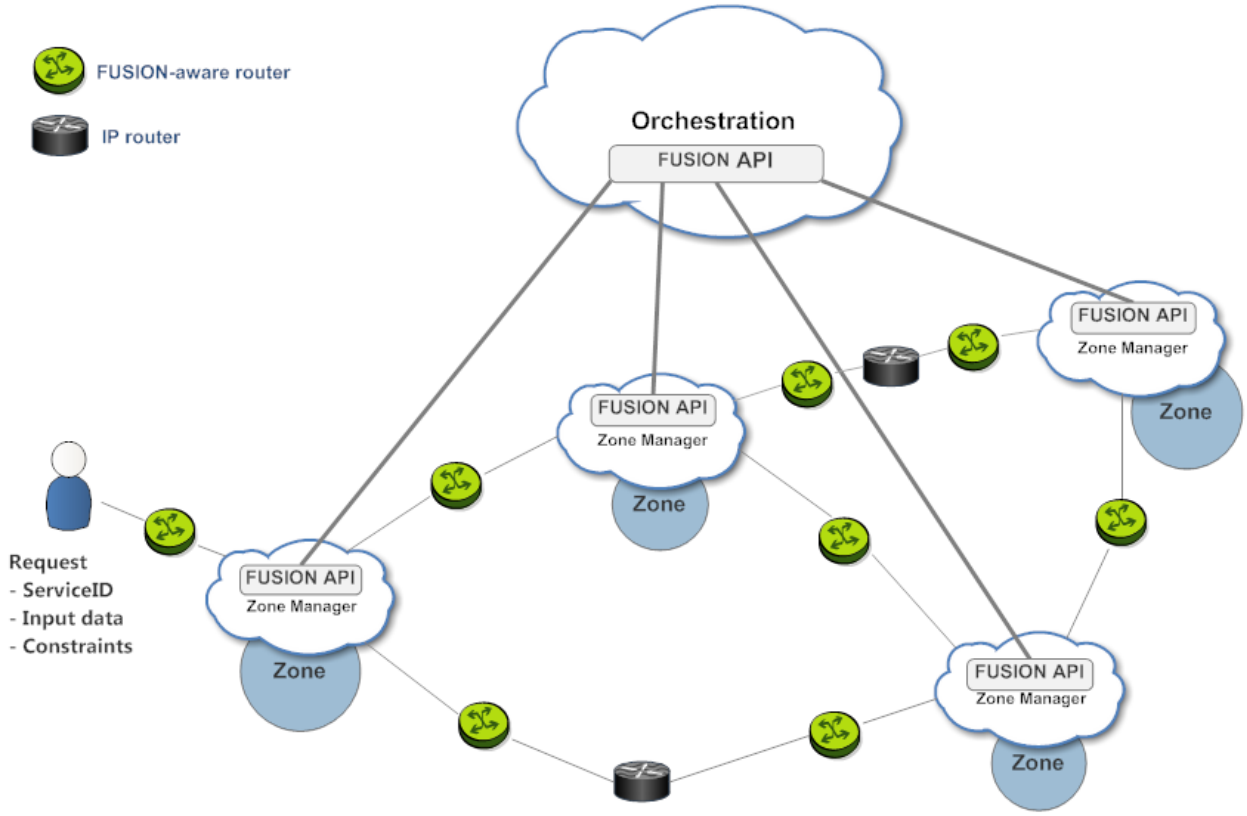


Figure 19: IOR Intra-Domain Architecture

We defined an execution zone as the smallest entity where the orchestration can deploy service instances. On the overlay network, requests for a service instance hosted in a certain destination zone are forwarded along the path to the service router in that zone. Once a request reaches the destination zone, it is further processed by the zone management protocol. This abstraction allows zone owners, such as an ISP or data centre owner, to run their current management software inside the zone while still participating in the FUSION overlay network. This will ease the transition to FUSION and reduce the integration cost.

FUSION specifies an API for exchanging control traffic between the execution zone and orchestration. The orchestration uses this API to communicate with a zone and forward instructions on. The component implementing the API in the execution zone is called a *zone manager*. A zone manager monitors its internal resources and builds a global zone representation, representing the available resources and current zone state in a manner understandable to the orchestration. As the orchestration does not have direct control of the zone infrastructure, it delegates instructions to a zone manager, which in turn will handle the local execution in that zone. The orchestration uses overlay monitoring information to find the best zone to deploy a new service on, but the zone manager chooses the physical resource to start the process on. As the orchestration does not know the address of that internal zone resource, it must also delegate forwarding table updates to the

zone manager. Thus, the zone manager is also responsible for populating the intra-zone service routing, as instructed by orchestration.

4.1.2 Example Routing/Orchestration Flow

The orchestration relies on monitoring information to make decisions concerning service placement and routing optimisation. Figure 20 illustrates this process: first an internal zone monitor protocol is used to gather information about local resources (1). This could be a simple log of the local resources available in a zone, but it can also be a larger scale monitor framework. The zone manager calculates a score representing the available resources of that zone (e.g. the amount of active connections per service instance in that zone) and forwards this to the orchestration. Transfer (2) represents the monitoring information on available zone resources to orchestration. After monitor information is gathered, the orchestration processes the information (3) and uses this as input for analysis and decision algorithms (4). These decisions are classified as either deployment or service routing instructions and delegated to the appropriate component (5). Service routing instructions are only sent when forwarding entries need to change (e.g. add, edit or delete); this approach saves bandwidth usage.

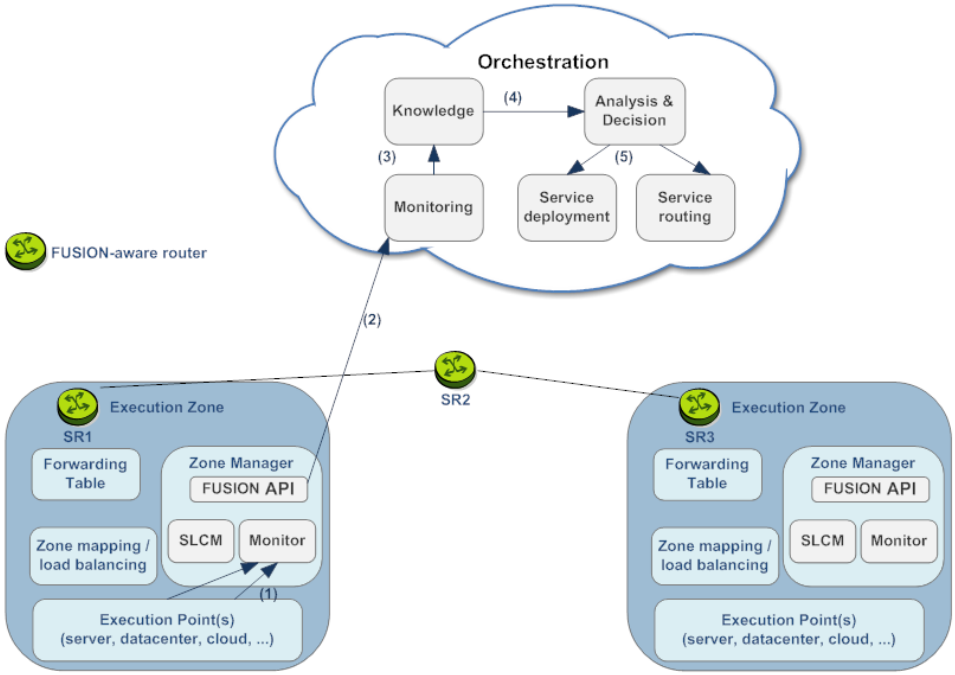


Figure 20: IOR Monitoring Flow

The zone availability together with known link characteristics is sufficient for the orchestration to find the path to the optimal service instance for each service router. This method guarantees that each user request will be forwarded by service routers along the path to that optimal service instance. FUSION excels from existing routing protocols by considering both network and host characteristics, which is needed to reach the desired QoS from services.

When a zone's workload reaches critical values or no service instance is located nearby users requesting it, the orchestration will make optimisations in the overlay network. In the first case, new incoming requests can be forwarded to an alternative service instance located in another zone to perform network load-balancing. In the second scenario, an additional instance could be deployed near the user to reduce latency to a minimum. In both cases, the orchestration will send the new forwarding entries to all service routers affected by this. In case of a zone, the orchestration will instruct the zone manager to update the zone's service router. Instructions can be forwarded using the overlay network or via a pre-established path, as illustrated in (1) of Figure 21.

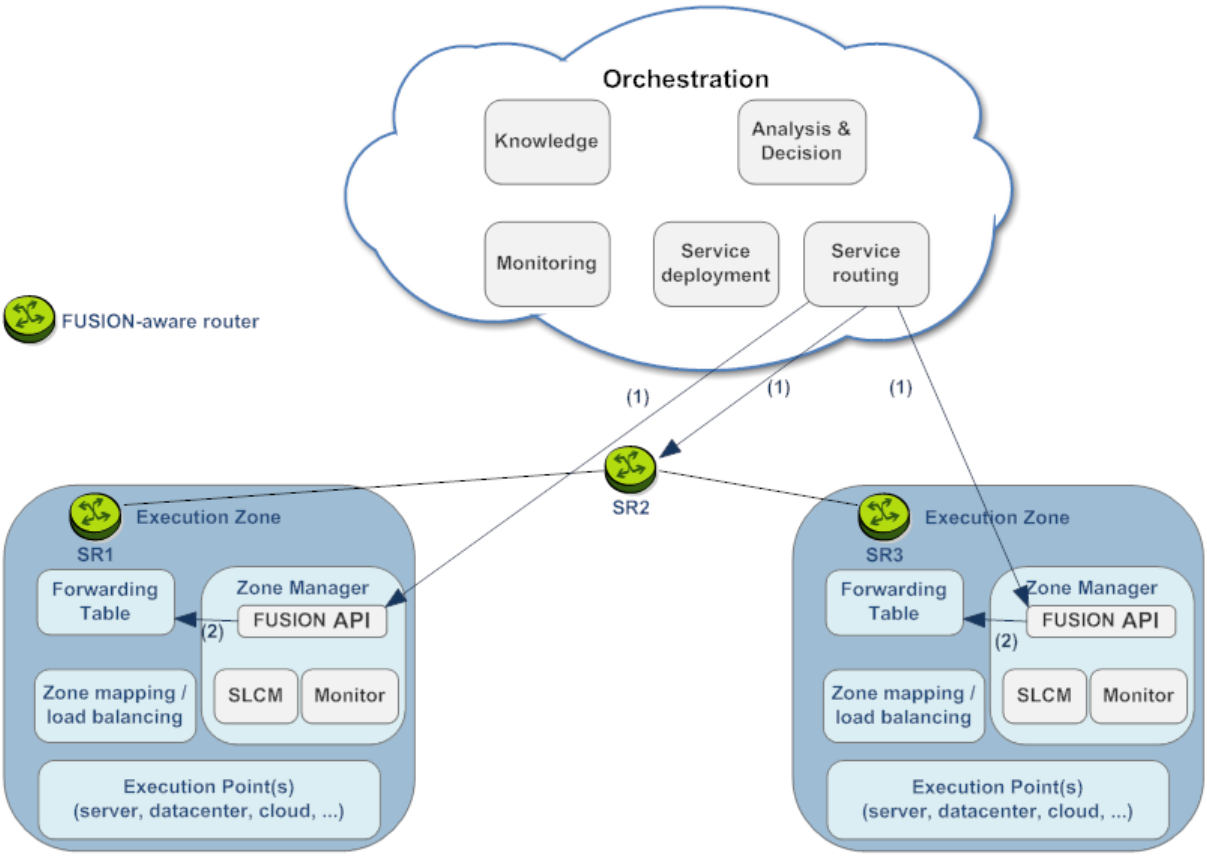


Figure 21: Forwarding Service Routing Results

In the previous sections we have not explained in detail how a zone manager executes service deployment requests from the orchestration. Once the orchestration forwarded a service instance execution instruction (1) to a zone manager and received confirmation about the execution, it will assume that all user requests arriving in the zone will be answered correctly. The zone manager has a *service life cycle management* (SLCM) component which handles these instructions (2). The SLCM can deploy several instances and use an internal load-balancer to utilise all instances for incoming requests (3). These inter-zone decisions are hidden from the domain level FUSION components, due to the abstraction of an execution zone. It is also possible that a zone chooses to use an IP-based protocol to address its execution points, which may in turn be physical or virtual resources. This is possible on condition that the incoming FUSION requests are accepted and that outgoing answers are converted back to a FUSION packet before leaving the zone. The next hop field in a service router's forwarding table contains the address of the next service router on the path. The final hop in the overlay network will always be a service router located in a zone hosting the requested service instance. Therefore, a zone must implement an internal request converter to convert name-based requests to IP-based ones. This is necessary for the request to reach the execution point running that service. As shown in Figure 22, the zone manager updates its service router's forwarding table after deploying an instance, as this address is not known to the orchestration. Only when the request reaches the destination zone it will become known which physical resource it must be processed on.

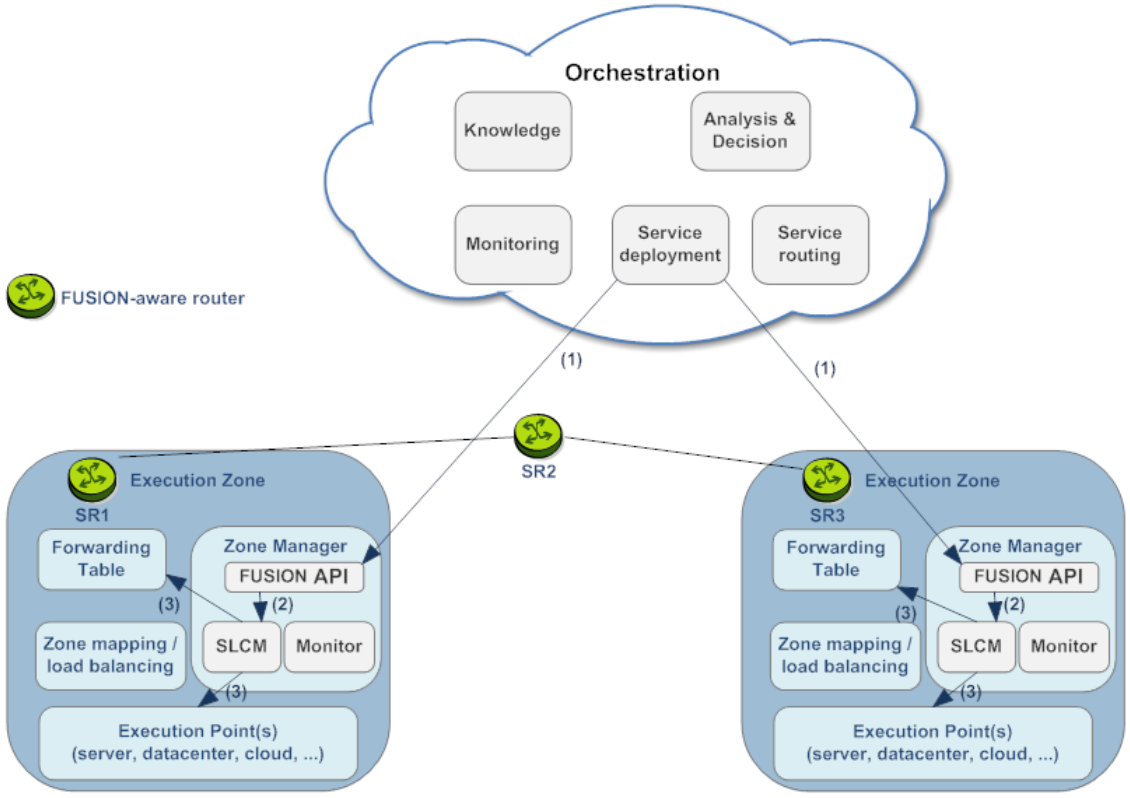


Figure 22: Orchestration-driven Service Deployment

4.1.3 Benefits of Integrated Orchestration and Routing

Delegating both service deployment and service routing to the orchestration enables FUSION to not only optimise service deployment on-the-fly, but also manage overlay forwarding in an efficient matter. The FUSION orchestration is able to use correlation between the service and network layer to optimise deployment decisions, as most routing updates origin from service deployment and migration.

FUSION should minimise the overhead generated by exchanging (service-instance) topology information. In IP-based protocols such as OSPF, routing table updates only occur when a link or neighbour goes down or joins the network. In FUSION, deployment decisions are based on more extensive information such as the amount of active clients a service instance has, or delay measured between two execution zones. These characteristics change at much higher rates, drastically increasing the amount of exchanged monitoring information. The suggested FUSION approach reduces flooding in a matter similar to OSPF Designated Routers and iBGP Route Reflectors; all routers and execution zones multicast monitoring information destined for the FUSION orchestration. The orchestration in turn relays instructions to service routers and zone managers using either the overlay network or pre-established links reserved for control traffic.

Routing algorithms and service deployment algorithms located on the orchestration both utilise this monitoring information as input. The orchestration is also able to utilise its centralised role to take more sophisticated decisions; e.g. prediction algorithms can be used to preemptively deploy service instances but withhold service routing updates until these instances are needed. This would be a much more complex problem when all routers and execution zones must agree on such decisions beforehand.

As mentioned in the previous section, out-of-band name resolving introduces increased latency. FUSION avoids this problem by decoupling control traffic and data traffic. Overlay monitoring information is used by the orchestration to optimise service placement and update service routers'

forwarding tables. User service requests follow active forwarding entries on a service router, which guarantees all requests will follow the path leading to the best service instance for that request.

4.2 Disjoint Orchestration and Routing (DOR)

4.2.1 General Architecture

This section gives an overview of how a FUSION system with separate orchestration and routing layers. The following figures show the interactions between functional blocks for service deployment, service routing as well as for performance monitoring. Sections 4.2.2 and 4.2.3 highlight the pros and cons of disjoint orchestration and routing compared to the integrated case discussed in the previous section.

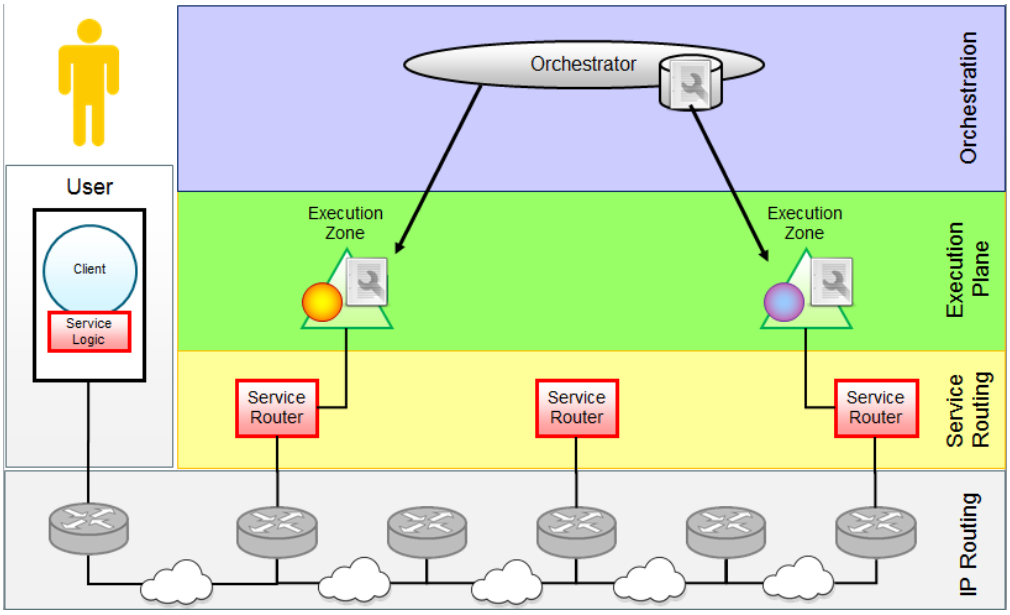


Figure 23: Service Deployment

Figure 23 shows the orchestration of service deployment. Service instances are installed in the execution zones – the figure depicts two different services – yellow and purple – being deployed in separate execution zones. Each is identified by a different serviceID. The service deployment phase is similar in the disjoint and integrated orchestration and routing cases.

Figure 24 shows how the service routers are populated with information about the running service instances. Execution zones announce serviceID availability (and number of available session slots) to their local service router. Service routers exchange routing information with one another in a distributed fashion (via either a distance-vector or link-state equivalent routing protocol) and populate their local forwarding entries with serviceIDs. The nature of the anycast routing algorithm and the way selection will be made between multiple instances of the same serviceID in different execution zones is not dealt with here – this is future work to be documented in year 2 of the project. Updates are announced and propagated when serviceIDs are introduced, taken out of service or when the number of available session slots changes (with appropriate mechanisms in place to avoid highly dynamic updates, such as announcing longer-term averages of available session slots for a serviceID within an execution zone).

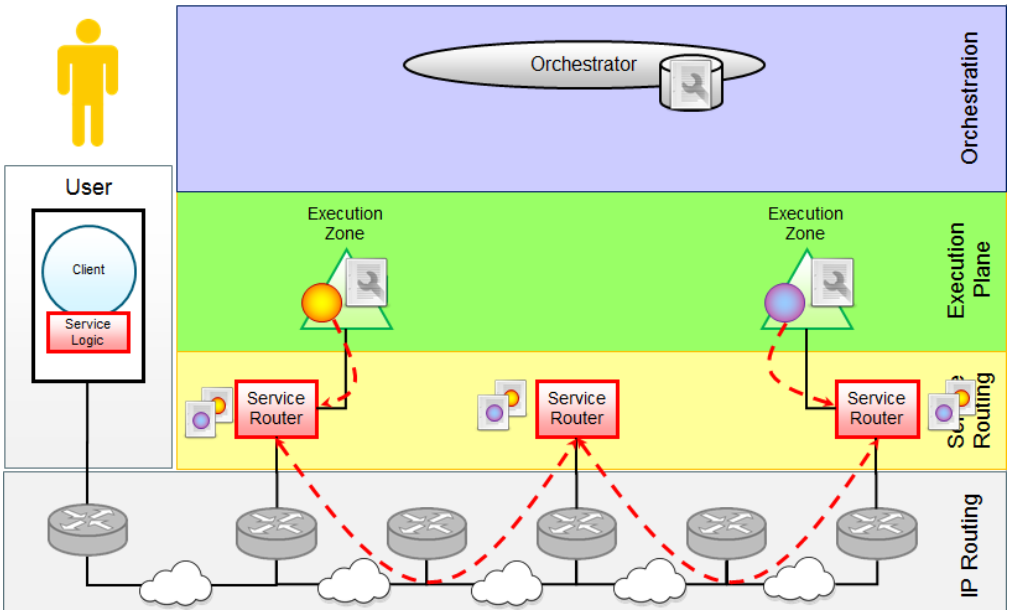


Figure 24: Routing Announcements

Figure 24 shows the main difference of disjoint orchestration and routing compared to integrated orchestration and routing: in the disjoint case the service routers run a distributed routing algorithm and configure their own local forwarding tables, whereas in the integrated case the routing function is centralised and co-located with the orchestrator with the forwarding entries in the service routers being configured remotely (cf. Figure 21).

Although these figures do not show multiple service routing domains routing updates would also be exchanged *between* domains in a similar manner to the updates shown in Figure 24. However, as is the case with BGP routing policies today, not all routing information on all serviceIDs would be exchanged with peers.

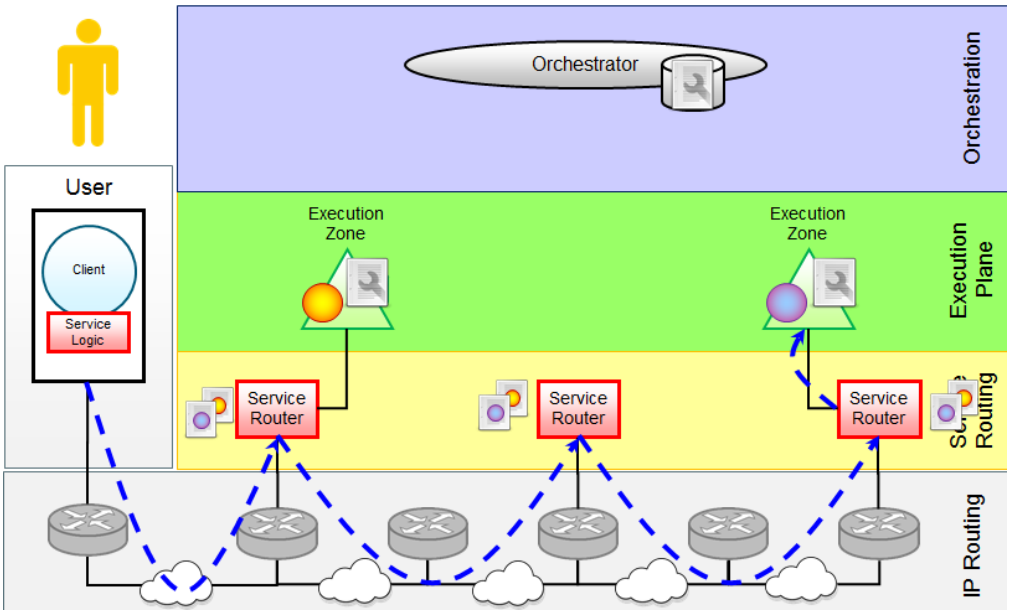


Figure 25: Service Invocation Routing

Once the routing protocol has exchanged routing updates and forwarding tables have been configures users may issue query/invocation requests to their local service router, see Figure 25. In the figure the user has requested the purple serviceID and the query/invocation request is forwarded to the relevant execution zone through one or more service router hops.

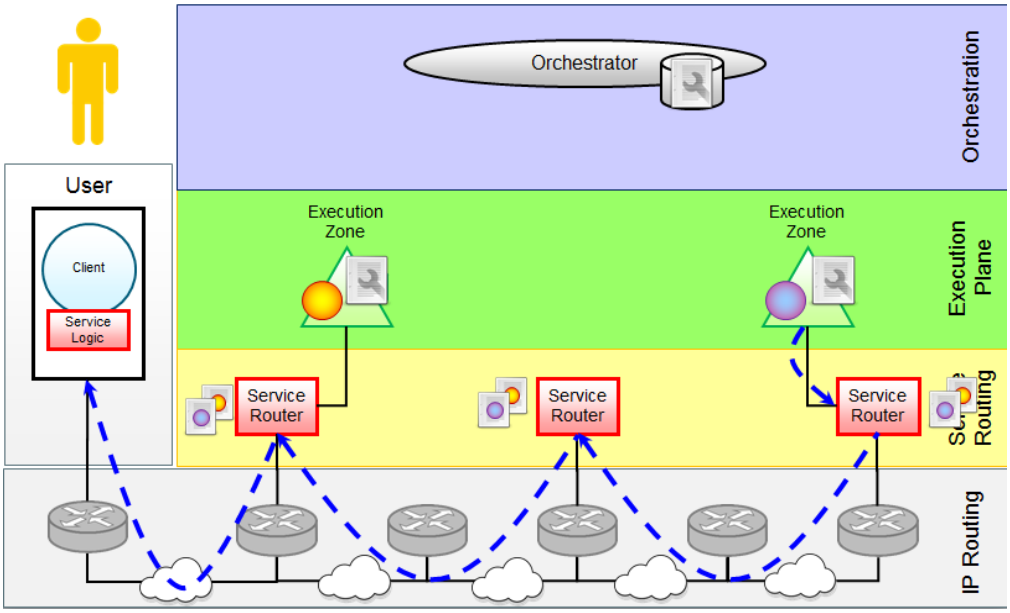


Figure 26: Service Response Routing (via overlay)

Figure 26 shows the service response being routed back to the client via the overlay. This corresponds to the service networking option described in section 3.8.3.3.1 (Overlay Routing: Data comes through overlay), and as described in the discussion in section 3.8.3.4 this is only considered feasible for short-lived sessions with small data requirements. Longer lived sessions or those with heavy amounts of data to return will communicate directly through IP as depicted in Figure 27.

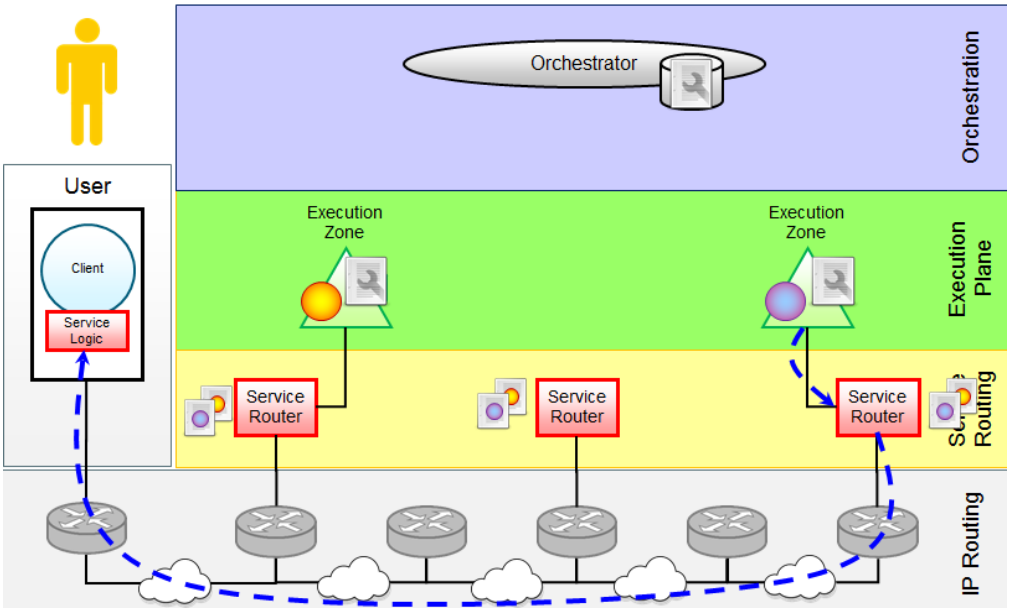


Figure 27: Service Response Routing (through IP)

The DNS inspired service resolution option, discussed in section 3.8.3.1.2, is not shown in the above figures. However it would work in a similar way to Figure 26 with one of the service routers along the path (not necessarily the first one) resolving the serviceID and returning the IP address back to the user through the overlay. The subsequent data plane communications would be routed as shown in Figure 27, initiated by the user.

4.2.2 Benefits of Disjoint Orchestration and Routing

In the integrated view, orchestration is a generic resource allocation platform that has control over both endpoint and network resources. Hence, it can effectively optimise over both. If there are conflicting requirements between say the network and the execution zones, it can solve this trade-off directly as a multi-objective optimisation problem. However it should be noted that this multi-objective optimisation could be complex, even within a single domain. At Internet scale it may require constraints to be relaxed so much for a feasible heuristic-based solution that solutions may be very sub-optimal.

- Inter-domain operation with disjoint orchestration and routing is more realistic as it avoids the need for a central global routing manager. This is important not only due to scalability issues, but because individual business entities need the freedom to implement policies independently.
- If inter-domain is achieved in the integrated case by interworking between local orchestrators/routing managers in each domain then we have moved to a distributed routing system anyway, at least for the inter-domain aspect of routing.
- In the separate view, orchestration and routing are doing two completely different things:
    - Orchestration is deploying services, optimising placement to match forecasted demand given execution zone location/availability.
    - Routing is discovering the best service instance out of those currently running and forwarding forwards requests to the next hop according to the tables created by routing.
- Orchestration and service routing may be run by different stakeholders with different interests that are not aligned.
- Dynamic network changes can be accommodated only by routing; this simplifies orchestration and makes churn much less of a problem.
- In the centralised case, the optimisation problem is exceedingly hard. This may necessitate approximations that will generate an *optimality gap*. It may very well be the case that this gap is comparable to that of the decentralised approaches, obviating the benefits of centralisation.

4.2.3 Drawbacks of Disjoint Orchestration and Routing

When considering the advantages of a separate architecture, it is useful to consider the following issues:

- Flooding routing information is wasteful.
- Orchestration decisions usually result in accompanying routing changes.
- Monitoring information is collected by the orchestrator anyway, so the need to flood similar information between service routers in separate routing updates is questionable.

4.3 Conclusion on Integrated vs Disjoint Orchestration and Routing

In the integrated case there is more control over routing decisions allowing them to be optimised more readily for specific service deployment patterns and adapted dynamically according to complex rules and algorithms which would be difficult to control in a distributed routing system. These advantages hold provided that the orchestration domain and service routing domain are integrated business entities (see also section 6 on business model considerations for FUSION). In alternative business models where routing domains and orchestration domains are not controlled by the same entity then a close cooperation of business entities for configuring routing/forwarding tables is less likely. This becomes more of as issue as the scope of service access is scaled to global coverage. So while integrated orchestration and routing is feasible and potentially more efficient at optimising

operations this has some bounds in terms of geographical coverage and, beyond a certain size, interworking between domains becomes inevitable. For this reason both options will be explored by the project to investigate the opportunities for optimal configuration of single domains (e.g. the ISP-centric model discussed in section 6 (Figure 34) as well as to study techniques for scalable service routing between domains in alternative business models, such as the OTT business model with a common service routing plane (Figure 36).

5. APPLICATION PROGRAMMING INTERFACES

In this section, we will present the key FUSION API functions at all levels of the FUSION system. As many of these API functions are part of overall FUSION core functions that span multiple layers, we start this section by describing these functions and highlight the role of each entity involved in some part of this function. Afterwards, we zoom in on the corresponding key APIs per layer or entity. We will not focus here on the local and/or implementation dependent APIs.

5.1 Key FUSION Functions

In this section, we will repeat the key FUSION functions and entities or stakeholders involved in many of these functions, present their high-level roles in a matrix form, followed by a more detailed graph-based representation of each of these functions, identifying the key interactions that result in the FUSION APIs that are described in the later sections.

5.1.1 Overview

As a reminder, we first present a non-exhaustive list of the key FUSION functions:

- Service registration
- Resource registration
- Service state management (includes instantiation, deployment, etc.)
- Service monitoring
- Resource monitoring
- Service mapping
- Service request
- Communication
- Routing

Below a list of the key FUSION entities and stakeholders that are involved in many of these core functions:

- Domain orchestration
- Execution zone
- Execution point
- Service router
- Service instance
- Client

Each entity can have a number of roles in each of the core FUSION functions. For example, the service routers are not only used to forward many of the messages corresponding to specific API calls between two entities, it also is involved in other functions, including network resource monitoring, etc.

Table 3 shows at a very high-level the potential role(s) of all entities with respect to the specific function.

	domain	exec. zone	exec. point	network	instance	client
service registration	register itself handle/manage	-	-	register SR forward/route	-	register new service type
resource registration	handle/manage	register zone handle/manage	register	register forward/route	-	register
service state mgmt.	supervise/trigger (prelaunch/LB)	handle/manage	provide	forward/route	(re)act	trigger
service monitoring	aggregate	aggregate	-	forward/route	probe (FPS, latency)	probe (roundtrip lat.)
resource monitoring	aggregate	aggregate	probe (quotas,usage)	probe aggreg./update forward/route	-	-
service mapping	Supervise	handle (provide score)	-	forward/route	estimate (calculate score)	-
service request	handle (on-demand scenario)	handle (on-demand scenario)	-	forward/route balance	accept/act/reply	trigger
comm. protocol	-	handle/manage (late binding)	support manage/control (e.g.: limit bw)	forward/route handle/manage (late binding)	use	use
routing	Supervise	supervise	-	handle/manage implement	-	-

Table 3: FUSION Architectural Functions

Further aspects of investigation concerning these functions and corresponding API calls include the definition of the communication models: push, pull, publish/subscribe, etc. These will be refined as the APIs are designed and implemented in the second year of the project.

In the next sections, we will now elaborate on each FUSION function, presenting the interaction graph.

5.1.2 Resource and Service Monitoring

In Figure 28, we present the monitoring interaction graph, depicting all entities, their relation with respect to each other and the communication flows between them.

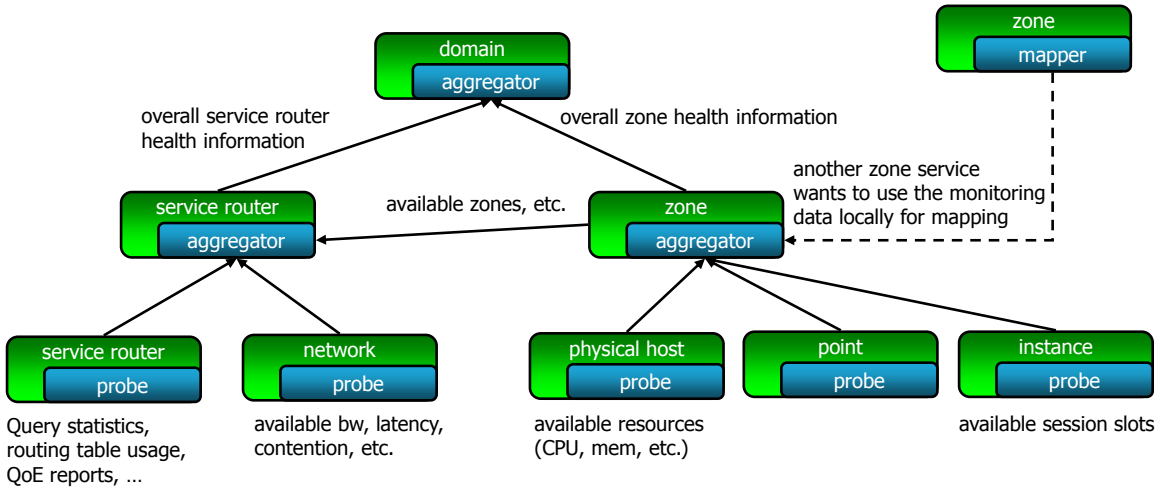


Figure 28: Monitoring Interaction Graph

In the above figure, each arrow represents a potential API call. However, some of these are important with respect to the overall FUSION architecture, whereas others are more implementation-dependent. For example, how execution zones will pass their information to the domain is a key API function, whereas the way the physical host probes pass their information to the zone manager is up to the zone implementation (if it even contains a physical host probe). The former type need to be formalised and worked out in detail, whereas the latter ones can just be exemplified, for example in the context of a prototype specification and implementation.

5.1.3 Service Requests

For this function, we will distinguish between the available-instance and the on-demand scenario. The former scenario is depicted in Figure 29 below.

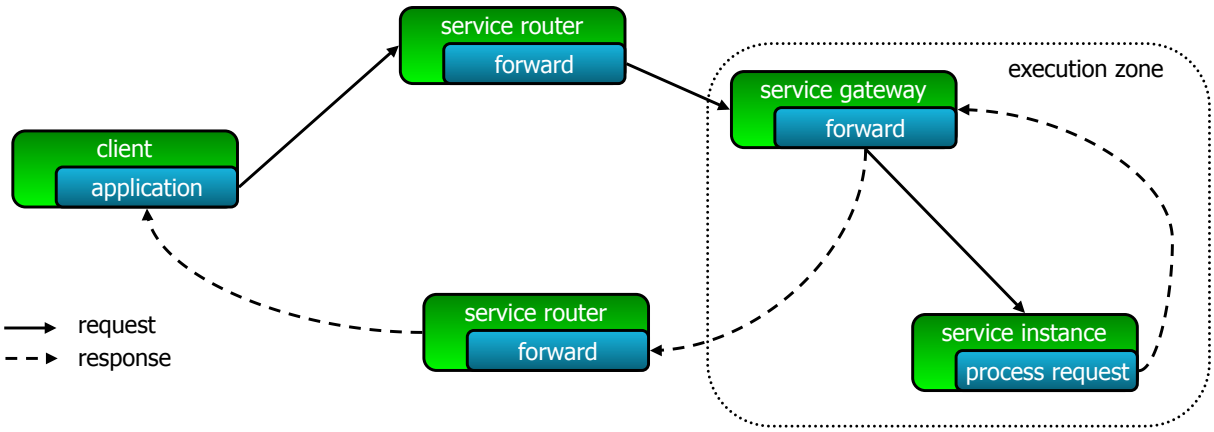


Figure 29: Available-Instance Scenario

The on-demand scenario is depicted in Figure 30. Although this scenario looks very similar with the previous one, the service mapping function itself involves many different functions, including the service registrar, the evaluator functions in the selected execution zones, service deployment, and instantiation, and setting up the monitoring for the new instance. Each of these are discussed on their own in their corresponding sections

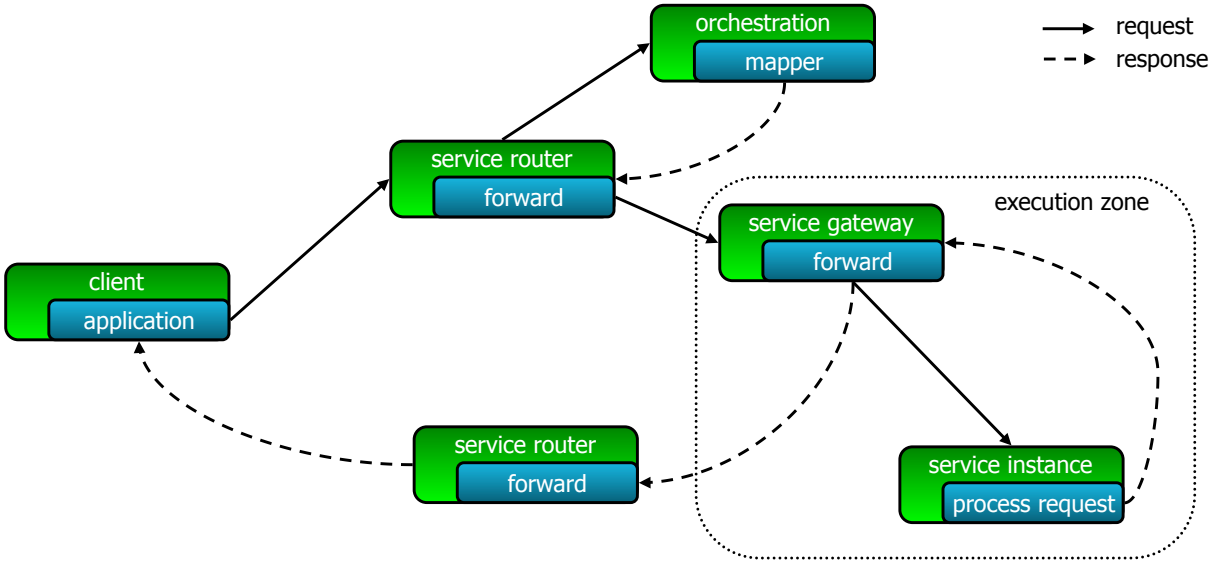


Figure 30: On-Demand Scenario

5.1.4 Service Placement

Service mapping is the act of finding the optimal location to create a new instance for a particular service type, taking into account the various instantiation parameters. In a first step, the

orchestration service needs to fetch the service manifest from the registrar service. Based on some high-level information, the domain-level mapper service will make a preselection of one or more execution zones, and will request each zone to assess the request for creating a new instance in that execution zone. It will likely use an intra-zone mapper that takes into account the local resource usage information coming from the intra-zone monitor, and the score from the supplier service. One or more offers is sent back from each execution zone and based on this, the domain-level mapper service selects the best offer.

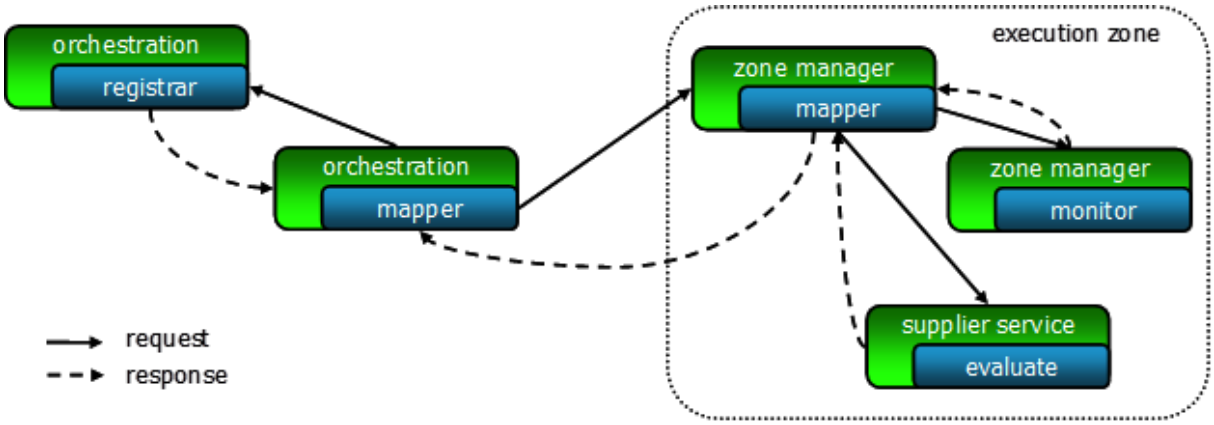


Figure 31: Service placement diagram

5.2 Services API

The following API analysis is not supposed to define the API at this early stage, but to describe what the software developers who are implementing services may expect from FUSION. These expectations are expressed partly as pseudo-code of a possible FUSION API plus sample-pseudo-code how an implementation of a service may look like.

5.2.1 API language

Proposed API language is C for the following reasons:

- Performance.
- A more comfortable C++ wrapper can be created easily.
- Bindings for other languages can be added easily (e.g. using SWIG).

Technically the API will likely contain various OO features, which can be expressed also in C by using well-known standard techniques (e.g. emulating classes via function pointers and baton parameters).

However, the following API spec use pseudo-code for more clarity. The pseudo-code does not show any error reporting, which of course is included in any real API.

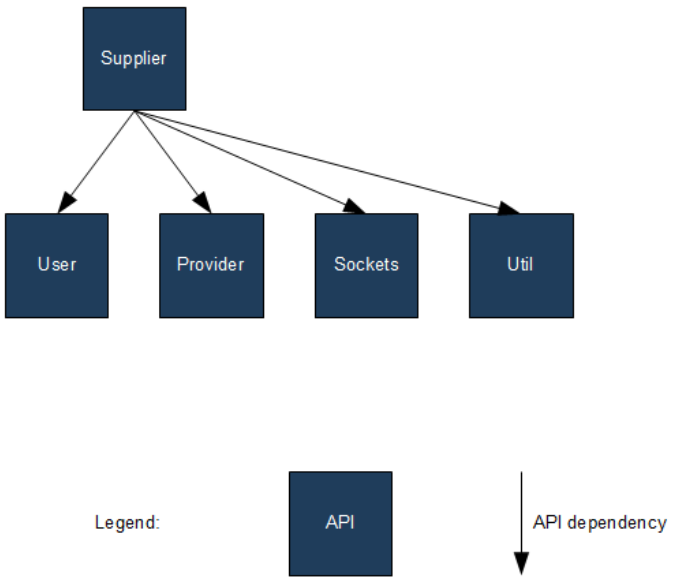


Figure 32: FUSION API Modules

5.2.2 FusionUser

Software using FUSION services needs an interface for requesting services and communicating with them. Expressed as pseudo-code, a very simple API for accessing services may look like the following, written in pseudo-code. The API may also include functions used by evaluators, which in this simple sample are contained in the same API:

```
// Implemented by FUSION:

// Send a request to a stateless service and return the answer.
byte[] fusionUserRequest(
               string service,
               byte[] params);

// Reference to a service instance:
class FusionUserServInst
{
        // Release the reference to the service instance.
        void release();

        // Location of the service instance, containing all information
        // required to connect to that service either using FUSION sockets
        // or to connect directly (e.g. via own TCP).
        // This information may be passed to other processes
        // on other computers.
        IpPort getIpPort();
        }

        // Request a new instance of a stateful service
        // according to the requirements passed in the description parameter.
        // The description is a black box for FUSION and is passed
        // to the service evaluators corresponding to the requested service,
        // see the FusionEvaluator API below.
        ServInst fusionUserInstantiate(
                       string service,
                       byte[] description);
```

If an application or a service asks for a service, then the description is custom data ignored by FUSION, and may contain service-specific parameters which are relevant for determining the optimal service instance location, for example

- Location of other running service instances (e.g. VoD servers).

- Location of all players in case of a multi-user server (e.g. a game server).
- Other services required but not instantiated yet.
- Location of running service instances required (e.g. database).

The point is that this data may be highly service dependent, but also environment dependent (e.g. locations of one or more users).

The method getIpPort can be used to establish a data connection, both by the software which requested the service, but also by other software. For example:

- The software which requested the service can establish a direct network connection via TCP
- If it is running on the same execution point, it can use the process ID to connect for example via shared memory.
- The software can pass the location to other software (e.g. other game clients) running on different execution points. These other clients can then connect to the same service (e.g. same game server).

The following diagram depicts a simple sample scenario consisting of a game server which is connected to multiple game clients, which each is connected via a video stream to a thin client:

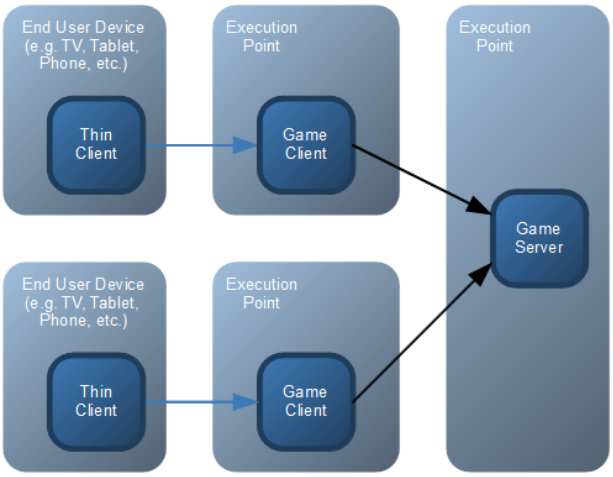


Figure 33: Sample Scenario (Game Server)

```
// Game lobby embedded into a website:
void webserver_launch_game_cgi(
                IpPort[] thinPlayerClients,
                IpPort databaseIpPort)
{
        serverDescription = object();
        serverDescription.thinPlayerClients;
        serverDescription.databaseIpPort = databaseIpPort;
        serverDescription.sessionId = createUniqueId();
        serverInst = fusionUserInstantiate(
                        "game_server", serverDescription);

        for(thinPlayerClient in thinPlayerClients)
        {
                clientDescription = object();
                clientDescription.serverIpPort = serverInst.getIpPort();
                clientDescription.sessionId
                        = serverDescription.sessionId;
                clientInst = fusionUserInstantiate(
                                "game_client", clientDescription);

                tell_thin_client_about_game_client(
                        "client at " + clientInst.getIpPort());
                wait_until_thin_client_has_connected_to_game_client();

                // Now we can release the service instance
                // since the thin client owns it now:
```

```
                clientInst.release();
        }

        // Now we can release the service instance
        // since the game client owns it now:
        serverInst.release();
}
```

In this sample some web-based game lobby code first requests a game server from fusion. In this request it passes the locatFigure 36ions of all game players, so that FUSION can determine the best location of the game server taking the locations of the players into account as described above under related services. Then after the server is created the code requests the game clients. In each request for a game client it passes the location of the server, both for allowing FUSION to determine the best location for that client, but also to allow the client to connect to the server.

5.2.3 FusionProvider

The provider interface is for implementing a service. It consists of a part implemented by FUSION and a part implemented by the service. Interface pseudo-code:

```
// Implemented by FUSION:

Object fusionProviderGetDescription();
bool fusionProviderWantsMeAlive();

// Implemented by the service, called by FUSION:

bool fusionProviderCanMigrateViaVirtualMachineMigration();
bool fusionProviderCanMigrateViaSerialization();
byte[] fusionProviderSerialize();
void fusionProviderDeserialize(
                byte[] serialization);
```

The implementation of the game server from the sample above my look like the following:

```
// Game server service implementation

void main()
{
        Object serverDescription = fusionProviderGetDescription();
        connect_to_game_database(serverDescription.databaseIpPort);

        createSharedMemory(serverDescription.sessionId);
        fusionSocketListen();

        while(game_is_active
                        || any_client_is_present
                        || fusionProviderWantsMeAlive())
        {
                if(client_connects_via_shared_memory)
                        establish_connection();
                if(incoming_connection)
                {
                        decompressed_stream = decompressor(incomding_connection);
                        establish_connection();
                }

                // Game loop:
                updateGame();
        }
}

bool fusionProviderCanMigrateViaVirtualMachineMigration()
{
        // Support virtual machine migration
        // only when not using shared memory:
        return !currentlyUsingSharedMemory();
}

bool fusionProviderCanMigrateViaSerialization()
```

```
{
        // In this case always support manual migration via serialization:
        return true;
}

byte[] fusionProviderSerialize()
{
        return serializeGameServer();
}

void fusionProviderDeserialize(byte[] serialization)
{
        deserializeGameServer(serialization);
        dataBaseReconnect();
}
```

The corresponding client may be implemented in the following way:

```
// Game client service implementation

void main()
{
        // Description is passed to us by FUSION:
        Object clientDescription = fusionProviderGetDescription();

        if(clientDescription.serverIpPort == ownServerIpPort)
                uncompressed_stream = accessSharedMemory(
                                clientDescription.sessionId);
   else
        {
                comressed_stream = fusionSocketConnectTo(
                                clientDescription.serverIpPort);
                uncompressed_stream = decompressor(compressed_stream);
        }

        while(game_is_active || fusionProviderWantsMeAlive())
        {
                if(server_has_migrated)
                        reconnectToServer(); // See below.

                // Game loop:
                updateGame();
        }
}

bool fusionProviderCanMigrateViaVirtualMachineMigration()
{
        // Support virtual machine migration
        // only when not using shared memory:
        return !currentlyUsingSharedMemory();
}

bool fusionProviderCanMigrateViaSerialization()
{
        // In this sample we always support manual migration
        // via serialization:
        return true;
}

// This function is not needed when using only virtual machine migration
// together with the FusionSockets API for all communication.
byte[] fusionProviderSerialize()
{
        return serializeGameClient();
}

// This function is not needed when using only virtual machine migration
// together with the FusionSockets API for all communication.
void fusionProviderDeserialize(byte[] serialization)
{
        deserializeGameClient(serialization);
        reconnectToServer(); // See below.
        reconnectToThinClient();
}
```

```
// This function is not needed when using
// the FusionSockets API for all communication.
void reconnectToServer()
{
        if(was_connected_via_shared_memory)
        {
                if(server_ip == own_ip)
                {
                        // Used shared memory before, using it now again:
                        uncompressed_stream = accessSharedMemory(
                                        clientDescription.sessionId);
                }
                else
                {
                        // Used shared memory before, but not anymore:
                        comressed_stream = fusionSocketConnectTo(
                                        clientDescription.serverIpPort);
                        uncompressed_stream = decompressor(compressed_stream);
                }
        }
        else
        {
                if(server_ip == own_ip)
                {
                        // Used socket before, but shared memory now:
                        uncompressed_stream = accessSharedMemory(
                                        clientDescription.sessionId);
                }
                else
                {
                        // Used socket before and socket now:
                        // No action needed, because the FusionSockets module
                        // supports transparent migration.
                }
        }
}
```

5.2.4 FusionEvaluator

The FUSION evaluator interface is for implementing evaluators which score possible locations of service instances. This API consists of a part implemented by FUSION and a part implemented by a evaluator service. Interface pseudo-code:

```
// Provided by FUSION

// Representation of scores
typedef map<string, float_or_something_else> Scores;

// Ask FUSION about the score if FUSION would instantiate
// that service at the best possible location
// according to the requirements passed in the description parameter.

Scores fusionEvaluateBest(
                string service,
                byte[] description);


// Implemented by the service evaluator, called by FUSION:

// Return all EPs this evaluator can evaluate.
// Must at minimum return own host it is currently running on:
IpPort[] fusionGetLocationsICanEvaluate();

// Score a possible location. The parameter possibleLocation
// will be one of the locations
// returned by fusionGetLocationsICanEvaluate.
Scores fusionEvaluatorEvaluate(
                string service,
                byte[] description,
                IpPort possibleLocation);
```

For example, an implementation of a fusion evaluator for a game server may do the following:

```
// Game server evaluator implementation
// for the sample:

// FUSION calls this function:
IpPort[] fusionGetLocationsICanEvaluate()
{
	return list_of_call_eps_on_this_cloud();
}

// FUSION calls this function:
Scores fusionEvaluatorEvaluate(
		string service,
		byte[] description,
		IpPort possibleLocation)
{
	Scores statisticsScore = map("default"
			-> calcLoadScoreFromStatistics(
				fusionSocketGetStatistics(possibleLocation)));

	Scores databaseScore = map("default"
			-> fusionUtilCalcNetworkDistanceBetween(
				possibleLocation, description.databaseIpPort,
				FUSION_UTIL_OPTION_SLOW_IS_FINE));

	Scores clientScores;
	for(thinPlayerClient in description.thinPlayerClients)
	{
		clientDescription = object();
		clientDescription.serverIpPort = possibleLocation;
		clientDescription.thinClientIpPort = thinPlayerClient;
		Scores clientScore = fusionUserEvaluate(
				"game_client", clientDescription);
		clientScores = max(clientScores, clientScore);
	}

	return max(clientScoresmap, statisicsScore, databaseScore);
}

Sample: Game client evaluator:

// Game client evaluator implementation

// FUSION calls this function:
IpPort[] fusionGetLocationsICanEvaluate()
{
	return list_of_call_eps_on_this_cloud();
}

// FUSION calls this function:
Scores fusionEvaluatorEvaluate(
		string service,
		byte[] description,
		IpPort possibleLocation)
{
	if(!computerHasGpuShaderModel3x0(possibleLocation))
		return 0.0; // Worst possible score

	map<string, float_or_something_else> statisticsScore
			= calcLoadScoreFromStatistics(
				fusionSocketGetStatistics(possibleLocation));

	map<string, float_or_something_else>serverScore
			= fusionUtilCalcNetworkDistanceBetween(
				possibleLocation, description.serverIpPort,
				FUSION_UTIL_OPTION_LOSSLESS_LOW_LATENCY_AND_JITTER);

	map<string, float_or_something_else>float thinClientScore
			= fusionUtilCalcNetworkDistanceBetween(
				possibleLocation, description.thinClientIpPort,
				FUSION_UTIL_OPTION_LOSSY_LOW_LATENCY_AND_JITTER);

	return min(statisticsScore, serverScore, thinClientScore);
}
```

5.2.5 FusionSockets

Interface pseudo-code:

```
// Implemented by FUSION:

... fusionSocketListen(...);
... fusionSocketConnectTo(
              ipPort target, ...);
... fusionSocketGetStatistics(...);
```

The purpose of the FUSION socket API is to provide basically the same functionality as classical sockets, but

- supporting transparent migration, and
- automatically collecting statistics for FUSION.

5.2.6 FusionUtil

Communication between services may be provided by FUSION in a separate API. A FUSION service may choose to use own proprietary communication means or may use the FUSION communication functions, which can offer additional features like transparent service instance migration. Interface pseudo-code:

```
enum
{
        FUSION_UTIL_OPTION_SLOW_IS_FINE,
        FUSION_UTIL_OPTION_LOSSLESS_LOW_LATENCY_AND_JITTER,
        FUSION_UTIL_OPTION_LOSSY_LOW_LATENCY_AND_JITTER
}

Scores fusionUtilCalcNetworkDistanceBetween(
               IpPort ipPortA,
               IpPort ipPortB,
               int options);
```

The utility module may contain function for standard scoring. This can be optionally used by a fusion evaluator implementation. These utility functions provide a standard scoring for placing a list of services each at a given execution point, where the services need a set of pair-wise connections provided by the "connections" array. Taking into account compatibility aspects, FUSION may provide more and more utility functions.

6. BUSINESS MODELS

This section describes the possible business models that can use FUSION as their substrate. The following set of figures show different options for the ownership of orchestration domains, routing planes and the underlying IP infrastructure.

Figure 34 shows an ISP-centric business model where ISPs deploy service routers within their domain to route to services running in their own execution zones. Each colour represents an orchestration-routing-ISP domain. There is a common ownership across all three layers (IP, service routing and execution) and, as such, the ISPs have privileged access to information across layers: for example the service routers can access internal information about the IP network to obtain network topology information and to monitor statistics that the ISP might not wish to share with third parties. Furthermore the ISP has detailed knowledge about the location and capabilities of its execution zones. The figure shows inter-domain connectivity between domains at the service routing level, however this is optional as an ISP may only make its services accessible to its own customers. One advantage of this scenario is that ISPs can easily configure their users' equipment to access the closest service router for making service queries/invocations – through DHCP, for example. The same is true for the execution zones' configuration for knowing which service router they should make service announcements and updates towards.

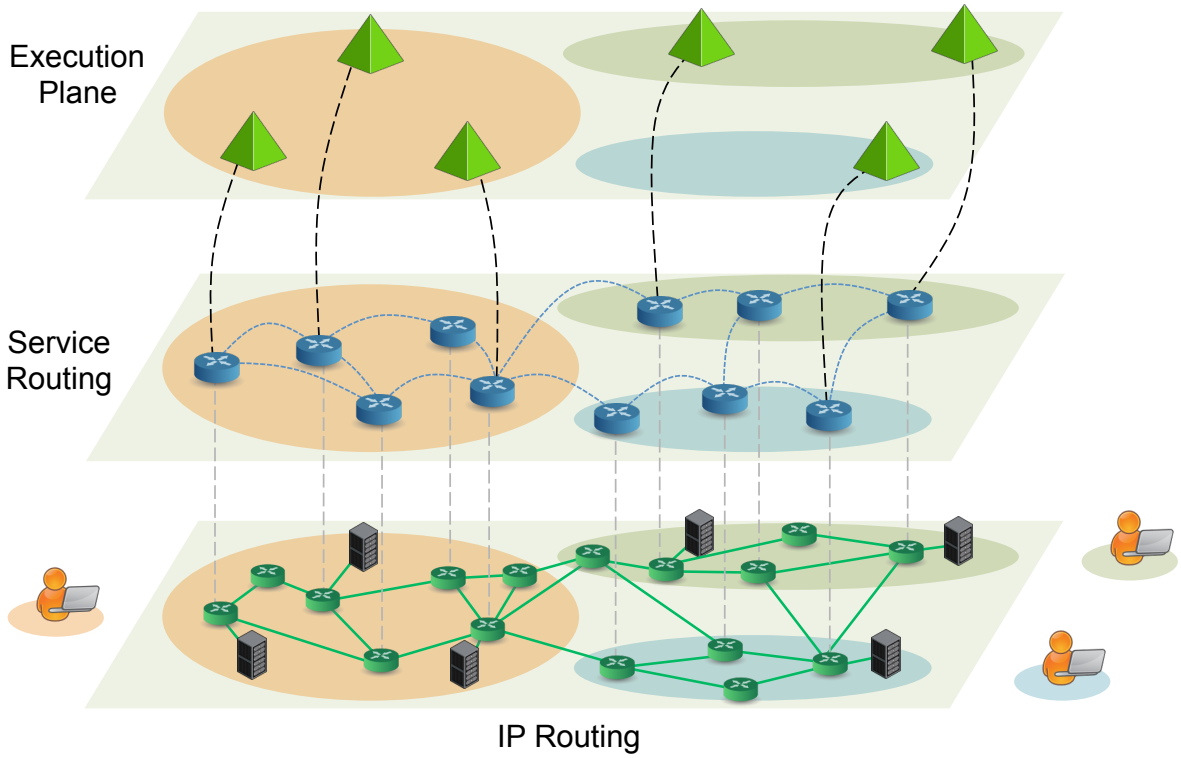


Figure 34: ISP-centric business model

Figure 35 shows a business model scenario where orchestration domains are separate from the service routing plane which, in this model, is operated by the ISPs. The execution zones that belong to – or are contracted to – each orchestration domain are represented by the three different coloured areas in the execution plane. Service announcements are injected into the service routing plane by the execution zones to the closest service router in the ISP operating the network to which the execution zone is attached. Routing information is exchanged within and between service routing domains to form a common, global routing plane in a similar fashion to the way IP operates today with IGP and EGP routing protocols. Users, in this model, are able to access services deployed and operated by any orchestration domain they are contracted to use, irrespective of the ISP they belong

to. As in the previous scenario the users' equipment can easily be configured to access their local service router for invocations/queries as it is operated by their own ISP.

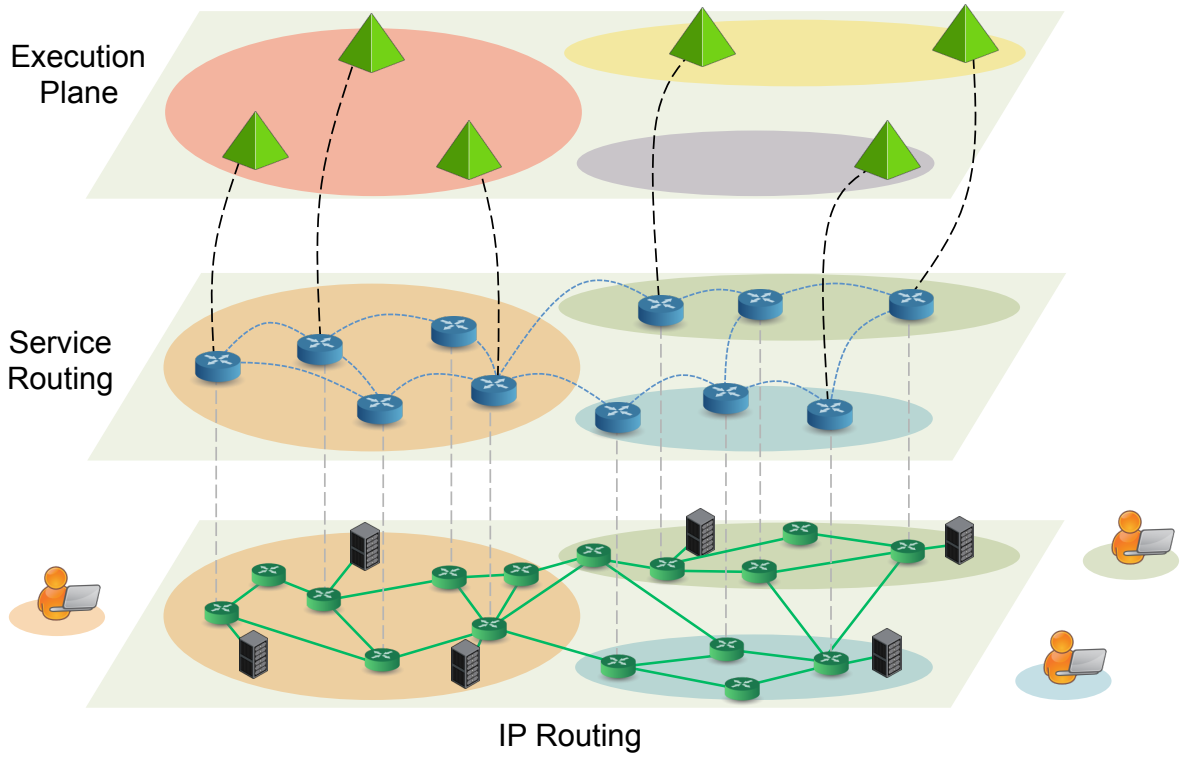


Figure 35: Third-party orchestration domain business model

Figure 36 shows a model of operation where the service routers are not operated by the ISPs and in this sense it is an over-the-top scenario. Orchestration domains own or contract execution zones and additionally deploy service routers, possibly within their execution zones. The orchestration domains interconnect their service routers to form a common, global service routing plane in a similar fashion to the way the ISPs interconnected their service routers in the previous example. Because this is an OTT mode of operation users need to connect to their closest (or at least a nearby) service router which is no longer operated by their ISP, hence it is not as straightforward as in the previous examples when the configuration could be done automatically by their ISP. It should be noted that in this figure some of the execution zones are shown with multiple colours. This is to represent the case where the same data centre is used by more than one orchestration domain.

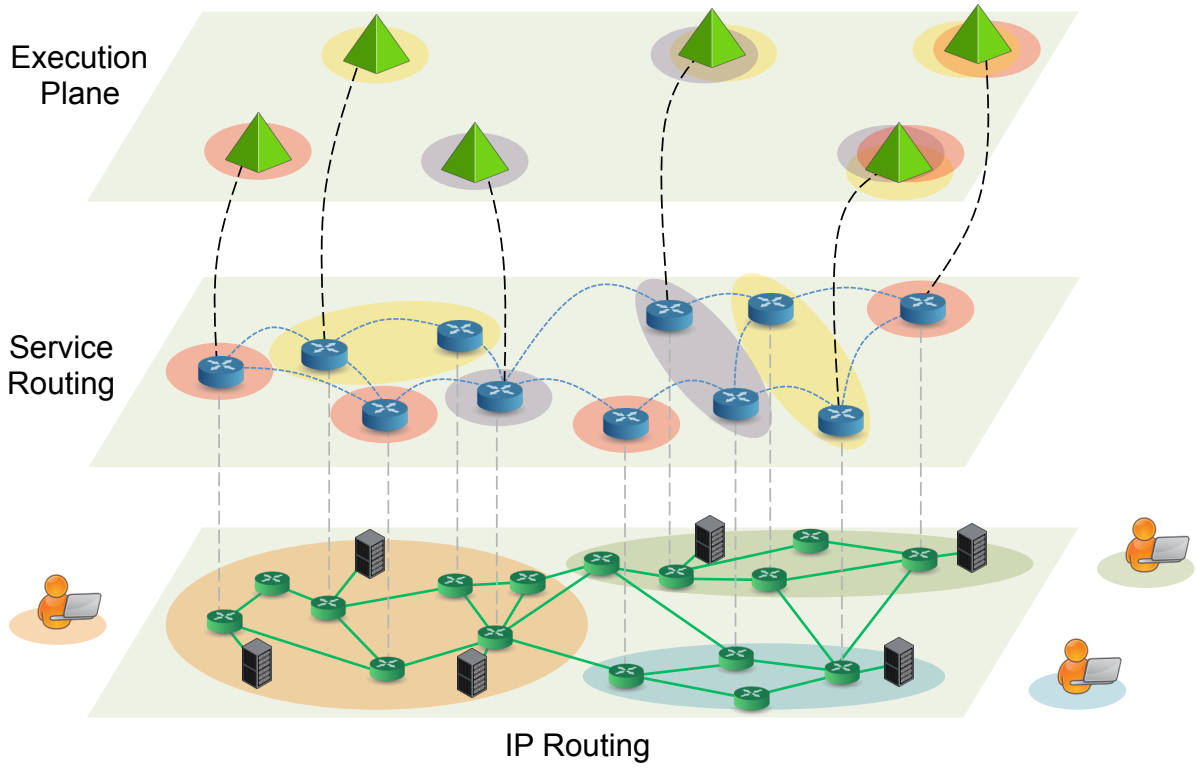


Figure 36: OTT business model with common service routing plane

Figure 37 shows a scenario where orchestration domains do not interconnect their service routers and instead operate disconnected service routing planes for each orchestration domain. This scenario is roughly equivalent to today's CDNs.

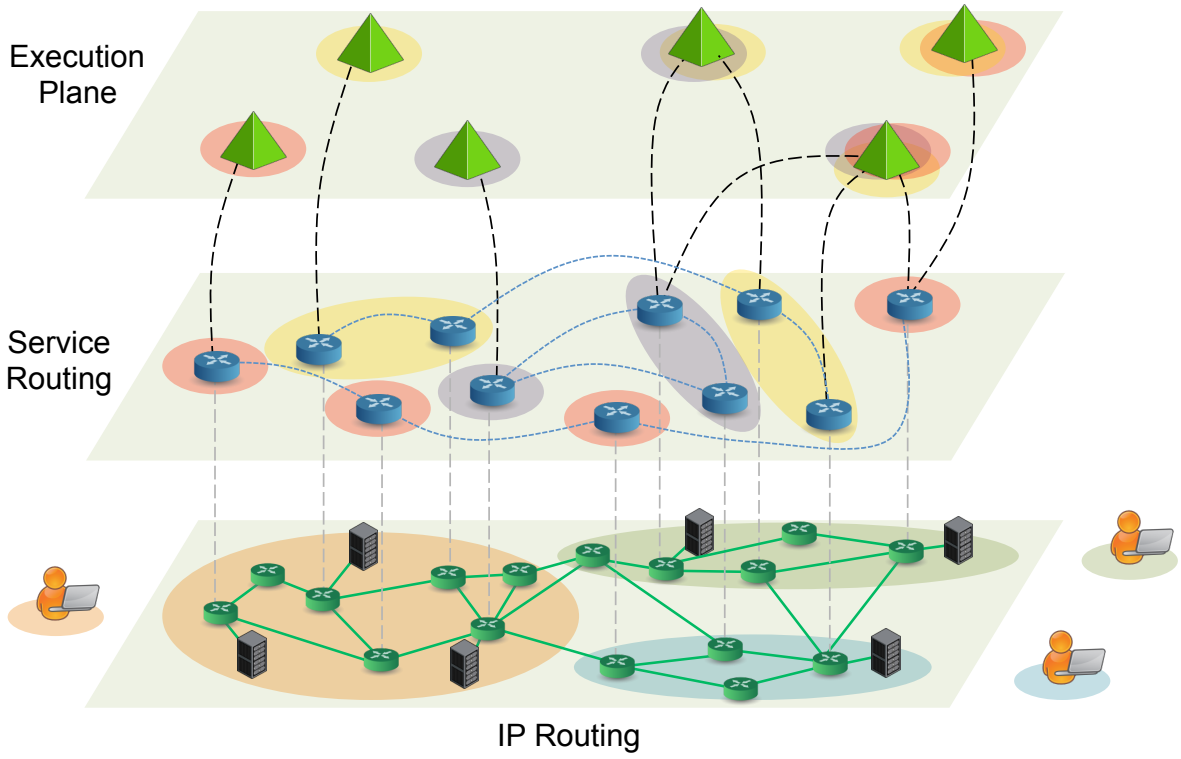


Figure 37: OTT business model with disjoint service routing planes per orchestration domain

Figure 38 shows the case where all three planes (IP, service routing and execution) are operated by different entities. This option is included for completeness but is seems less likely than the other

models above. One of the main issues concerns who would operate the service routing plane as this would need to be a separate business entity to both the underlying ISPs and the orchestrators operating the services.

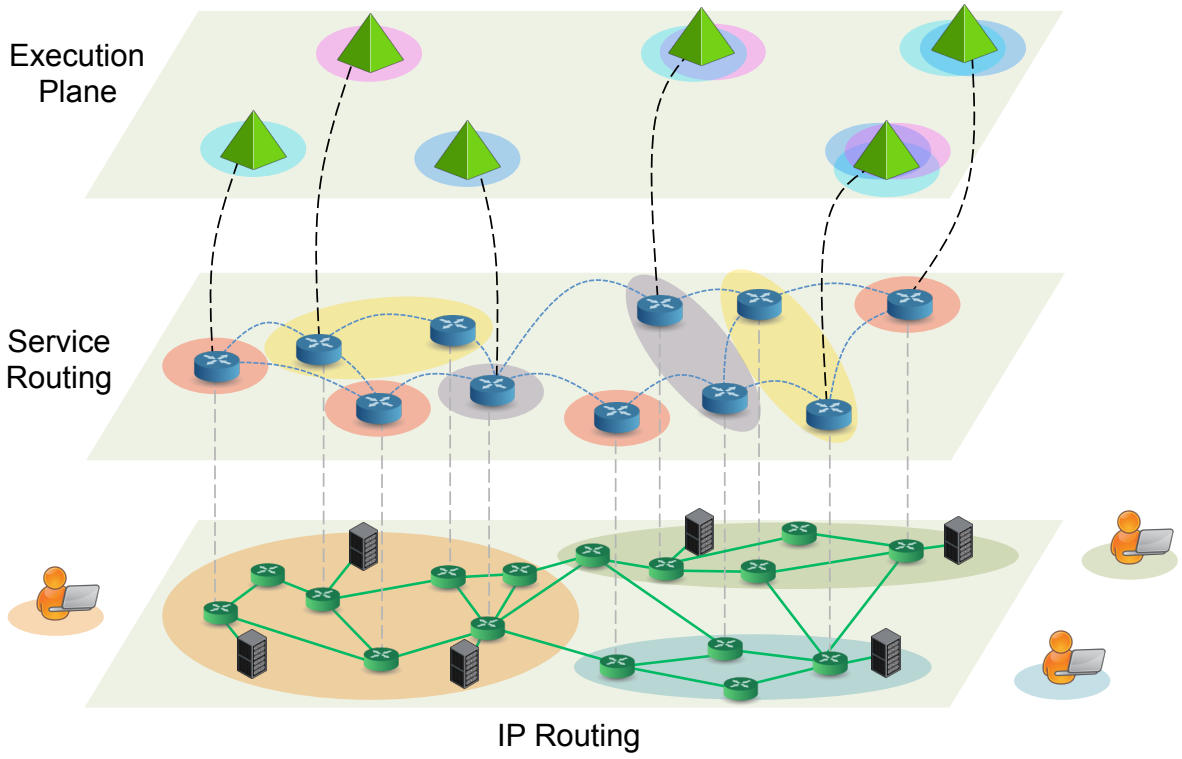


Figure 38: Separate routing and orchestration domains

In addition to the five scenarios above various hybrid models are also possibilities, for example both ISPs and orchestration domains could operate service routers in Figure 36. Alternatively ISPs operating in the ISP-centric model in Figure 34 may also make use of remote execution zones by contracting resources from third party data-centres.

Following the overview of business model scenarios as described above the following subsections define the stakeholders before describing a set of business models showing the flow of money between them.

6.1 Stakeholders

The following is a list of the essential possible stakeholders in the FUSION ecosystem. We limit ourselves to the essential ones. Others, like advertising agencies, etc. are not listed but they are assumed as a possibility in some business models.

6.1.1 Users

Users are the final consumers of FUSION enabled services. They can be personal or enterprise based. They can be accessing through wired access or through a wireless/mobile carrier. Providing services to users is the final goal of all FUSION interactions.

6.1.2 Application Developers

Application developers are the producers of any given service. They can be a content producer, a computation intensive provider among many other examples. It is assumed that the developer wants to create a revenue stream to monetise the service either by direct payment (selling or renting) or through adverts. Free services are also a possibility providing all the stakeholders participate.

6.1.3 IP Connectivity Providers

This refers to network providers commonly referred as ISPs (Internet Service Providers). They can be wired or wireless. They can be edge ISPs (eyeball ISPs) or transit ISPS, although we expect the latter to be less likely to participate in the FUSION framework.

6.1.4 Computation provider

Computation providers, which run FUSION execution zones, own and operate computation equipment where services' instances can run. These can include traditional cloud providers like Amazon EC2 or Rackspace, open clouds provided by the ISPs or smaller third party operators.

6.1.5 Orchestration Provider

This stakeholder deploys the service entities; this could be a single entity for the entire Internet (similar to Akamai today) or any brokerage service that interacts with smaller services with limited footprint. Potentially they can have an interface with the user through something similar to today's app stores.

6.1.6 Service Routing Provider

This stakeholder is created by FUSION; it provides service routing but no service hosting. This means that this entity provides service routing without providing execution capabilities to the orchestrator. It is expected that the first candidates to run such entities will be the ISPs. It is crucial that they will cooperate honestly with other service routing providers.

6.2 Examples of Business Models

In this section we enumerate the possible business models where FUSION can operate.

6.2.1 User Pays App Developer

Here the user pays for the service directly to the application developer. This payment can be done by money exchanging hands or through an advertising model where the user “pays” by viewing adverts. The app developer pays the orchestrator which then pays the several execution zones to instantiate the services.

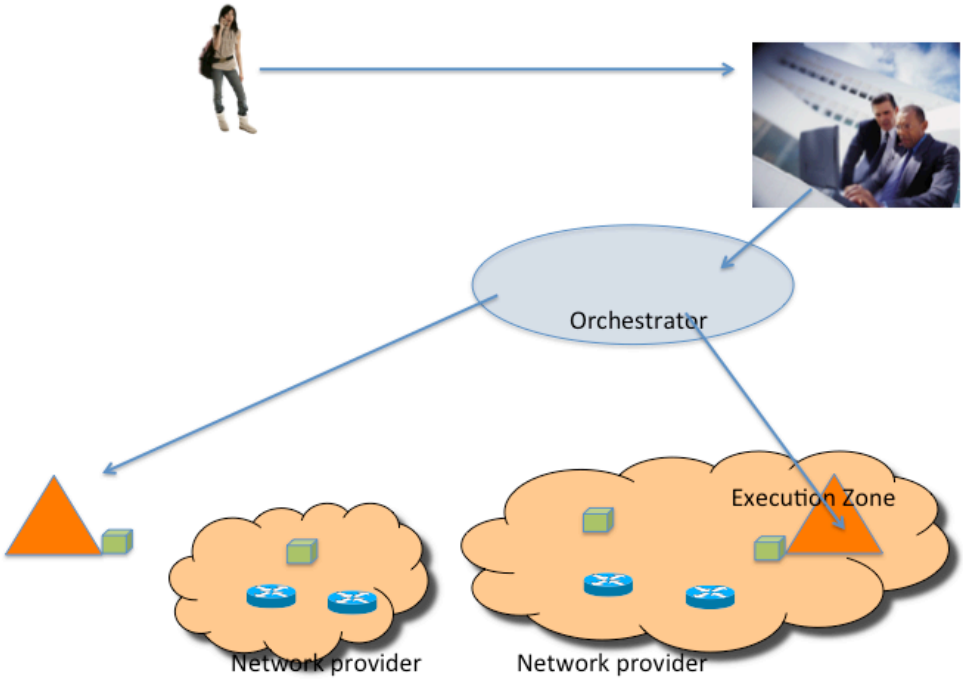


Figure 39 - User Pays App Developer

6.2.2 User Pays ISP

In this model the user pays its ISP for the service. In this way, the user only needs to trust its provider with payment information, which he/she needed anyway. The ISP shares some of this revenue with the application provider which then pays the orchestrator to instantiate the services themselves.

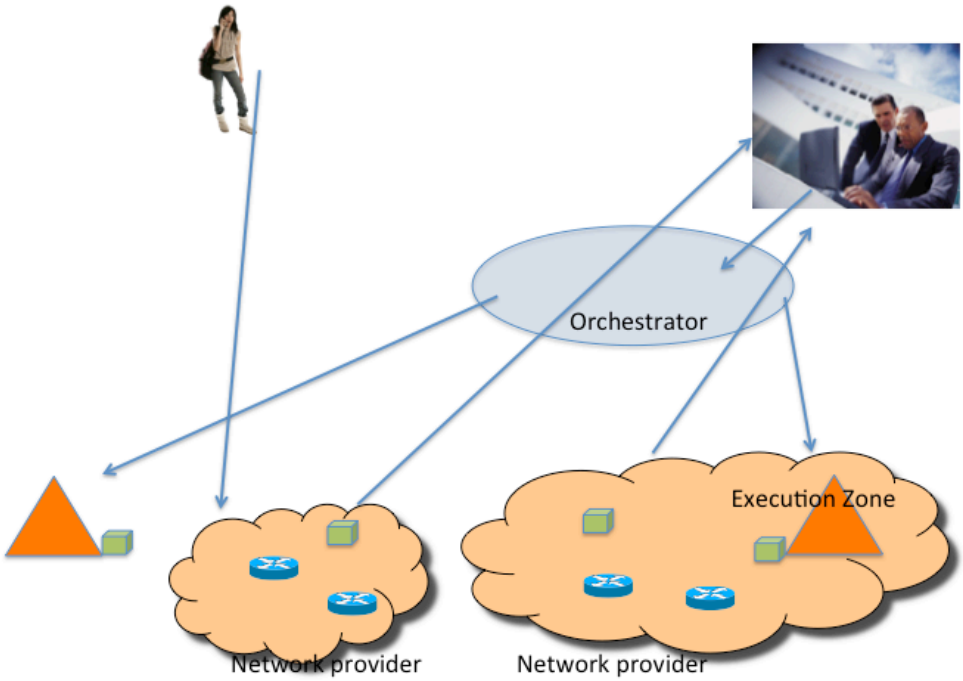


Figure 40 - User Pays ISP

6.2.3 Orchestrator as a Reseller (User pays ISP)

In this scenario the user pays its local ISP which needs to pay to another service provider in case, for example, if the user moves. In this case the ISP's orchestrator acts as a reseller of the service and they also pay the application developer.

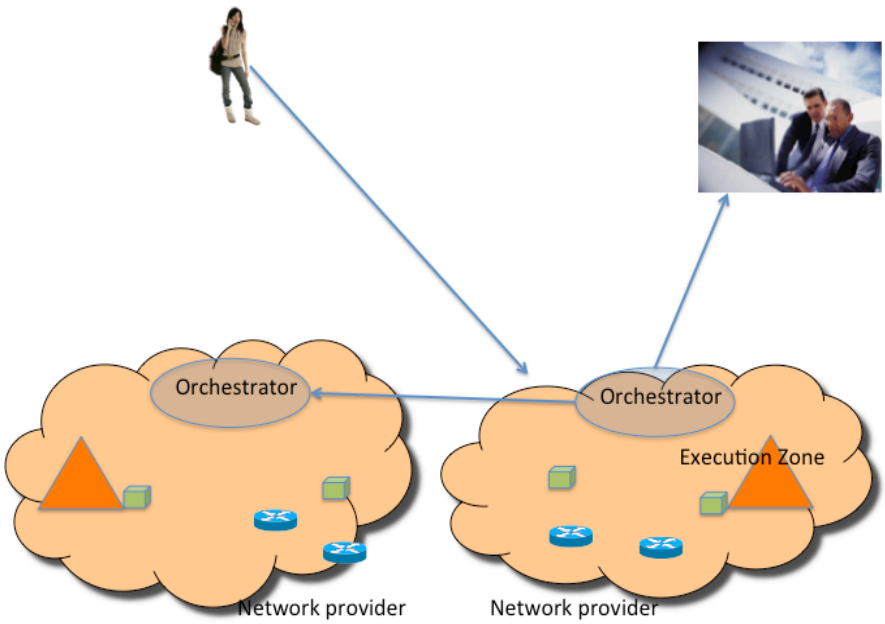


Figure 41 – Orchestrator as a Reseller (User pays ISP)

6.2.4 Orchestrator as a Reseller (User pays App Developer)

This scenario is similar to the previous one but the user pays the application developer instead. This pays the different orchestrators which then resell the services to each other.

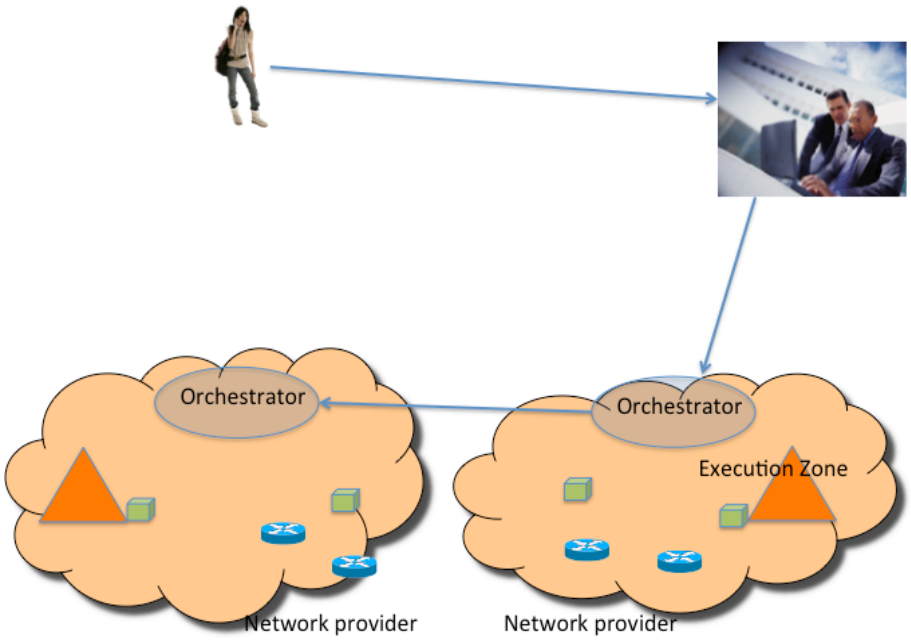


Figure 42 - Orchestrator as a Reseller (User pays App Developer)

6.2.5 User pays Orchestrator

In this scenario the orchestrator acts as a front-end to the user and charges him/her directly. This is similar to today's app stores (Apple, Amazon, Google Play). These orchestrators establish relations with the app developers as they do today and send payments after taking a cut. The novelty here is that they also establish relationships with execution zones and are directly involved in the instantiation of new services.

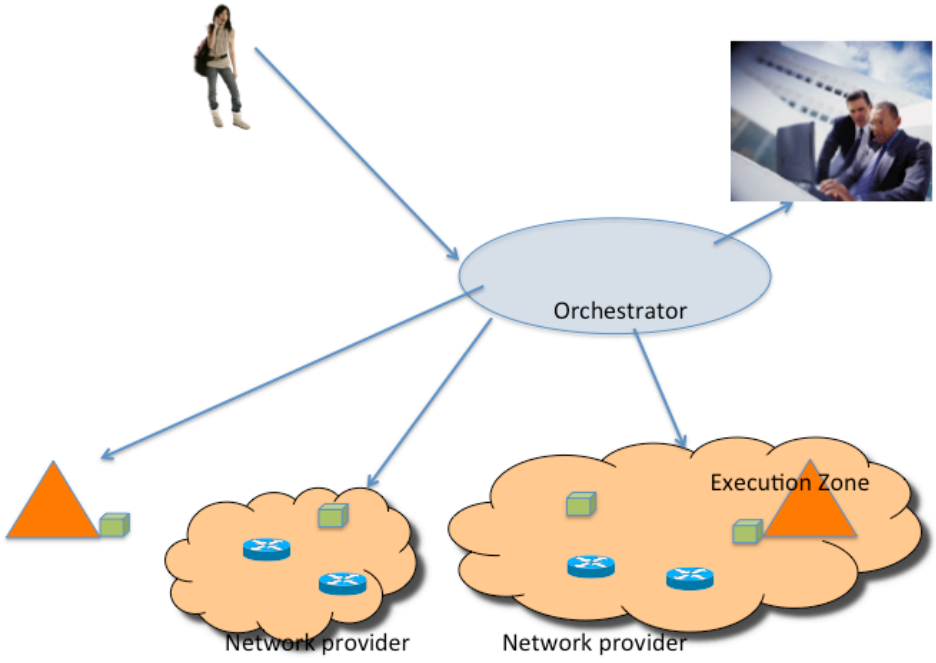


Figure 43 - User pays Orchestrator

6.3 Adopted business models

The project will initially concentrate on two business model scenarios: the ISP-centric business model presented in Figure 34 and with the user paying the ISP as discussed in section 6.2.2, and an OTT business model with common service routing plane as shown in Figure 36 with the user paying the app developer as discussed in section 6.2.1. We believe that these two scenarios will cover the most frequent service usage cases. The other scenarios (orchestrator as reseller and user pays orchestrator), although possible, are not foreseen to be typical. Finally, despite the business model differences they do not change significantly the operational workflow of the FUSION interactions. A deeper consideration of these and alternative business models from the perspectives of exploitation of the project results will be undertaken as the technical work progresses.

7. SECURITY ASPECTS OF SERVICE ORIENTED NETWORKING

In this section we first look at current security issues in clouds and then enumerate the extra specific issues concerning the FUSION framework.

7.1.1 Security approaches in related work

7.1.1.1 Cloud-related security

Cloud services bring a set of new security problems due to novel interactions between Cloud Providers (CP) and Cloud Clients (CC). The following security risks are stated in the European Union Agency for Network and Information Security's report: "Cloud Computing Benefits, risks and recommendations for information security" [ENISA12].

1. Resource exhaustion (under or over provisioning): As Cloud services are on-demand services, there is the possibility that the CP won't be able to meet an increased demand in a certain shared resource, or to maintain a given service level.
2. Isolation Failure: In shared environments, errors or attacks can lead to situations where one tenant has access to another tenant's resources or data. In the case of attacks, an attacker gets access to the resources or data of a specific customer, or even of all customers of the Cloud service.
3. Cloud provider malicious insider - abuse of high privilege roles. Malicious insiders at the CP can cause various kinds of damage to a CC's assets.
4. Management interface compromise (manipulation, availability of infrastructure). The customer management interfaces of public cloud providers are Internet accessible and mediate access to larger sets of resources (than traditional hosting providers) and therefore pose an increased risk especially when combined with remote access and web browser vulnerabilities.
5. Intercepting data in transit. Whenever data is transferred between different computers or sites, there is the possibility that the transfer can be intercepted. This is especially relevant in shared environments and when data is transferred between sites (e.g. between CC and CP).
6. Insecure or ineffective deletion of data: Deleting data from Cloud storage does not in fact mean that the data is removed from the storage or eventual backup media. If disk storage is not encrypted, the data could be accessed at later time by another customer of a Cloud provider.
7. Distributed denial of service (DDoS): Distributed Denial of Service attacks aim at overloading a resource (network or service interface) by flooding it with requests from many sources distributed across a wide geographical or topological area, so that the legitimate users are unable to use the resource as intended.
8. Economic denial of service (EDoS). As a consequence of attacks, poor budget planning, or misconfigurations, the cost of a Cloud service can strain the financial resources of a CC to an extent that the service is no longer affordable.
9. Compromise of Service Engine. The service engine is a fundamental part of a Cloud service. A compromise of the service engine will give an attacker access to the data of all customers, resulting in a potential complete loss of data or denial of service.
10. Loss of Cryptographic Keys. The loss or compromise of cryptographic keys used for encryption, authentication or digital signatures can lead to data loss, denial of services, or financial damages.
11. Non Cloud-Specific Network-Related Technical Failures or Attacks. Cloud services can be affected by a number of network-related technical failures that can also occur on classic IT settings. Examples include the loss of Internet connectivity due to failures at the CC's site or the CC's Internet service provider, temporarily reduced network bandwidth on the path between CC and CP, disruptions in the global Internet routing infrastructure leading to the loss of the network path between CC and CP, and failures of the CP's Internet connectivity.
12. Loss of Backups. The backups a CP makes of its customers' data can get lost, damaged, or the physical media on which the backup is stored can get stolen.

13. Natural disasters Natural disasters like flooding, earthquakes, tsunamis can affect the infrastructure of a CP. This way, a CC might be affected by natural disasters occurring far away from its own location.

7.1.2 FUSION-specific security threat models

7.1.2.1 Confidentiality of the µVM

Short Description	In an environment where code runs in shared virtual machines, one needs to assure no data leaks to other instances (not necessarily FUSION-specific as it related to the lightweight container model).	
Risk Rating	Probability: Low	Impact: High
Comparison with Traditional Cloud provision	FUSION's lightweight containers are lighter than traditional virtual machines since, in order to minimise instantiation time, they allow server instance processes to share the same VM or host. This may reduce the isolation between service instances in particular circumstances.	

7.1.2.2 Service Pollution/False Announcements

Short Description	One needs to be sure that an announcement of a replica corresponds to the right service. This applies both to the instances themselves and to the service routers.	
Risk Rating	Probability: High	Impact: Medium
Comparison with Traditional Cloud provision	Since anycast is at the centre of the FUSION architecture and it's a novel feature, this problem does not really exist in today's clouds infrastructures. The only scenario where this could happen today would be in layer3 anycast but because this is done by single companies the risk is mitigated.	

7.1.2.3 Service Authentication

Short Description	Service Authentication: In some services only authenticated users should be able to send a message to be resolved/replied to (there may be even greater security concerns re denial of service, etc. in the case of service instantiation on demand)	
Risk Rating	Probability: Medium	Impact: Low
Comparison with Traditional Cloud provision	FUSION will allow open services to require authentication from users. This will ideally even stop users reaching the service if they are not authorised. This is a new feature that solves a well-known Internet problem.	

7.1.2.4 Authentication of the service instantiation

Short Description	Authentication of the service instantiation: At the point of service instantiation one needs to be sure that the publisher is authorised	
Risk Rating	Probability: Low	Impact: Very High
Comparison with Traditional Cloud provision	The orchestration plane of FUSION is a novel feature since it allows different orchestrators to instantiate services in several execution zones. Third party computation providers can house executions zones from many orchestration domains within a single physical data centre. This opens new vulnerabilities in authenticating who can instantiate in which zones/data centres.	

7.1.2.5 Performance isolation

Short Description	When a service gets instantiated one needs to be sure that services cannot be degraded on purpose by other services	
Risk Rating	Probability: Low	Impact: Medium
Comparison with Traditional Cloud provision	As stated in 7.1.2.1, FUSION's lightweight containers are lighter than traditional virtual machines since, in order to minimise instantiation time, they allow server instance processes to share the same VM or host. This can reduce the isolation between processes. If two instances are from different providers this increases the risk of a misbehaving process affecting other process's performance.	

7.1.2.6 Traceability

Short Description	Service usage history needs to be logged for audit/trail/forensics purposes	
Risk Rating	Probability: Low	Impact: Medium
Comparison with Traditional Cloud provision	In FUSION services run across many execution zones, running in several providers. To obtain a trace of service usage for forensics purposes will be much more difficult and will potentially have to involve the service routing itself.	

7.1.2.7 Code Integrity

Short Description	One needs to be sure that code hasn't be tampered with before instantiation and during service lifetime	
Risk Rating	Probability: Low	Impact: High
Comparison with Traditional Cloud provision	Due to the extra complexity of having services running in several executions zones, FUSION will potentially increase the risk of code being tampered with when in transit between service providers, orchestrators and execution zones.	

7.1.2.8 Integrity of the monitoring system

Short Description	One needs to be sure that network measurements do not get tampered with.	
Risk Rating	Probability: Medium	Impact: Low
Comparison with Traditional Cloud provision	The open nature of the service routing, that relies heavily on measurements to assess, brings a new class of attacks where measurements are faked in order to subvert the routing system. This will be particularly concerning if we allow services instances to contribute measurements to the routing system.	

8. RELATED WORK

In this section we position other systems and architectures with respect to FUSION's approach as described in this deliverable. We focus on the comparison and positioning of the overall architectural issues in this section: a more detailed comparison of the distributed service management dimension of the considered architectures is contained in chapter 3 of deliverable D3.1 [D3.1]; and an analysis of the service-aware networking dimension from the perspectives of naming, addressing, service discovery, service binding and routing is documented in section 3.2 of deliverable D4.1 [D4.1].

8.1 Grid and Cloud Computing

A reference architecture for grids has been standardised in [OSGA06]. An in-depth overview of grid technologies based on the OSGA architecture has been covered thoroughly in [FOST04]. According to the latter, at the highest level, grids can be viewed as an extension of Web Services framework to address issues related to service semantics, e.g. service creation, lifetime / state management, and fault management. According to the authors, OGSI (Open Grid Services Infrastructure) addresses these issues and a Web service that complies with OGSI is referred to as a Grid service.

As grids and clouds share a lot of architectural concepts, in the following we provide their short overview, mostly based on a comparative description [FOST08]. The latter is rather high-level and is organised around such aspects of grids and clouds as resource management, programming model, and application model.

8.1.1 Resource Management

Resource management in distributed computing can be viewed from several angles, and in the following we address three of them, namely the compute model, data and monitoring. We note however that the inclusion of data imposes also storage to be considered as a managed resource. Moreover, other resources as network, software and services themselves can be considered as resources in future generation grids and clouds [SCHW05].

8.1.1.1 Compute Model

Most Grids use a batch-scheduled compute model which imposes expensive scheduling decisions, data staging in and out, and potentially long queue times. For this reason the support for interactive applications is not very common in Grids, although there have been efforts to enable lower latencies to resources based on multi-level scheduling so that applications with many short-running tasks can execute efficiently on Grids [RAIC07]. In fact, scheduling in distributed systems is a very exploited area; one possible direction is to plug the activity migration into the scheduling framework [SAIR08]. Cloud Computing compute model differs from the above as resources in the Cloud are shared by all users at the same time (as opposed to dedicated resources governed by a queuing system). This is due to the fact that the basic model adopted for Cloud-based applications assumes transaction oriented (small tasks in the order of milliseconds to seconds) and interactive applications. In many cases latency sensitive applications can run smoothly on Clouds, but guaranteeing the required level of QoS to the end users in general will be one of the major challenges for Clouds. In fact, scheduling in distributed systems is a very exploited area (see e.g. [SAIR08] and references therein); one possible direction of research is to plug the activity (service) migration into the scheduling framework [SAIR08].

8.1.1.2 Data

A viable option for computing in the future Internet is that it will be organised around Data, Cloud Computing, and Client Computing. Data management (e.g., fragmentation replication, caching) will thus play an important role. But addressing the storage problems in separation from computation implies a lot of data movement which will soon lead to scalability issues, resulting also in low

utilisation of the hardware resources. Thus, achieving good scalability for Clouds, Grids, and their applications requires that data must be highly distributed, and computations must be forwarded towards best place to execute in order to minimise the communication costs. For FUSION, it is important to decide whether such data-aware schedulers and dispersing data close to processors is a part of FUSION architecture or if coping with these problems will be delegated to specialised black-boxes. In either case, appropriate representation of storage will need to be proposed in order to achieve storage awareness of FUSION orchestration plane.

8.1.1.3 Monitoring

Abstract/unified resources offered in Clouds usually are provided by means of virtualisation, so tracking the issues and fine-control over the resources might be more difficult than in non-virtualised platforms. To cope with these problems monitoring in Clouds thus requires a fine balance of business application monitoring, enterprise server management, virtual machine monitoring, and hardware maintenance. It is worth mentioning that existing Cloud platforms often adopt existing frameworks for monitoring services. For example Amazon's Web Services CloudStatus [CLOUST] is implemented using an open source monitoring and management infrastructure called Hyperic HQ [HYPEHQ]. Akogrimo Monitoring System [AKOG05] is another example which is based on OGSA and WSRF specifications, and uses Globus Toolkit 4 platform for the implementation of the monitoring services. It is expected that for FUSION it will be sufficient to take a similar approach to the monitoring framework.

Since the start of the FUSION project, the following recent related work (since 2010) has been studied. Firstly, there are a number of papers that provide an overview and comparison of monitoring in the cloud [M1], including specific surveys on platforms, techniques, and tools for monitoring cloud infrastructures, services and applications as well as open issues and future directions [M2].

Many papers highlight some specific aspect of monitoring. For these, we have selected a few of them based on the aspect they highlight. In [M3], the PEP monitoring model is presented, focusing on lower level monitoring and specifically study the event propagation aspect. In [M4], the authors study the performance analysis of monitoring results aspect, and propose a performance analysis framework to enable smart system resource monitoring in the cloud. Dhingra et al. [M5] study the impact of how to present monitoring information to the customer and contains a proposal for a distributed resource monitoring framework.

There also exist already many papers studying VM monitoring. For example, in [M6], they focus on performance monitoring in KVM-based cloud environments, providing a low level detailed explanation on capabilities, challenges and future direction concerning guest PMU development, more specifically for perf based virtualisation performance monitoring. Secondly, in [M7], PAPI-V is proposed, describing how to extend the PAPI hardware counter library for virtual environments. In [M10], the authors present a monitoring framework that is capable of monitoring across different virtualisation technologies, in what they call hybrid cloud monitoring.

Another aspect is about detecting misbehaviour in a cloud infrastructure. In [M8], the authors discuss possible security threats through the use of several service scenarios and proposes a methodology and architecture for detecting abnormal behaviour through the monitoring of both host and network data.

Service and application monitoring is studied in [M9], where they present M4CLOUD as a novel approach for classifying and monitoring application level metrics in a resource-shared cloud environment. Network monitoring in cloud environments is studied in [M11], where they describe a method to do low-cost in-line link measurements to reveal the true traffic-perceived service quality in short timescales. Finally, in [M12], a scalable architecture is presented to enable real-time monitoring in large information systems.

Note that next to the aspects of monitoring we discussed above, there are still other aspects important to monitoring in general. We did not take these into account here, as these fall outside the scope of FUSION.

8.1.2 Programming Model

Programming model in grid environments is similar to that in traditional parallel and distributed environments. It is however more complex as it has to deal with such issues as multiple administrative domains, large resource heterogeneity, stability and performance, exception handling in highly dynamic environments, etc. For example, applications in grids usually are loosely coupled (i.e., the output of one may be passed as input to one or more others) and the programmer's focus is on management issues relating to the large numbers of datasets and tasks rather than on the optimisation of inter-process communication. For this reason workflow systems [ZHRA08] are more appropriate to specify and control the execution of applications in grids. This includes controlling the pattern of interactions between individual service components within an application (e.g., to govern the flow of control/data through components, monitor SLA assurance and reallocate the execution, handle exceptions, etc.). In clouds, it is also conceivable to introduce workflow capability for advanced applications, although according to [FOST04] "Mesh-up's and scripting (Java Script, PHP, Python etc.) have been taking the place of a workflow system in the Cloud world, since there is no easy way to integrate services and applications from various providers". Workflow managers are often designed to allow for optimisation of various system parameters, and we note there is a vast body of work on this topic (see, e.g., [YUBU05]). Current trend is to enrich the set of system-oriented metrics (e.g., overall resource utilisation) with metrics directly related to QoS such as time to completion or deadline [GOUG07], [PHOS07].

In the above context, it is important to decide on a general approach FUSION should take regarding workflow management and scheduling. One important question that arises is whether and to what extent FUSION can build on known solutions for workflow management while developing the concept of service manifest. To this end FUSION first needs to explore how expressive the service graph should be (i.e., what dependencies within the application should be modelled). In particular, care should be taken whether workflow systems based on traditional languages like BPEL are sufficient for FUSION in supporting dynamic service composition as they have been reported as being deficient in this respect [DUYA12]. Moreover, it has to be analysed whether FUSION should incorporate extended models that assume combined service composition with routing [HUSH10] (appropriate services executed in sequence along the data path).

8.1.3 Application Model

According to [FOST04], grids are capable of supporting a wide spectrum of applications, ranging from high performance computing (HPC) to high throughput computing (HTC). The main field of applications for HPC are tightly coupled parallel jobs that require low-latency interconnects and are typically executed on a single machine rather than across a wide area network; these applications often use message passing for inter-process communication. But grids have also been successful in the execution of more loosely coupled applications, often managed through sophisticated workflow systems. Such applications can comprise many tasks that can be deployed using different resources located in multiple administrative domains. In principle, there are not "paradigmatic" constraints as to many features of tasks run in both grids, e.g. task size, uni/multiprocessor, compute/data- and clouds intensive, static/dynamic, homogeneous/heterogeneous, loosely/tightly coupled, small/large aggregate number of tasks, quantity of computing, and volumes of data.

FUSION targets fine-grained and distributed interactive multimedia applications that can maintain sustainable data exchange flows between components. For this reason the main differences with respect to grids/clouds may result from the fact that networking aspects are targeted explicitly by FUSION, and also because the execution environment for certain FUSION applications (e.g., games)

may be distributed by definition (and not only optional). This in turn can have impact on detailed resource management mechanism that FUSION will need.

8.2 Cloud platforms - OpenStack case

There are multiple cloud platforms available but related state-of-the-art analysis is not the central point in this report. We assume that data centres, and also cloud computing platforms, will become a sustainable component of future Internet environment; a component that will be governed by its own rules. Therefore, in this report we feel more desirable to suggest only a possible general position of FUSION with respect to data-centre technologies or cloud computing platforms. In the following we arbitrarily use OpenStack as a reference solution.

OpenStack is a cloud management system that defines a set of interfaces that provide three main infrastructure (IaaS) services [OpenStack]:

- OpenStack compute aka Nova, is a python-based software used to orchestrate cloud and manage virtual machines and networks. Nova allows us to create and manage virtual servers using machine images. To this end OpenStack supports many popular hypervisors, e.g. KVM, QEMU, Xen, VMware.
- OpenStack Object Storage aka Swift is roughly similar to Amazon S3. Swift allows us to store objects in massively scalable infrastructure with in-built redundancy and fail-over. It can be used to store static data (like images and videos), make back-ups, archive data, and so on. Swift will write copies of data to multiple redundant servers which are logically grouped into Zones. Zones are isolated from each other to safeguard from failures. We can configure Swift and decide the number of Zones and replicas we need to have in the system. Object is the basic storage entity in Swift. An object can be anything like a document, audio, or video data. A container, which is similar to buckets in S3, allows you to organise objects by grouping them. Swift simply provides API endpoints to store and manipulate objects. We cannot use Swift as a file system and objects are not accessible via any file sharing protocols.
- OpenStack Image Service aka Glance, is responsible for storage, discovery, and retrieval of virtual machine images. Glance can be configured to store VM images in Object Storage, Amazon S3, or simple file-system. Glance-registry and Glance-api are the two important components of Image Service. Glance-registry stores and retrieves metadata about images. Nova interacts with Glance using Glance-api for querying and retrieving actual VM images.

Moreover, Open VSwitch defines OpenStack network API which is intended to provide "network connectivity as a service" between devices managed by OpenStack compute service. The service is based on the notion of virtual networks that effectively are virtual L2 broadcast domains. We notice that on a high abstraction level, such virtual domains could correspond to ISONI VSNs, but as the focus of IRMOS is more on services, the detailed models of its VSNs differ significantly from those of OpenStack.

In OpenStack, a related network-level construct is Open VSwitch [http://openvswitch.org/] that is a NFV-like virtual switch that supports many useful features, e.g.:

- Visibility into inter-VM communication via NetFlow, sFlow(R), IPFIX, SPAN, RSPAN, and GRE-tunnelled mirrors
- LACP (IEEE 802.1AX-2008)
- Standard 802.1Q VLAN model with trunking
- A subset of 802.1ag CCM link monitoring
- STP (IEEE 802.1D-1998)
- Fine-grained QoS control
- Support for HFSC qdisc
- Per VM interface traffic policing

- NIC bonding with source-MAC load balancing, active backup, and L4 hashing
- OpenFlow protocol support (including many extensions for virtualisation)
- IPv6 support
- Multiple tunnelling protocols (Ethernet over GRE, CAPWAP, IPsec, GRE over IPsec)
- Remote configuration protocol with local python bindings
- Compatibility layer for the Linux bridging code
- Kernel and user-space forwarding engine options
- Multi-table forwarding pipeline with flow-caching engine

Our current working assumption is that FUSION could use the set of APIs to the cloud infrastructure (OCCI in particular) for accessing IaaS services and thus maintaining a level of isolation from internal details of data center architecture. Such approach would position FUSION as an application running on top of cloud platforms. As such FUSION may thus require to request specific capabilities from the cloud platform – a topic that will be studied in more detail by other WPs throughout the project.

8.3 NaaS for the convergence of Cloud Computing and Networking

SOA principle has been adopted in many Cloud service provisioning platforms. An observed trend is towards exposing network services in a SOA manner - in the form of Network as a Service (NaaS) - based on the network virtualisation principle.

An extensive overview of related work is given in [DUYA12]. It is believed that SOA for delivering NaaS based on network virtualisation enables to rely on SOA standards for network service description, discovery, and composition etc., thus allowing to combine network resources and services with computing and storage Cloud services. The survey focuses mainly on state-of-the-art in service description, discovery and composition as applied to NaaS with applications to heterogeneous services (composite Network-Cloud services), and on the support of Cloud features such as adaptiveness, elasticity and mobility. In particular, several existing options for network services description languages are discussed including attempts to describe both computing and networking resources; moreover, the challenges related to describing QoS and specifying user requests for network services are stressed. Regarding service discovery, existing standards are reviewed to conclude that network service discovery framework for NaaS still requires development; identified challenges relate to QoS models, dynamicity, heterogeneity and mobility in service discovery. Service composition is discussed from the point of view of such required features as dynamicity and adaptivity (in addition to heterogeneity imposed by combining cloud and networking services). To this end, new approaches beyond traditional workflow systems such as those based on WS-BPEL are needed. For example, combined service composition and network routing is expected to optimise service composition in a network. Two particular works that have adopted this approach are described briefly in the following.

FP7 project GEYSERS proposes a novel architecture for coordinated provisioning of optical and IT resources as well as end-to-end service delivery to overcome limitations of network domain segmentation [VICA11]. They start from a novel resource plane definition that covers both network and IT resources, and design a network control plane that is able to provision such compound services in an on-demand fashion. It is assumed that such a plane will be operated based on appropriate extensions of GMPLS signalling protocols. From the FUSION perspective, the applicability of this framework to support fine-grained network services still needs to be evaluated. In contrast to GEYSERS focused on optical networks, and aiming at a shorter-term solution in order to avoid changes to existing network infrastructure, in [GHMA12] an architecture compliant with NaaS is proposed for fine-grained network resource reservation for the purpose of VM migration. The solution uses an overlay network for fine-grained QoS-enabled data path setup in the transport network based on anycast destination address to support VM migration (in the service routing control plane, and the Cloud management platform is a client for the Network management platform). The limitations of the latter proposition relate to that only VM image transfer delay is

considered and other application communication requirements are disregarded; moreover, the anycast routing scheme (routing protocol, form of routing information) remains unspecified. Nevertheless, the general concepts presented therein may be inspiring for FUSION provided that the combined service and transport network routing paradigm is adopted in FUSION.

8.4 Service-aware networks

In this section we present in more detail selected architectures that attempt to combine grid/cloud services and network services.

8.4.1 IRMOS/ISONI (Intelligent Service Oriented Network Infrastructure)

The general goal of IRMOS is to enhance SLAs in a grid/cloud computing platform with strict quality guarantees in the transport network. To this end, all elements of IRMOS platform (computational nodes, network links and storage boxes) should be able to provide guarantees to individual activities while the physical resources are shared across multiple services.

IRMOS provides means for automatic deployment of services on best fitting resources distributed in a network. The deployment and instantiation of developers' service is based on an abstract description of all the execution environment requirements of the service (in the form of Virtual Service Network, VSN), including the description of the interconnections between service components and their individual QoS demands. Within the IRMOS platform, ISONI (Intelligent Service Oriented Network Infrastructure) implements the overall architecture including resource control plane, path manager, and execution environment.

Computing and storage resources are managed by ISONI's resource manager in a way similar to that in clouds/grids (virtualisation, workflows, resource reservation, etc.). This is the responsibility of the execution environment that provides global resource management and allocation policy for scheduling services enhanced with real-time attributes. In particular, the composition of applications into service components in a workflow, and their timing requirements are taken into account.

IP overlay or clean-slate?	**IP overlay (capable of using also other transport technologies). IRMOS provides a framework for a QoS enabled cloud/grid architecture where QoS provisioning is based on the use of proprietary ISONI eXchange Box nodes to build virtual networks for each instance of a composite service.**
Composite services	Service composition is based on the model of Virtual Service Network (VSN) being a service graph describing both computational, storage and network level requirements of a service. A decision about where and when the application service components are to execute is made at reservation time; after that no changes are made concerning the component during run time such as migrating to another machine in case of breaking the SLA or finding a better resource for execution.
Service execution	Execution environment in grid style using task scheduling based on timing requirements of services and advanced reservation mechanisms. Changes to OS kernel required. Service encapsulation is achieved through VM. Workflow management realised on the IRMOS level
Service publication/registration	Information service is used by infrastructure providers to advertise the ISONI capabilities with the purpose to select (suggest) the appropriate candidates for deploying the

	applications (based on the capabilities, price, …).
Service orchestration	Workload manager orchestrates services globally at PaaS level while Workload enactors (one per each VSN) orchestrate service components within respective VSN.

Table 4: IRMOS/ISONI and FUSION

8.4.2 IMS/SIP-based architectures

IMS is a service control overlay based on SIP designed by 3GPP and ETSI for controlling session-based services in 3GPP and converged NGN networks. Apart from session-oriented services it also supports other functions as, e.g., simple subscribe/notify service, and instant messaging and presence based on SIP.

IP overlay or clean-slate?	**IP overlay. IMS provides a framework for control and delivery of session-based services over diverse IP-based networks with QoS support by the underlying networks, enhanced with simple composition of component services that can be plugged into the service through rerouting the signalling messages.**
Composite services	Triggered at the level of S-CSCF based on IFC that constitute a simple workflow management logic and match trigger criteria with application servers. Chaining at the IMS level is in principle static in the sense that IFC is a static structure; dynamic behaviour is only possible due to matching/no matching trigger criteria and/or due to application server decisions reflected in the type/content of SIP messages they issue.
Service execution	Application hosting infrastructure is not managed by IMS. However, services (application servers) can well be hosted in clouds. Service execution is triggered due to IFC and results in directing SIP messages to appropriate application servers.
Service publication/registration	Could be based on IMS presence service model based on SIP presence.
Service orchestration	IMS does not manage the location or number of running service instances. Only rerouting of messages by e.g. S-CSCF could be based on load balancing criteria, however, it is out of the scope of the standard.

Table 5: IMS and FUSION

8.4.3 NGSON: Next Generation Service Oriented Network

NGSON [NGSO11] was proposed by the IEEE, and is designed as an overlay framework for control and delivery of composite services over heterogeneous IP-based networks. In NGSON, service configurations can be customised and adapted to the dynamic context of users, devices, services and networks.

NGSON specifies a functional architecture that provides advanced service and transport-related functions to support context-aware, dynamically adaptive, and self-organizing networks. To this end it defines functional entities and abstract protocol mechanisms, but does not provide their efficient implementation. These functions are realised by strategically located nodes with service-specific

forwarding and control capabilities. According to NGSON, the general organisation of "the network" follows the split into the signalling/control plane (to achieve service awareness) and transport plane. NGSON is declared to target current transport networks (IP, and also other technologies), and also cloud networks (in the sense that the services supported by NGSON may run in clouds, and also the functions of NGSON platform can be run as cloud applications, e.g., for auto-scaling purposes).

IP overlay or clean-slate?	**IP overlay. NGSON provides a framework for control and delivery of composite or component services over diverse IP-based networks (e.g. legacy IP, P2P, IP Multimedia Subsystem), possibly with QoS support by the underlying networks.**
Composite services	No standardised protocol to register/publish composite services. No standardised notation to specify composite services. In[LEKA12], the Business Process Execution Language is used for workflow management of composite services. Dynamic chaining should be possible: decide at runtime how to chain base services in an appropriate order and which instances. Service Composition element optimises for performance the selection of service instances in a composite service. • a single service instance (or a list of suggested instances) is typically obtained from Service Discovery and Negotiation function. • Negotiation for optimal QoS for a single service invocation (both network and service aspects) is up to Service Policy Decision function.
Service execution	Application hosting infrastructure is not managed by NGSON. However, in [Shan] clouds are abstracted as services, and VM operations are forwarded as service requests to the underlying clouds.
Service publication/registration	Provided by the Service Register functional entity. It assists Service Routers to discover where and how to route a service request. It maintains current location information of the services.
Service discovery and resolution	Service Discovery and Negotiation is typically used when abstract service is specified in the request. The service requestor may provide a number of criteria, such as service interface, availability, QoS, SLA, version, network area, regional area. The Service Discovery and Negotiation function obtains a list of similar or relative services (instances) and returns it to the requester. In a second phase, Service Routing will contact the Service Register functional entity for name-to-address mapping.
Service orchestration	NGSON does not manage explicitly the location or number of running service instances. However, various forms of coordination between native NGSON and service orchestration are feasible, e.g., configuring NGSON's service routing based on data available from either NGSON context information management function or service coordination, or based on service discovery and negotiation policies.

Table 6: NGSON and FUSION

Suitability for FUSION: NGSON identifies several individual architectural components and functionalities that are relevant for FUSION. As of today, only the functional architecture of NGSON has been standardised, but no interface specifications are available. NGSON provides capabilities for service composition which is meant as a process of coordinating the invocation of several basic services (service atoms) in response to a single message received from the requester. The latter resembles in essence the operation of Initial Filter Criteria known from IMS (as opposed to IMS, NGSON does not impose any particular notation for such purposes, and one example is BPEL which has been adopted in a demo implementation of NGSON as reported in [LEKA12]). We note however that classical notations for service composition such as BPEL or IFC do not cater for transporting service data in channels that do not use service signalling which is an important aspect in FUSION. Thus, it seems that the extent to which FUSION can rely on such frameworks still needs further study. Our initial guess is that at least their appropriate extensions would be mandatory for the adoption in FUSION. Overall, NGSON could serve as a blueprint for the service routing and related capabilities in FUSION.

8.4.4 CCN (Content-Centric Network)

CCN architecture (a.k.a. NDN – Named Data Networking) has been proposed under the heading of Information-Centric Networks as a clean-slate solution for the future Internet [JSTP09], [NDN]. CCN defines a forwarding plane of a transport network that operates based on the route-by-name principle assuming hierarchical names of content objects. According to CCN, content objects (data chunks of named content) are delivered in response to requests specifying the name of the requested object.

IP overlay or clean-slate?	**Clean slate, although CCN "faces" can run on top of different layers including IP.**
Composite services	Not within the scope of CCN – composition has to be built at the "application" layer with respect to CCN.
Service execution	Not within the scope of CCN – the architecture is only concerned with the transport of (authenticated) named data chunks.
Service publication/registration	Could be built using routing capabilities of CCN (however, routing is still an open issue in CCN, although several propositions have been proposed).
Service orchestration	Not within the scope of CCN – has to be built at the "application" layer with respect to CCN.

Table 7: CCN and FUSION

8.4.5 NetInf

NetInf has been developed by FP7 SAIL project as a proposition for future Internet ICN. NetInf layer defines NetInf protocol for named data object publishing, searching and retrieval along with name-based routing and name resolution services.

IP overlay or clean-slate?	**IP overlay in the sense that can run on legacy IP networks, and that data transfer can be over TCP/IP stack. Clean slate in the sense that NetInf protocol can run without IP support. In the latter case NetInf data transfer resembles CCN, although reverse path is maintained by using label stack header (similar to Via header in SIP or HTTP).**
Composite services	Beyond the scope.
Service execution	Outside the scope of NetInf.
Service publication/registration	PUBLISH method is used for publishing data objects including objects themselves and their metadata. Details depend on the policies of the Name Resolution Service.
Service orchestration	Outside the scope of NetInf.

Table 8: NetInf and FUSION

8.4.6 PSIRP/PURSUIT

PSIRP/PURSUIT architecture introduces an ICN-like architecture that is based on a publish-subscribe forwarding paradigm where DHT-based Pub/Sub rendezvous function allows for matching publications with subscriptions, topology management and formation function serves the routing purposes to provide delivery information to the forwarding function.

IP overlay or clean-slate?	**Clean slate. Pub/Sub model assumed for data registration and search capabilities at the Rendezvous layer, and source routing based on Bloom filters is used in the data plane. Source routes are delivered by a separate Topology management and formation layer.**
Composite services	Not within the scope of PURSUIT – composition has to be built at the “application” layer with respect to PURSUIT.
Service execution	Not within the scope of PURSUIT – the architecture is concerned with the transport of (authenticated) named data chunks.
Service publication/registration	Could be built adopting the capabilities of the Rendezvous function. PURSUIT claims to support mobility, so Rendezvous potentially offers a good platform to support dynamic service instantiation.
Service orchestration	Not within the scope of PURSUIT - has to be built at the “application” layer.

Table 9: PSIRP/PURSUIT and FUSION

8.4.7 SERVAL

Serval [NDGK12] is a new modification to the internet architecture where service access is central. Serval allows users to find a service instance based on several metrics such as load and latency, and maintain a connection with that service instance. Serval introduces a service oriented layer (named SAL – Service Access Layer) located between the transport and the network layers. Its main role is to enable applications to communicate using service names.

IP overlay or clean-slate?	**IP overlay. Serval adds a Service Access Layer that sits between the transport and network layers. The SAL maps service names to IP addresses before connections are made. Once a connection is established, communication is based on standard IP.**
Composite services	Not within the scope of Serval – composition has to be built at the "application" layer with respect to Serval.
Service execution	Serval does not manage the service placement. On-path Serval load-balancers can choose which service instance a request is forwarded to. The logic behind is out of scope of Serval.
Service publication/registration	An application calls bind() on a Serval socket API and registers a local forwarding entry. Local service controller can propagate this local announcement to upstream service controllers.
Service orchestration	Serval does not manage the location or number of running service instances. More generally, Serval has not insight into the performance at the level of network/compute resources; it does not define any monitoring capabilities for the service controller plane to gather relevant performance statistics. Thus, in order to make the service controller plane resolve services based on performance, which is important for FUSION, dedicated means (e.g., monitoring) would have to be provided.

Table 10: SERVAL and FUSION

8.4.8 eXpressive Internet Architecture (XIA)

XIA [HADL12] is the result of collaboration between Boston University, Carnegie Mellon University and the University of Wisconsin/Madison. XIA recognises the shift from IP-based Internet to new architectures focusing on different principals, such as content or users. XIA describes an architecture designed to support different principals and allows new principals to be added over time. It also allows incremental deployment due to a fall back mechanism for routers which don't support XIA.

IP overlay or clean-slate?	**Both. XIA addressing uses clean slate service ID addressing schemes, but it allows for intermediate routers to fall back to IP based addressing if XIA is not supported.**
Composite services	No explicit support provided. Chaining requests is possible: the user gives flexibility to the network to find the content.
Service execution	XIA does not manage the execution process. Each application uses its own protocol.
Service publication/registration	Services use a XIA socket API: a service calls bind() to bind itself

	to the public key of that service. This bind inserts the service ID in the host's forwarding table. Available services are propagated, following a given scope. Propagating this information is future research.
Service orchestration	XIA does not manage the location or number of running service instances.

Table 11: XIA and FUSION

8.4.9 Summary: Models for Service-Aware Networking

In this section we summarise the discussion given in previous subsections by presenting a high-level overview of the way selected state-of-the-art architectures enable networks to be service aware.

8.4.9.1 IRMOS

Network resources: IRMOS/ISONI follows a bandwidth-broker based network resource management in the style of NaaS that is additionally coupled with compute power resource management by means of a common deployment manager functional block. Virtual network resources are provided by proprietary ISONI IXB nodes that overlay virtual links on physical network resources using ISONI-specific policies and mechanisms.

Compute resources: explicit, full control in Grid/Cloud style in coordination with network resources (due to common Deployment Manager). Actually, the IRMOS platform provides two main models for cloud services: PaaS (IRMOS level) and IaaS (ISONI level).

Storage resources: as compute resources, and based on the Lustre file system.

8.4.9.2 IMS/SIP

Network resources: Out-of-band service routing plane based on SIP can access the SPD FE (Service Policy Decision) that derives and translates service requirements and negotiates network-level QoS. Essentially, physical transport of user data is separate from request routing.

Compute resources: no sophisticated capabilities for controlling compute resources. Simple re-routing of SIP initial requests is possible by S-CSCF (Serving Call-Session Control Function) function towards application servers in order to trigger services; this re-routing is done based on Initial Filtering Criteria (IFC) downloaded by S-CSCF from HSS on user's registration. An IFC is a logical expressions matching a message to bind triggers with application servers and from FUSION perspective can be seen as a simple service graph.

Storage resources: no such capabilities.

8.4.9.3 NGSON

Network resources: NGSON follows the IMS-style approach to use service routing overlay that can request QoS levels for data streams from the transport layer. Here, SPD FE (Service Policy Decision) derives and translates service requirements (seen at the service routing level in request messages) to the transport network QoS and negotiates network-level QoS. Essentially, physical transport of user data is thus separate from service request routing. For example, CD (Content Delivery) FE is assumed to support different types of cache functions with multiple protocols like, e.g., HTTP, FTP, P2P (such as BitTorrent), and RTSP.

Compute resources: implicit control based on static/dynamic information about services stored in SDN FE and SReg FE, and through appropriate choices of SC FE and SR FE. However, NGSON does not take decisions about service deployment.

Storage resources: only implicitly controlled through the internal capabilities of content delivery (CD) FE (CDN-like black-box entity).

8.4.9.4 CCN

Network resources: can only control transmission links (strategy layer), but currently "strict" QoS guarantees are not possible.

Compute resources: no control; should be built "over the top" of CCN.

Storage resources: no control apart from the caches; should be built "over the top" of CCN.

8.4.9.4.1 SERVAL

Network resources: currently no QoS control possible; this function could be attributed to the service controller layer, but additional capabilities like monitoring should be built into the platform.

Compute resources: no control; possibly could be built at least partially into service control layer.

Storage resources: no control apart from the caches; possibly could be built at least partially into service control layer.

9. SUMMARY

This document presented an architecture for service-oriented networking as envisioned by the FUSION project. The document introduced the advantages and benefits of adopting a service-oriented approach for a range of applications/service use cases, drawing the set of requirements for the system behaviour. The overall system architecture was defined, identifying the key functional blocks and the interfaces between them. A range of possible business models were introduced highlighting the interactions between the different actors and roles involved in the provision and operation of services. A set of security threat models were analysed to identify security issues beyond those applicable to general cloud computing systems. Finally, related architectures, technologies and projects were reviewed highlighting the relevance to the FUSION system and identifying the key differences of the FUSION approach to service-oriented networking.

This deliverable presented the initial specification of the FUSION system – the service architecture and service networking are being studied in WP3 and WP4 respectively and more a more detailed description of the functions, algorithms and protocols are available in deliverables D3.1 [D3.1] and D4.1 [D4.1]. The initial overview of the interfaces and interactions specified in this document will be refined in deliverable D2.2 in year 2 of the project. The final system architecture will be documented together with an evaluation of the cross-layer models in deliverable D2.3 in year 3 of the project.

10. REFERENCES

[AKOG05] Akogrimo Deliverable D4.3.1, "Architecture of the Infrastructure services Layer V1": http://www.akogrimo.org/download/Deliverables/Akogrimo_D431_Version_1_Final.pdf

[CAJF11] J. Cheny, M. Arumaithuraiy, L. Jiao, X. Fu, K.K. Ramakrishnan, COPSS: An Efficient Content Oriented Publish/Subscribe System, 2011 Seventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS 2011), New York, 2011.

[Chan] Distributed and Dynamic Mobility Management in Mobile Internet: Current Approaches and Issues. Journal of Communications, vol. 6(1), Feb 2011

[CLOUST] CloudStatus: http://www.cloudstatus.com

[D3.1] FUSION deliverable D3.1, "Initial specification of algorithms and protocols for service-oriented network management", FUSION consortium, December 2013.

[D4.1] FUSION deliverable D4.1, "Initial specification of service-centric routing protocols, forwarding and service-localisation algorithms", FUSION consortium, December 2013.

[DUYA12] Q. Duan, Y. Yan, A. Vasilakos, "A survey on Service-Oriented Network Virtualization Toward Convergence of Networking and Cloud Computing", IEEE Trans. Networks and Service Management, Dec. 2012.

[ENISA12] European Union Agency for Network and Information Security (ENISA), Cloud Computing Benefits, risks and recommendations for information security, December 2012.

[FASCINATE] FASCINATE FP7 project, http://www.fascinate-project.eu/.

[FOST04] J. Foster, C. Kesselman, "Grid2", 2004.

[FOST08] I. Foster, Y. Zhao, I. Raicu, S. Lu, "Computing and Grid Computing 360-Degree Compared", 2008.

[GHMA12] M. Gharbaoui, B. Martini, P. Castoldi, "Anycast-Based Optimizations for Inter-Data-Center Interconnections", J. Opt. Commun. Netw., Vol. 4, No. 11, November 2012.

[GOUG07] A McGough, et al., "GRIDCC: real-time workflow system", In Proceedings of the 2nd Workshop on Workflows in Support of Large-Scale Science, Monterey, California, USA, 2007.

[HAAW13] M. Hoque, S. Obaid Amin, A. Alyyan, L. Wang, B. Zhang, L. Zhang, NLSR: Named-data Link State Routing Protocol, The 3rd ACM SIGCOMM Workshop on Information-Centric Networking, August 2013.

[HADL12] Dongsu Han et al. XIA: Efficient Support for Evolvable Internetworking. The 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI'12), San Jose, CA, April 2012.

[HUSH10] X. Huang, S. Shanbhag, and T. Wolf, "Automated service composition and routing in networks with data-path services," in Proc. 2010 IEEE International Conference on Computer Communication Network, 2010.

[HYPEHQ] Hyperic HQ: http://www.hyperic.com

[IEEE] IEEE Standard for the Functional Architecture of Next Generation Service Overlay Networks.

[ietf-dmm] H. Chan et al. Requirements for Distributed Mobility Management. Informational Internet Draft, May 2013.

[JSTP09] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, R. L. Braynard (PARC) Networking Named Content, CoNEXT 2009, Rome, December, 2009. http://www.ccnx.org/.

[JZEA09] Jokela, P., Zahemszky, A., Esteve, C., Arianfar, S., and Nikander, P., LIPSIN: Line Speed Publish/Subscribe Inter-Networking. SIGCOMM'09. http://www.psirp.org/

[Lee] Lee et al. NGSON: Features, State of the Art, and Realization. IEEE Communications Magazine, January 2012

[LEKA12] Seung-Ik Lee, Shin-Gak Kang, NGSON: features, state of the art, and realization, IEEE Communications Magazine, Jan 2012.

[M1] "A comparative study of the current Cloud Computing technologies and offers", 2012, Meriam Mahjoub, Afef Mdhaffar, Riadh Ben Halima, Mohamed Jmaiel, University of Sfax, ReDCAD Laboratory, B.P. 1173, Sfax-Tunisia

[M10] "Hybrid cloud monitoring", oct 2012, Jernej Vičič, Andrej Brodnik, UP IAM, Muzejskitrg 2, 6000 Koper, UL FRI, Tržaška 25, 1000 Ljubljana.

[M11] "A Service-Oriented Measurement Infrastructure for Cloud Computing Environments", 2012, Gregg Hamilton, Dimitrios Pezaros, School of Computing Science, University of Glasgow, Glasgow, UK

[M12] "A Scalable Architecture for Real-Time Monitoring of Large Information Systems", 2012, Mauro Andreolini, Michele Colajanni, Marcello Pietri, University of Modena and Reggio Emilia, Via Vignolese, 905/b - 41125 Modena, Italy

[M2] "Cloud Monitoring: a Survey", Jan 2013, Giuseppe Aceto, Alessio Botta, Walter de Donato, Antonio Pescapè, University of Napoli Federico II (Italy).

[M3] "Virtual resource monitoring in cloud computing", 2011, HAN Fang-fang, PENG Jun-jie , ZHANG Wu , LI Qing , LI Jian-dun , JIANG Qin-long , YUAN Qin, Shangai University.

[M4] "Smart System Resource Monitoring in Cloud", Feb 2012, Rich C. Lee, Computer Science, National Taipei University of Technology, Taipei, Taiwan, China.

[M5] Resource Usage Monitoring in Clouds, 2012, Mohit Dhingra, J. Lakshmi, S. K. Nandy, CAD Lab, Indian Institute of Science, Bangalore 560 012, India.

[M6] Performance Monitoring in Linux KVM Cloud Environment, 2012, Anshuman Khandual, Linux Technology Centre, IBM Systems and Technology Lab.

[M7] PAPI-V: Performance Monitoring for Virtual Machines, 2012, Matt Johnson, Heike McCraw, Shirley Moore, Phil Mucci, John Nelson, Dan Terpstra, Vince Weaver, Electrical Engineering and Computer Science Dept., University of Tennessee, Knoxville, TN 37996 & Tushar Mohan, Minimal Metrics.

[M8] "Monitoring and Detecting Abnormal Behavior in Mobile Cloud Infrastructure", 2010, Taehyun Kim, Yeongrak Choi, Seunghee Han, Jae Yoon Chung, Jonghwan Hyun, Jian Li, and James Won-Ki Hong, Department of Computer Science and Engineering, POSTECH, Pohang, Korea.

[M9] "M4CLOUD - Generic Application Level Monitoring For Resource-Shared Cloud Environments", 2012, Toni Mastelic, Vincent C. Emeakaroha, Michael Maurer and Ivona Brandic Information Systems Institute, Vienna University of Technology, Argentinierstrasse 8/184-1, A-1040 Vienna, Austria.

[NDGK12] Serval: An End-Host Stack for Service-Centric Networking. By Erik Nordstram, David Shue, Prem Gopalan, Rob Kiefer, Matvey Arye, Steven Ko, Jennifer Rexford, and Michael

J. Freedman. In Proc. 9th Symposium on Networked Systems Design and Implementation (NSDI 12), San Jose, CA, April 2012.

[NDN] Named Data Networking, project Web page, available at http://named-data.net/project/

[NGSO11] NGSON (Next Generation Service Oriented Network) – IEEE 1903, "Standard for the Functional Architecture of Next Generation Service Overlay Networks", 2011.

[OpenStack] https://www.simple-talk.com/cloud/infrastructure-as-a-service/introduction-to-openstack/

[OSGA06] The Open Grid Services Architecture, Version 1.5, 2006. Available at http://forge.gridforum.projects/ogsa-wg.

[PAJA11] Android & iPhone Augmented Reality War: Should Apple Join Forces With Facebook? http://internationaldigitalmarketing.com/2011/03/07/android-iphone-augmented-reality-war-should-apple-join-forces-with-facebook/

[PHOS07] Deliverable D.5.2., "QoS-aware Resource Scheduling", Project Phosphorus.

[QSC13] Qualcomm aims to solve the coming mobile data crunch with small cell base stations. http://www.engadget.com/2013/05/01/qualcomm-small-cell-network/

[RAIC07] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, M. Wilde. "Falkon: a Fast and Light-weight tasK executiON framework", IEEE/ACM SuperComputing 2007.

[SAIR08] "State of the Art on IRMOS technologies", Interactive Realtime Multimedia Applications on Service Oriented Infrastructures ICT FP7-214777, Deliverable D2.3.1, 2008.

[SCHW05] U. Schwiegelshohn, R. Yahyapour, P. Wieder, "Resource Management For Future Generation Grids", TBC.

[Shan] Shan et al. Inter-Cloud Operations via NGSON. IEEE Communications Magazine, January 2012.

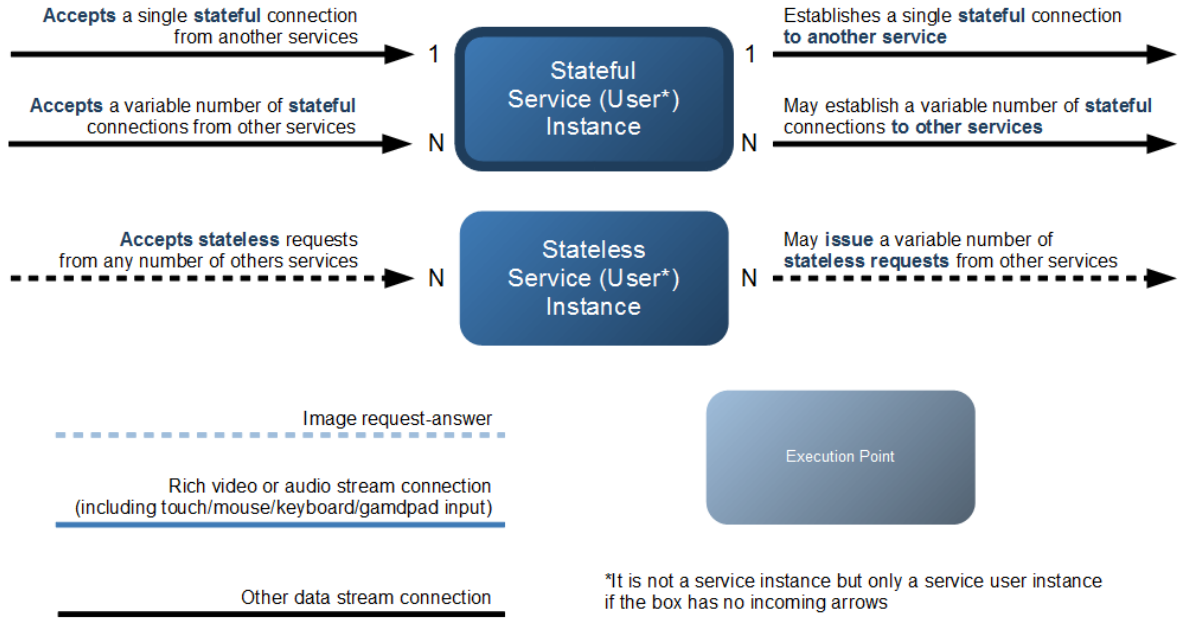
[VICA11] P. Vicat-Blans et al., "Bringing Optical Networks to the Cloud: An architecture for a Sustainable Future Internet", Lect. Notes Comp. Sci., Vol. 6656/2011, pp. 307–320, Apr. 2011.

[YUBU05] J. Yu, R. Buyya, "A Taxonomy of Workflow Management Systems for Grid Computing", Journal of Grid Computing, Vol. 3, Issue 3-4, Springer, 2005.

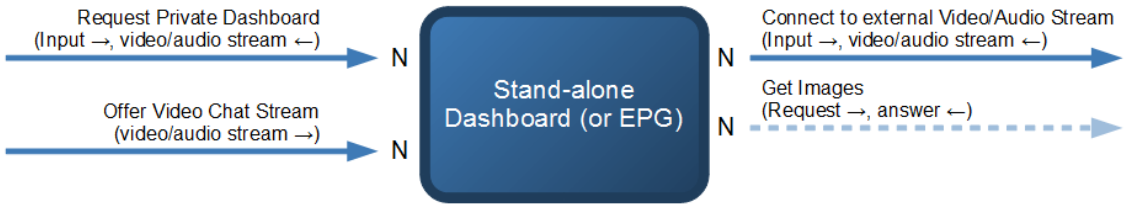
[ZHRA08] Y. Zhao, I. Raicu, I. Foster. "Scientific Workflow Systems for 21st Century, New Bottle or New Wine?" IEEE Workshop on Scientific Workflows, 2008.

11. APPENDIX A: SERVICE COMPONENTS

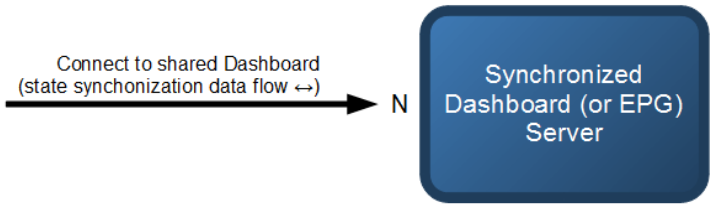
This legend explains the diagram elements used in Appendix B: Service Components and Appendix C: Service Configuration Scenarios. It distinguishes between *stateless* and *stateful* services. It also distinguishes basic classes of communication data streams between service instances.



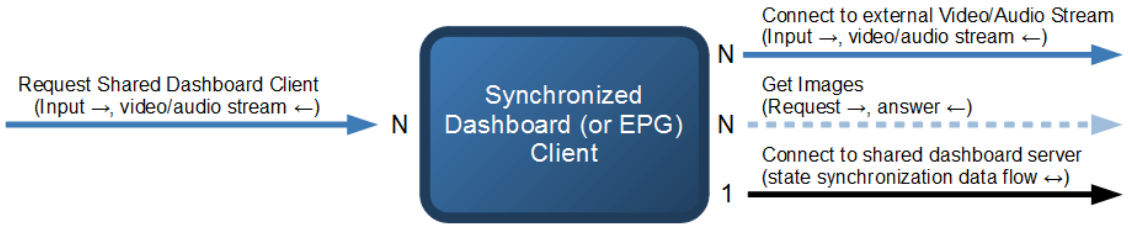
Single-Person Dashboard



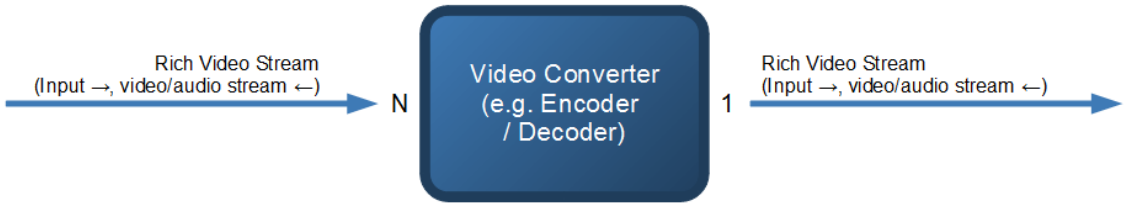
Server for a Multi-User Dashboard with Individual Views



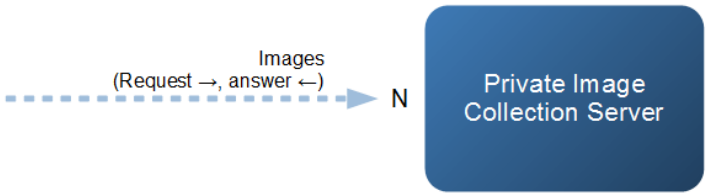
Client for A Multi-User Dashboard with Individual Views



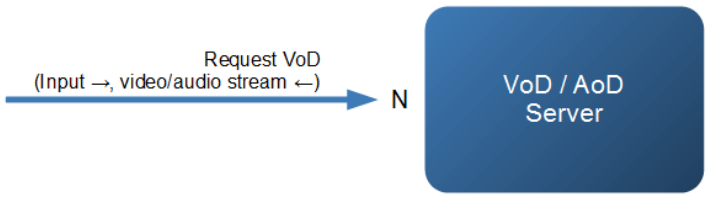
Video Encoder or Decoder



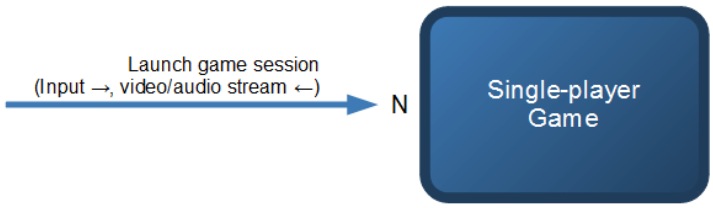
Server for a Private Image Collection



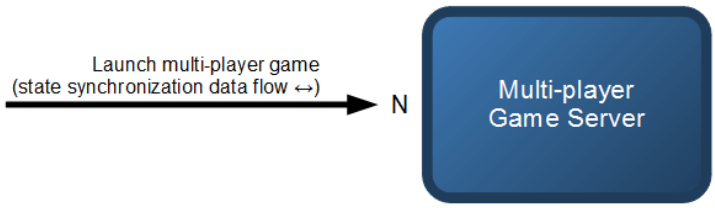
Video on Demand (VoD) Server



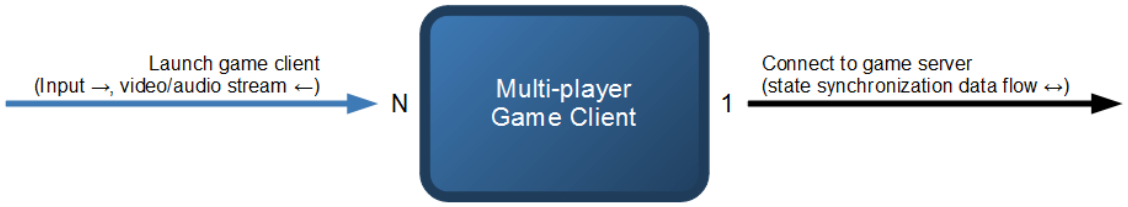
Single-Player Game



Multi-Player Game Server

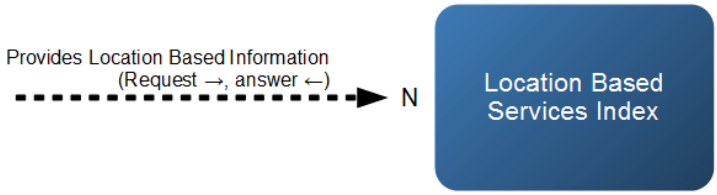


Multi-Player Game Client

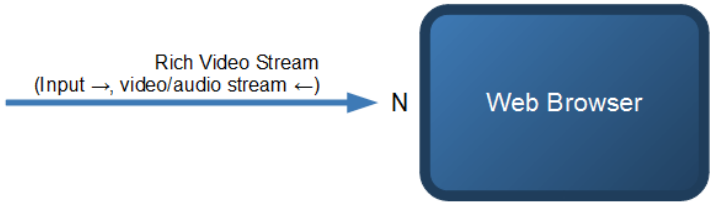


Augmented Reality Overlay Renderer

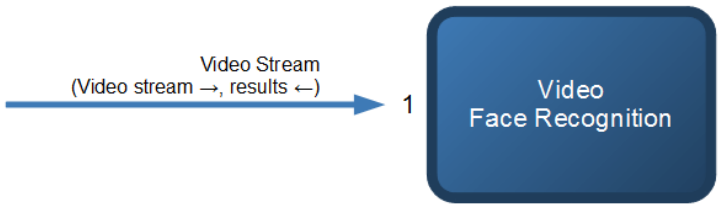
Location Based Services Index/Database



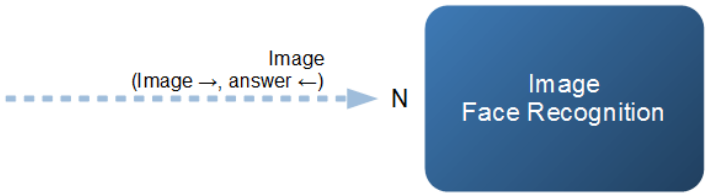
Web Browser



Face Recognition in Video Stream

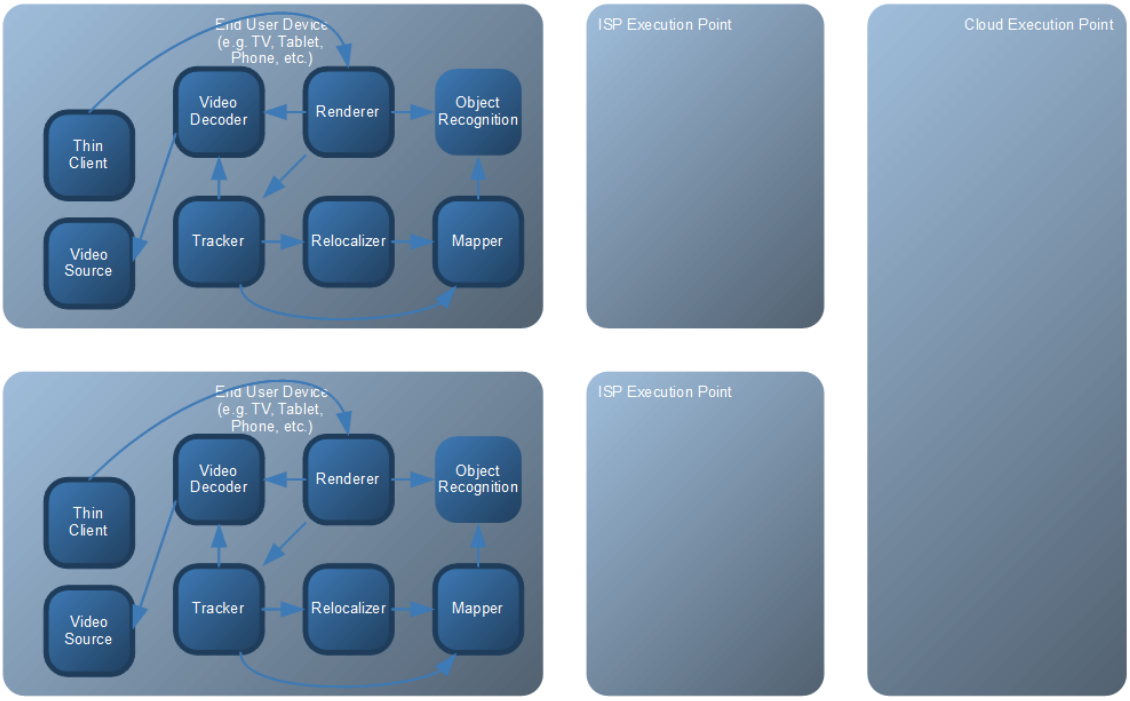


Face Recognition in Individual Images

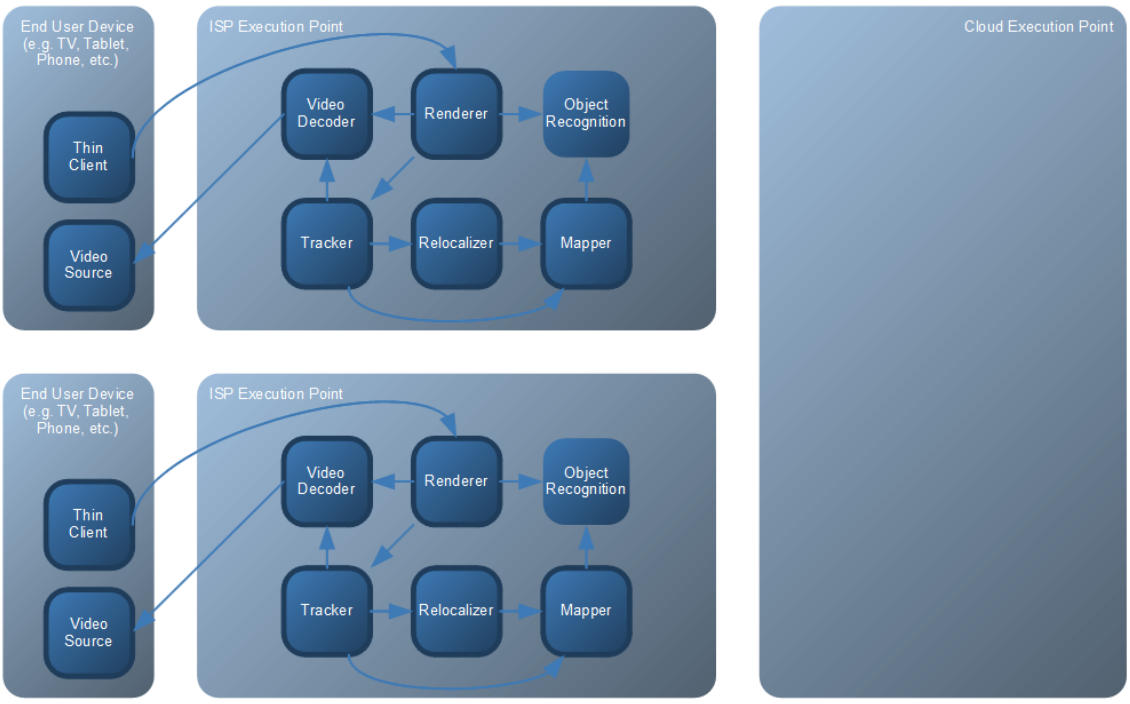


12. APPENDIX B: 7 SERVICE CONFIGURATION SCENARIOS

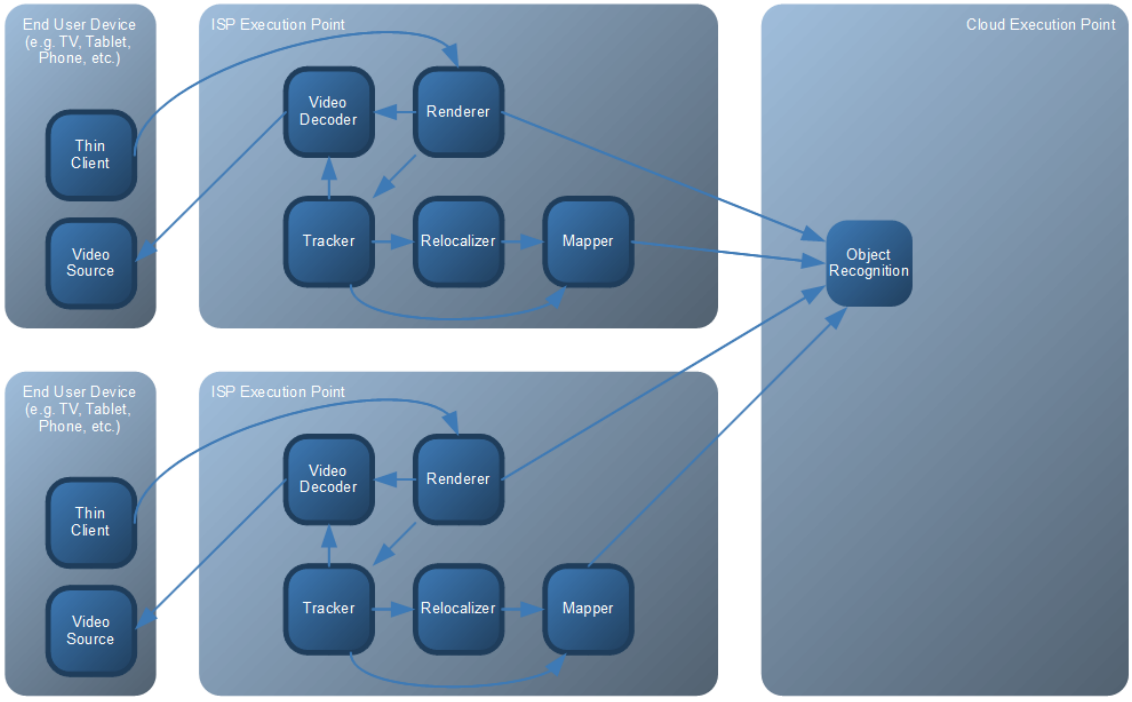
Augmented Reality Application on User-Device



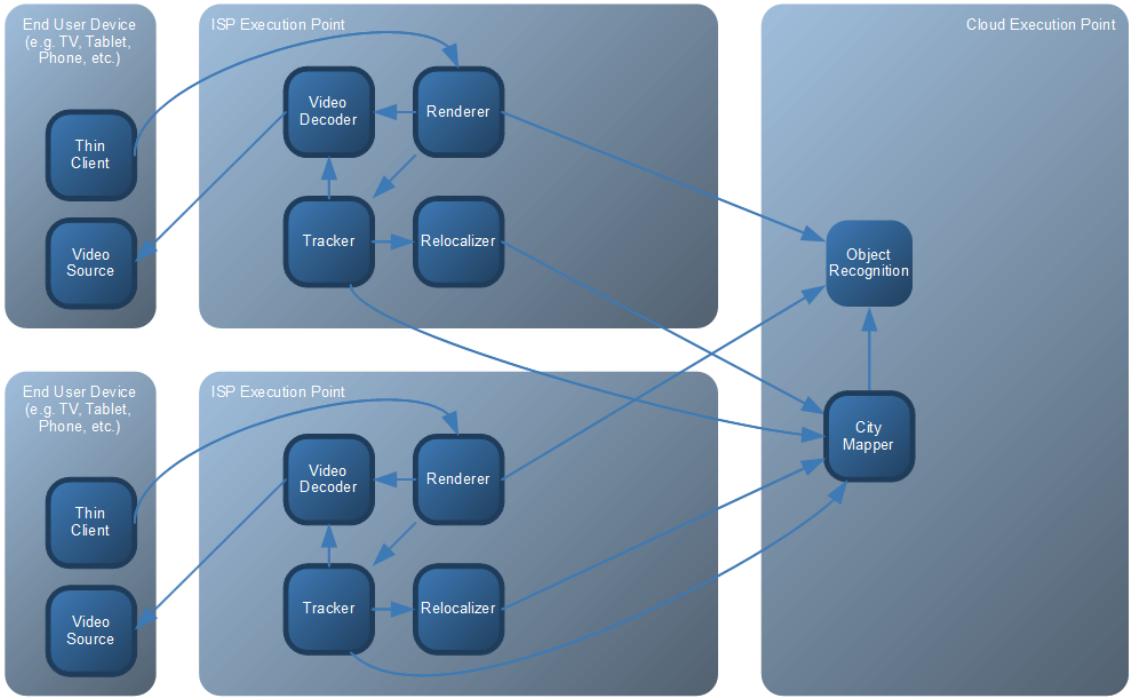
Thin Client Augmented Reality Application



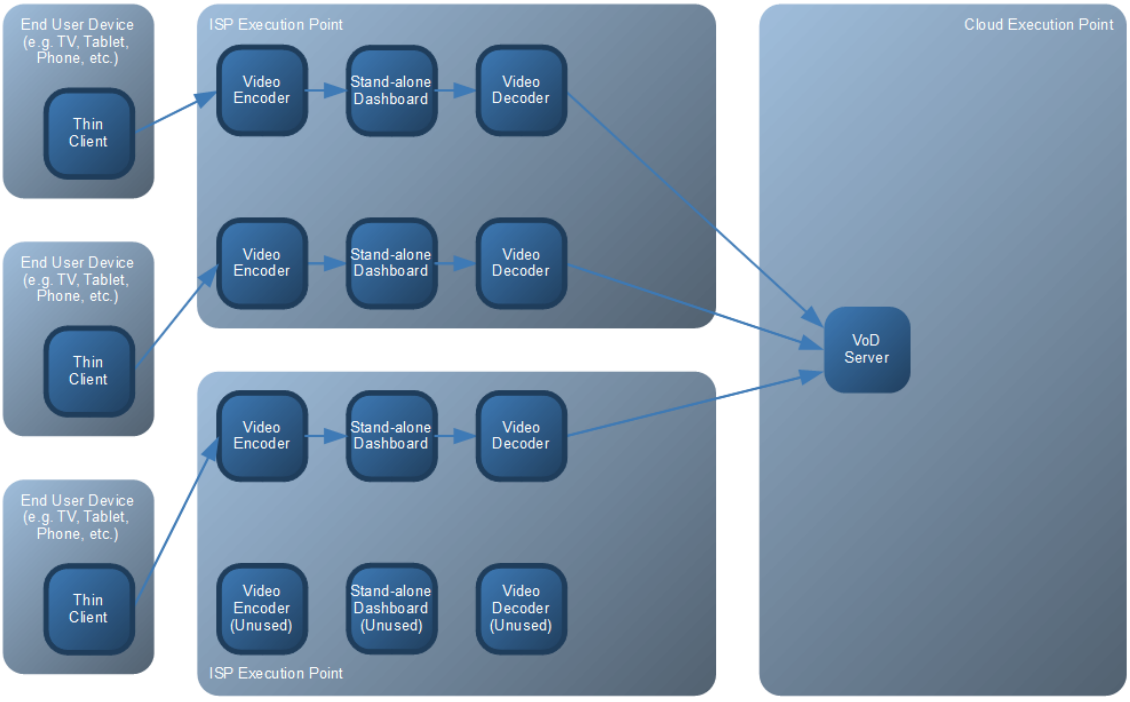
Thin Client Augmented Reality Application, Partly in Cloud



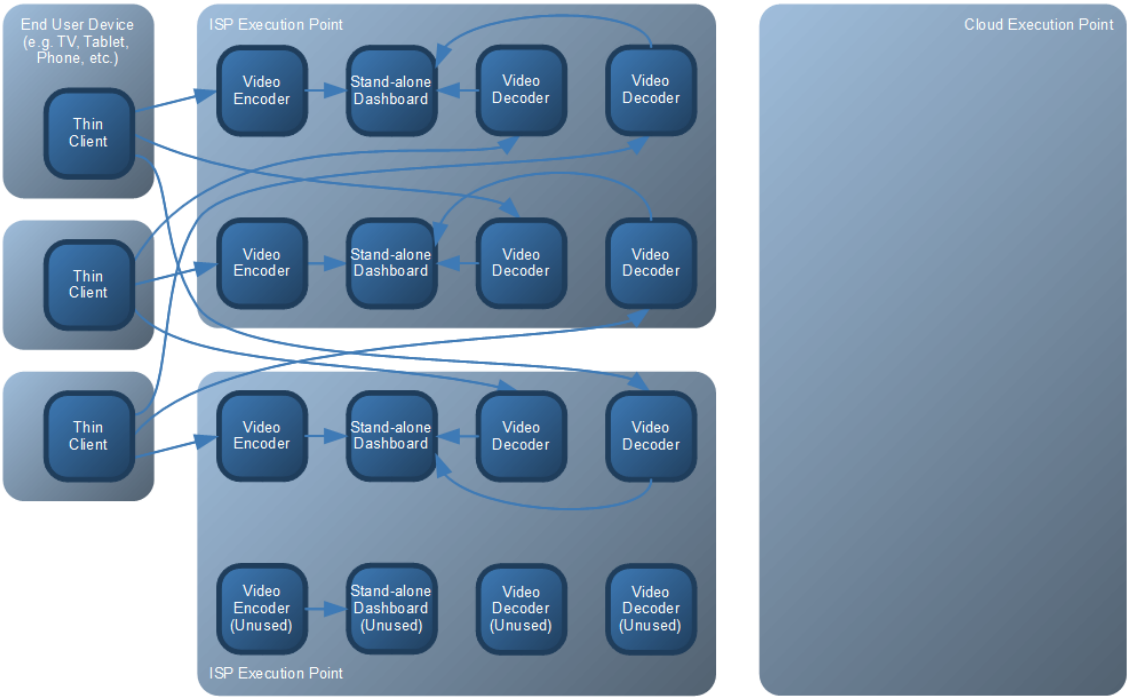
Thin Client Augmented Reality Application, More in Cloud



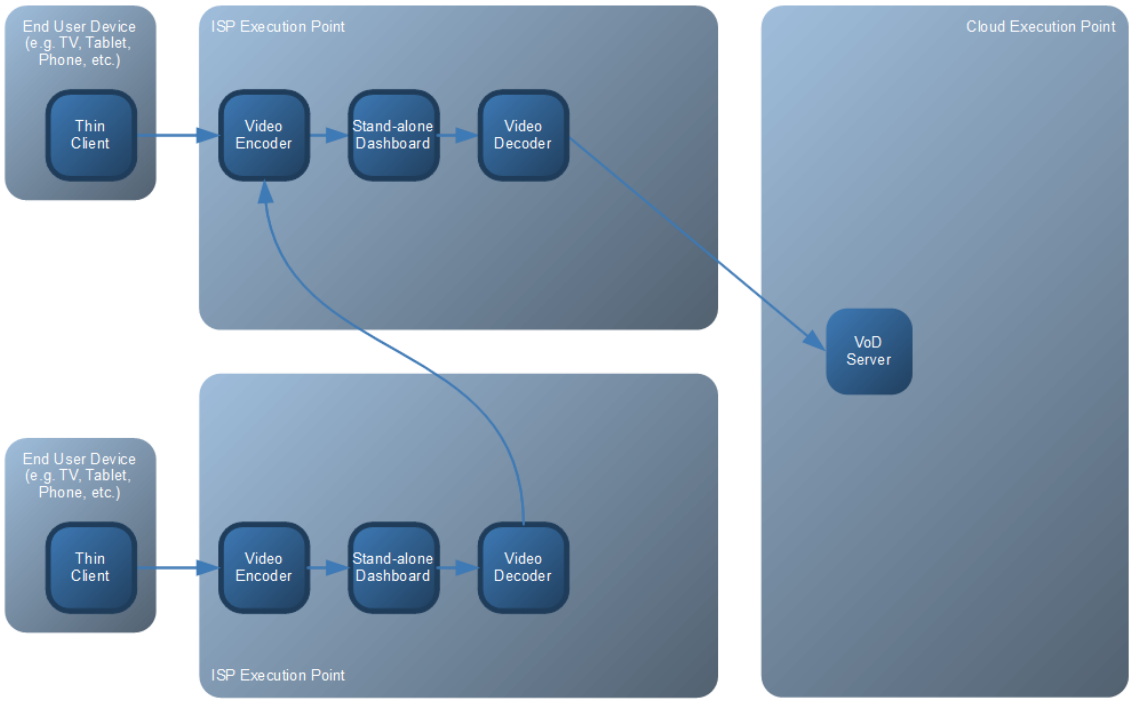
VoD Dashboard



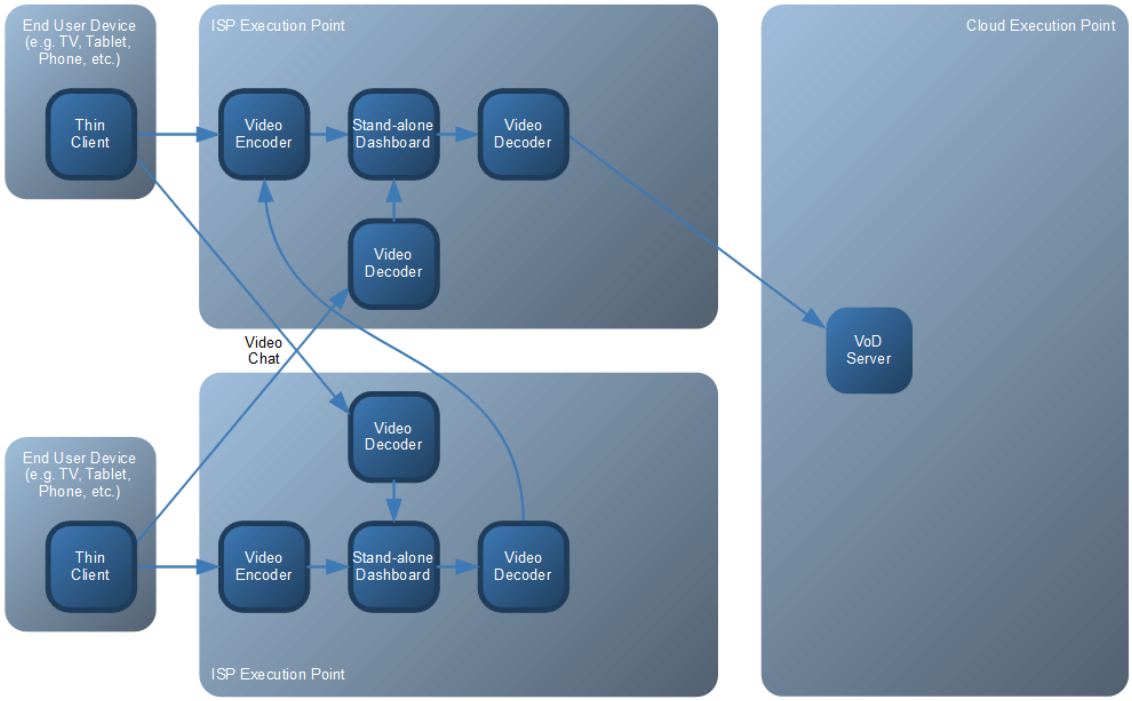
Triple Video Chat



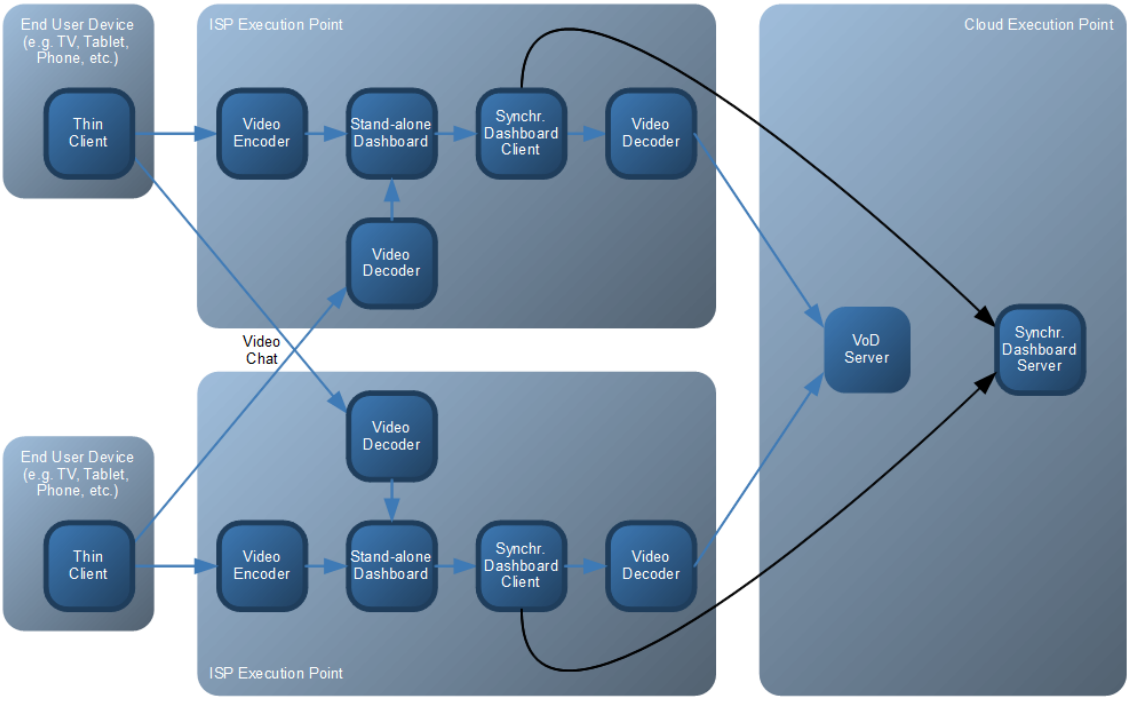
Helping a Friend with his VoD Dashboard



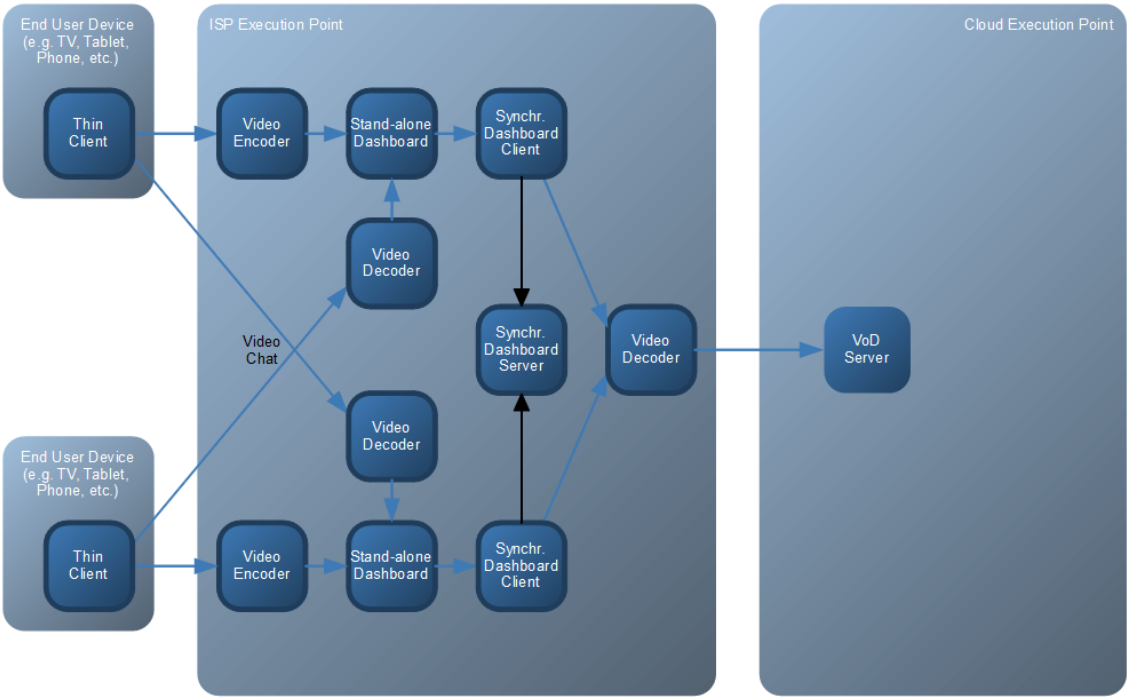
Helping a Friend with his VoD Dashboard with Video Chat



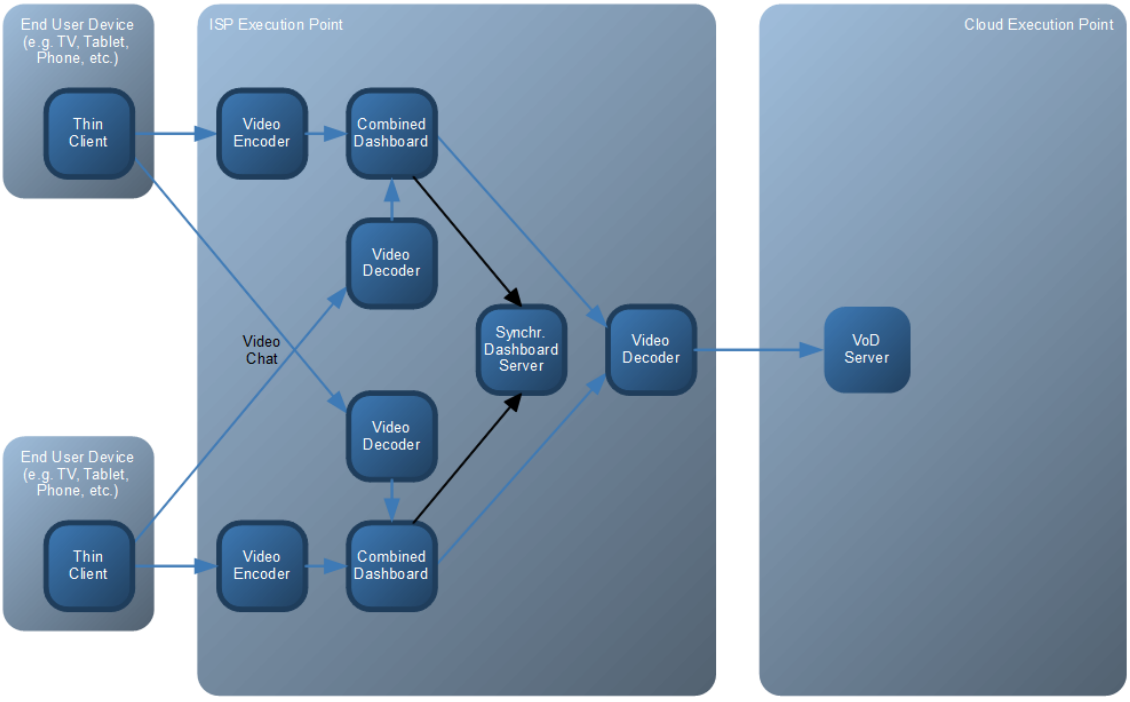
Synchronised VoD Dashboard



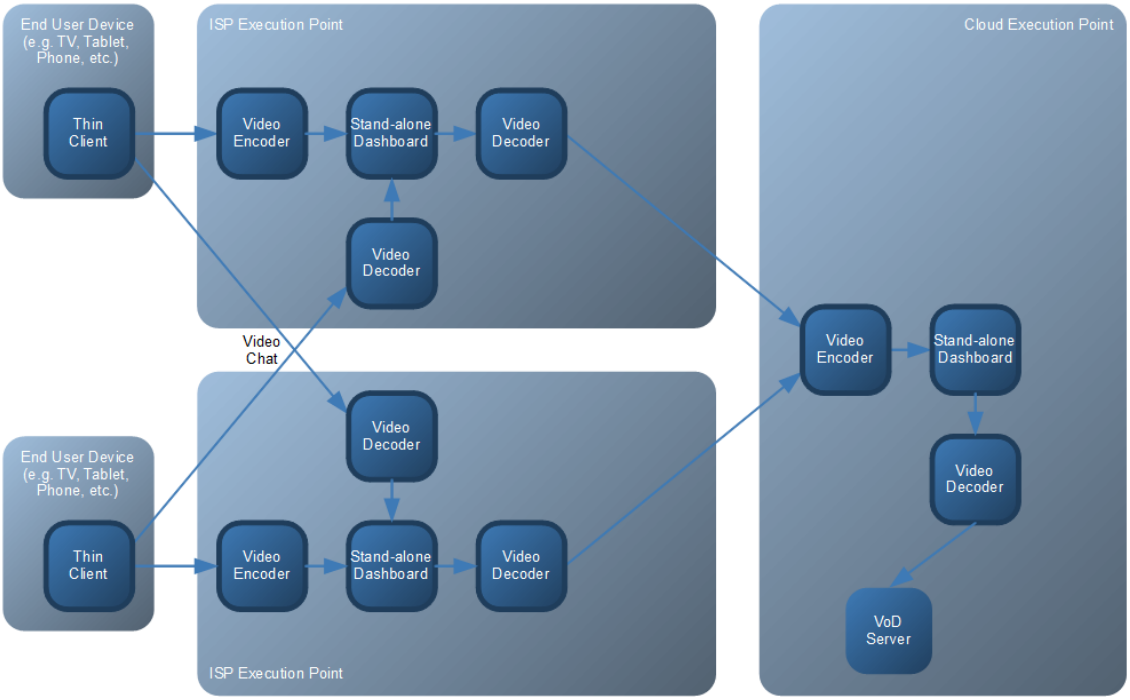
Synchronised VoD Dashboard on same ISP



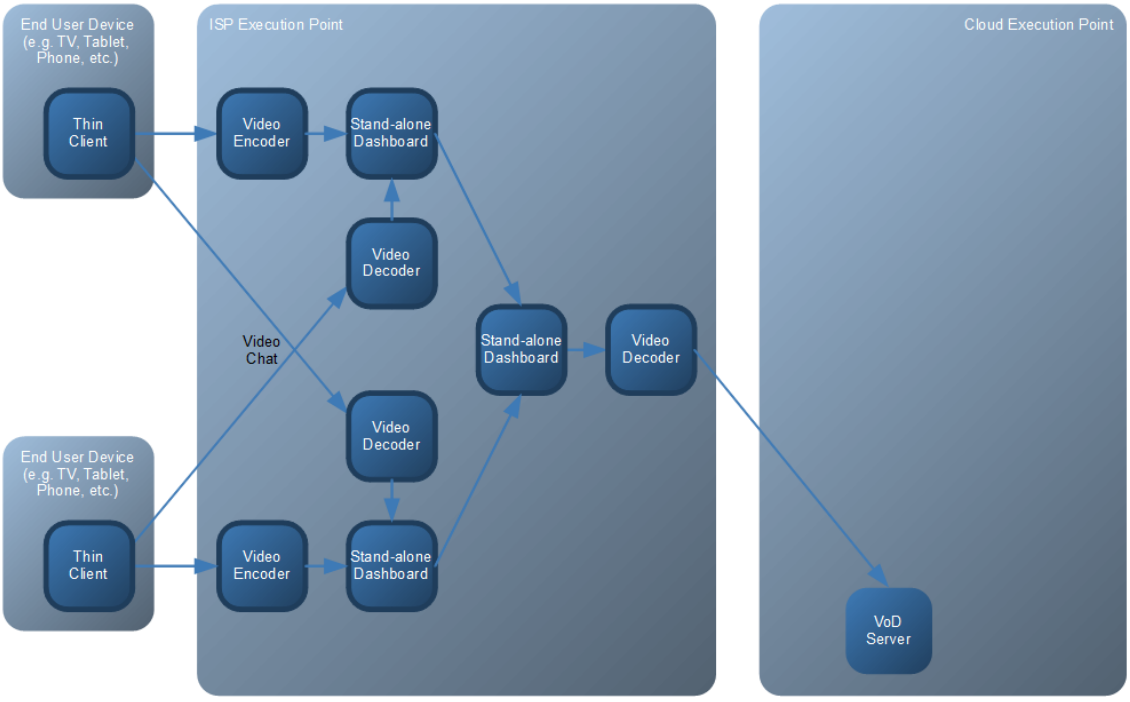
Merged Synchronised VoD Dashboard on same ISP



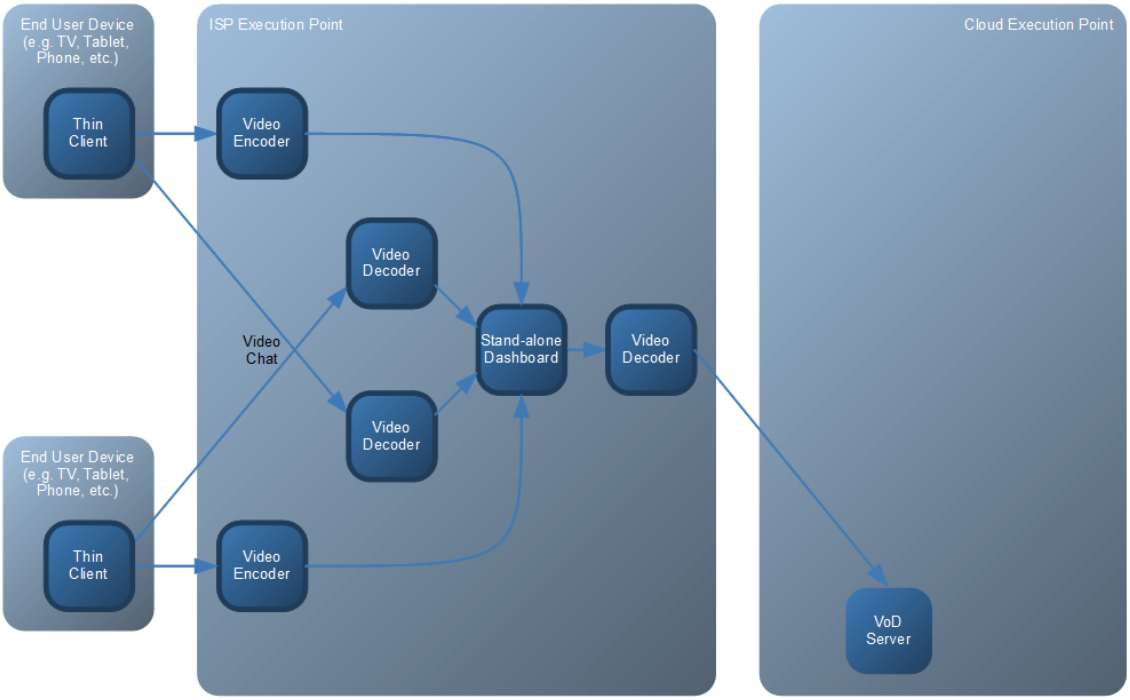
Common VoD Dashboard



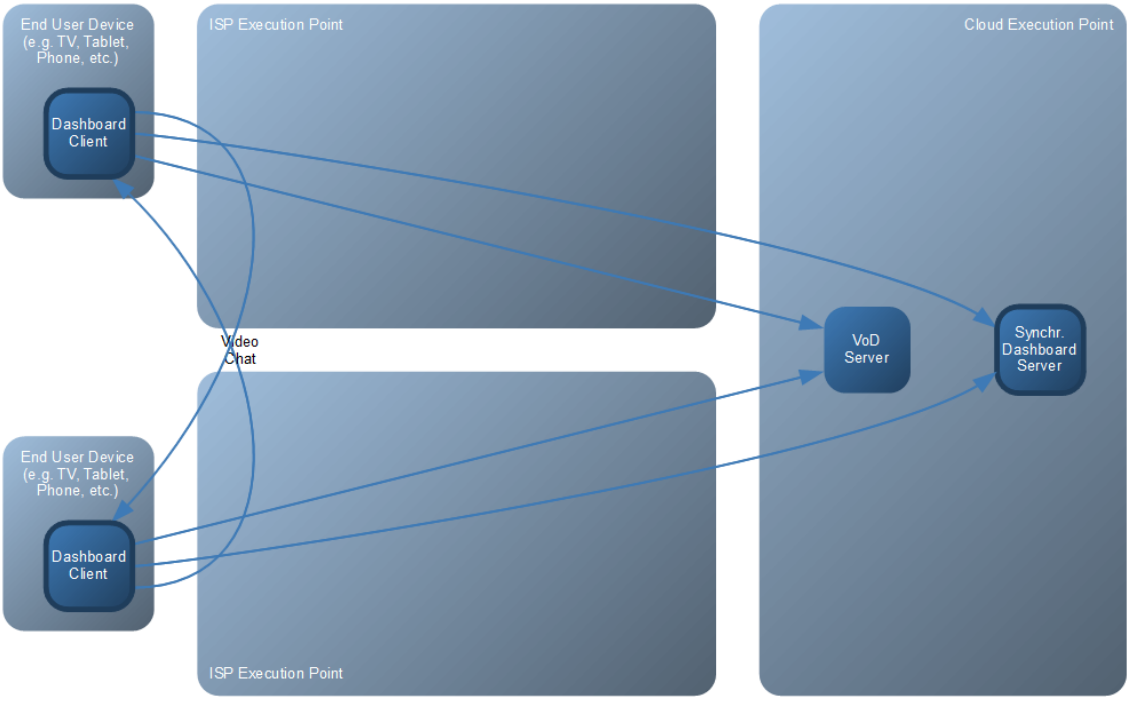
Common VoD Dashboard on same ISP



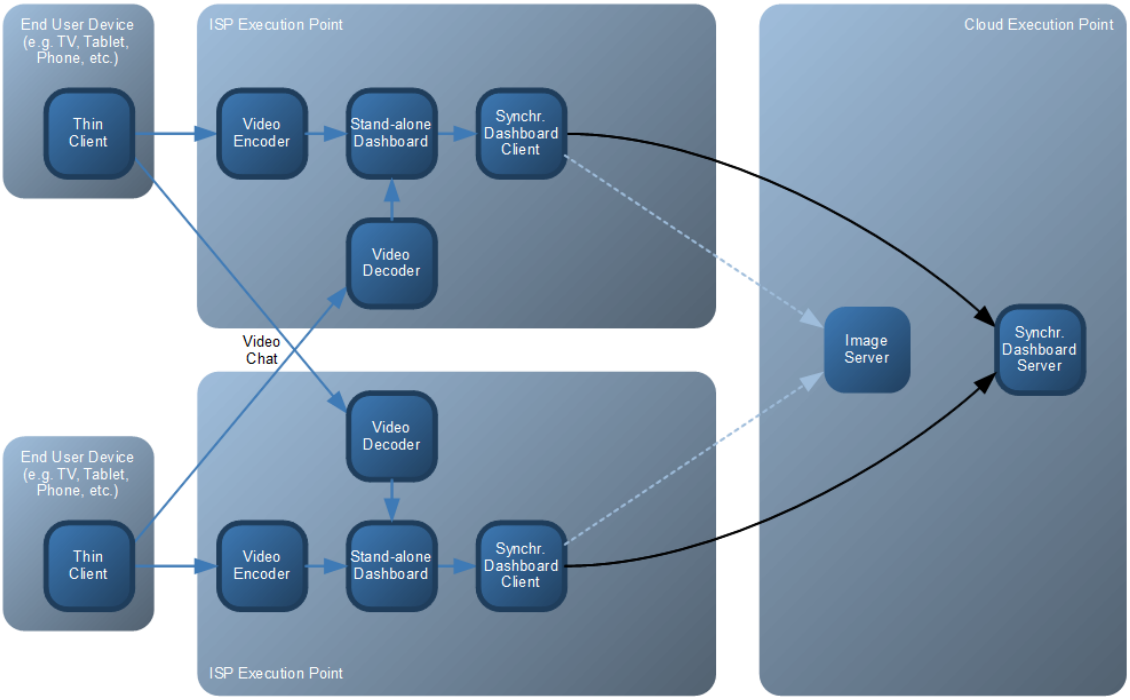
Identical VoD Dashboard



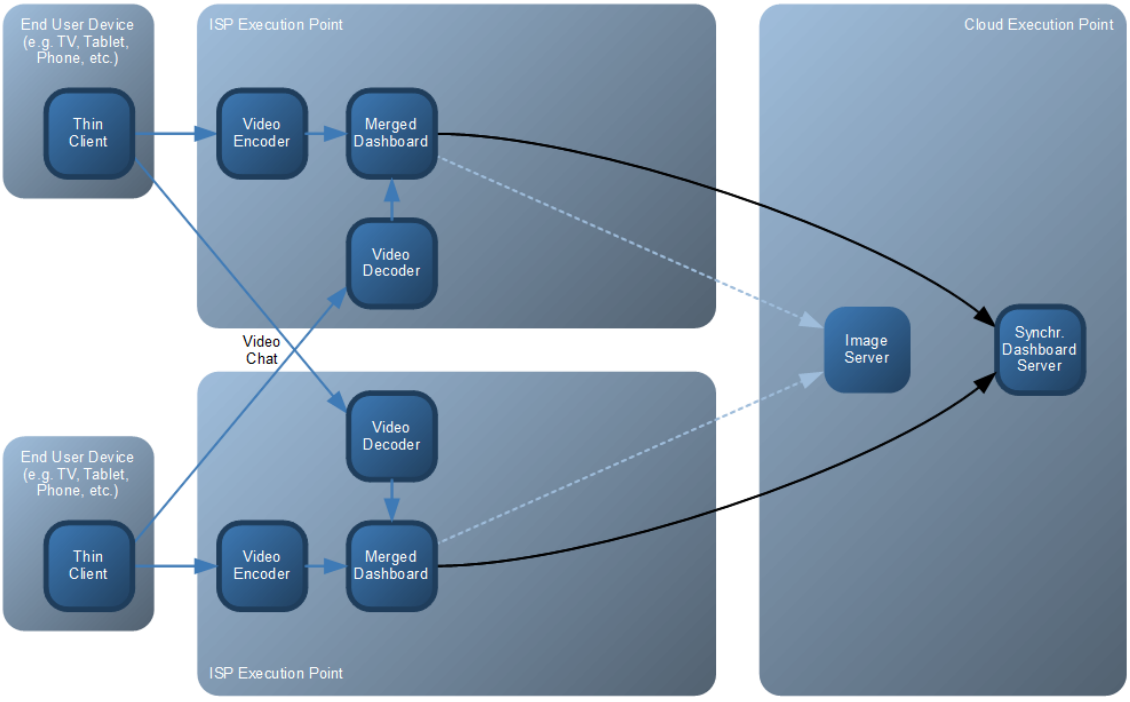
Common or Synchronised Client-Based VoD Dashboard



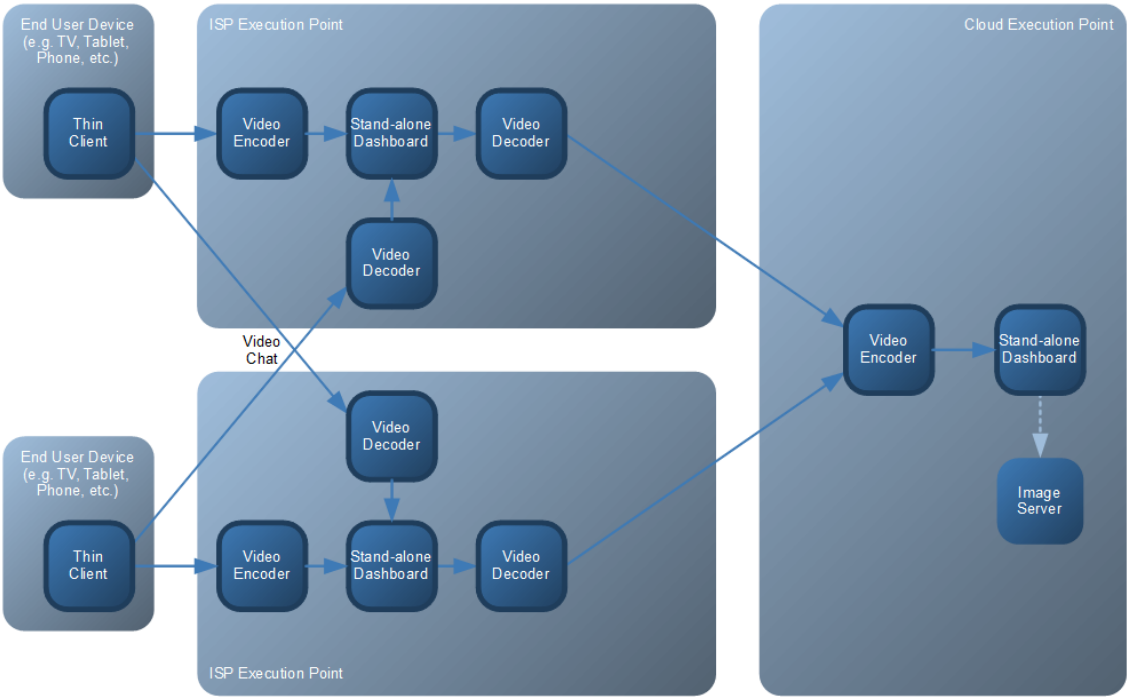
Synchronised Image Collection Dashboard



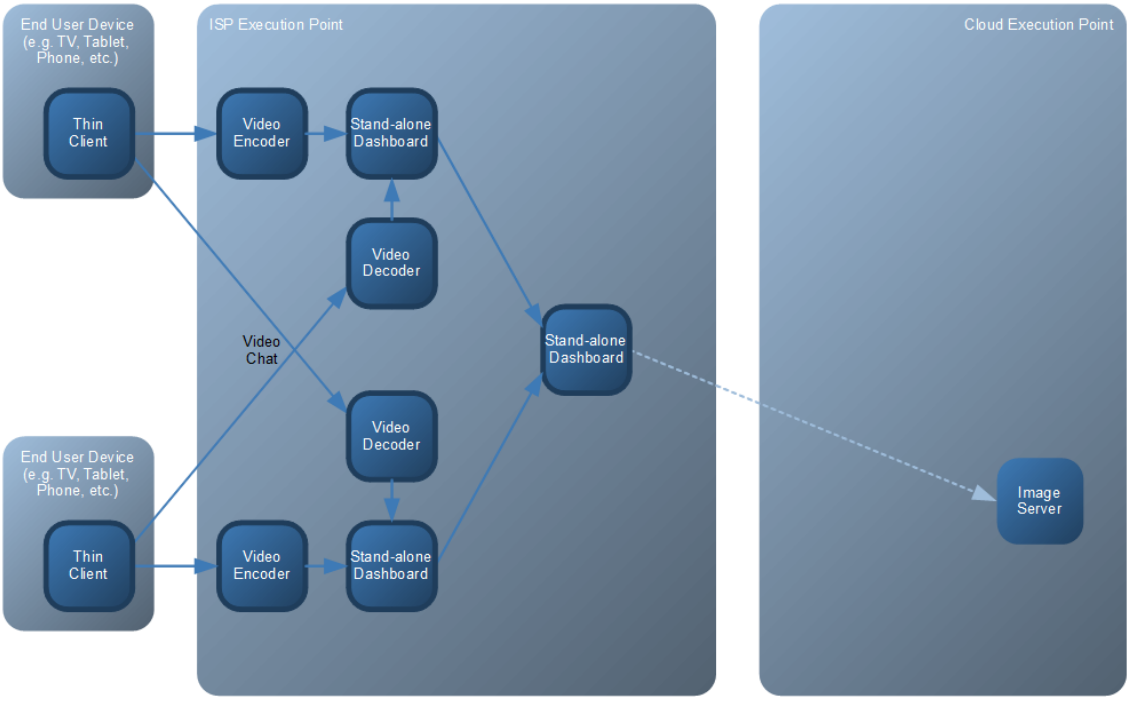
Merged Synchronised Image Collection Dashboard



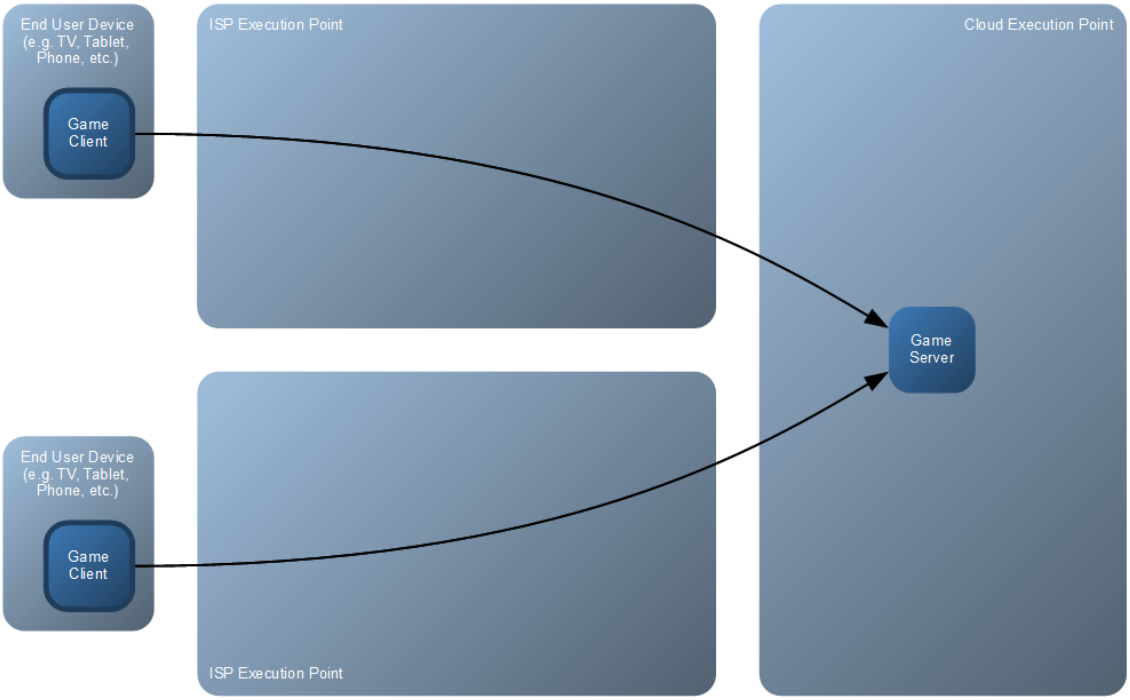
Common Image Collection Dashboard



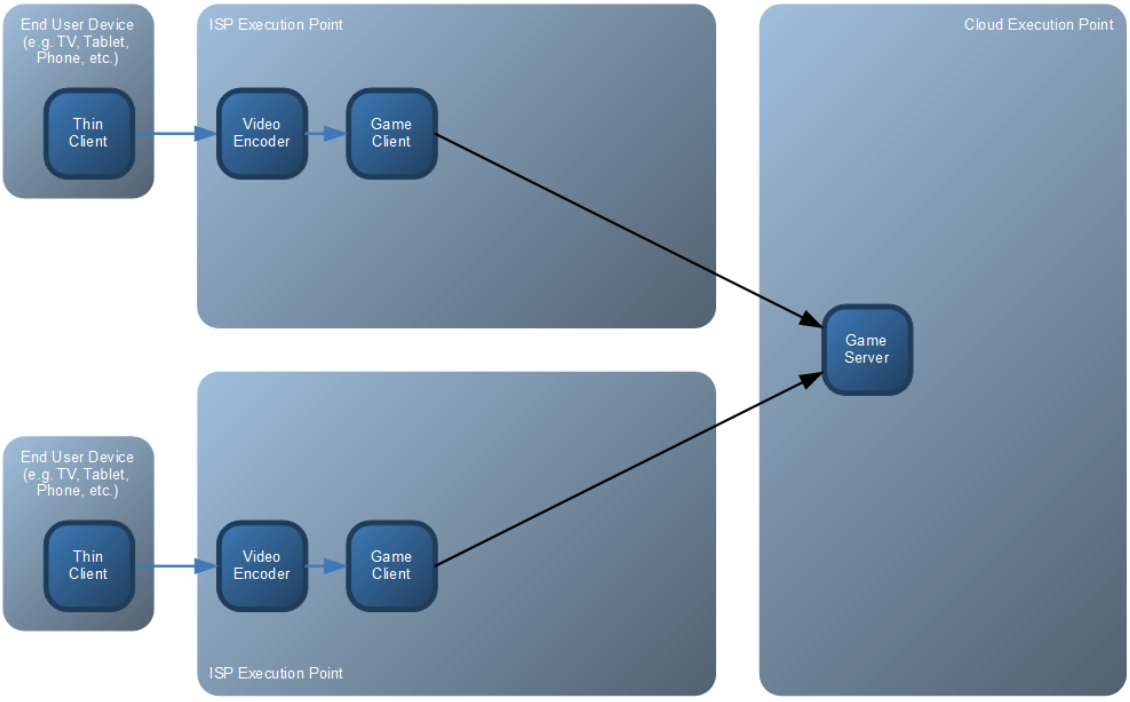
Common Image Collection Dashboard on same ISP



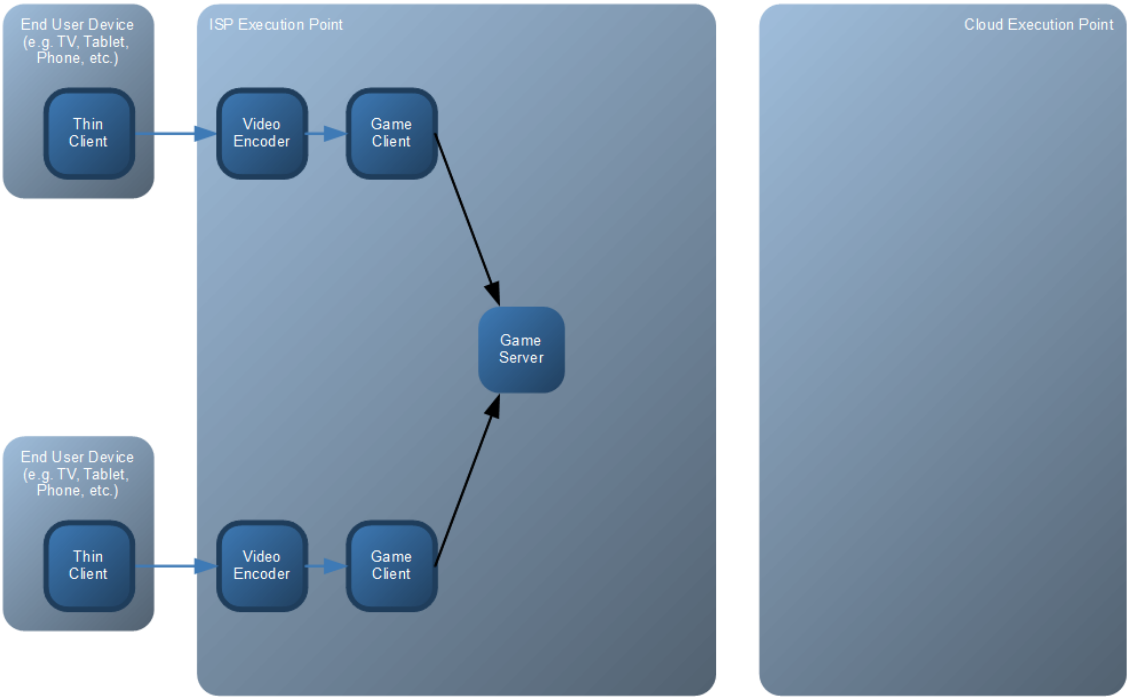
Client-Based Multi-Player Game



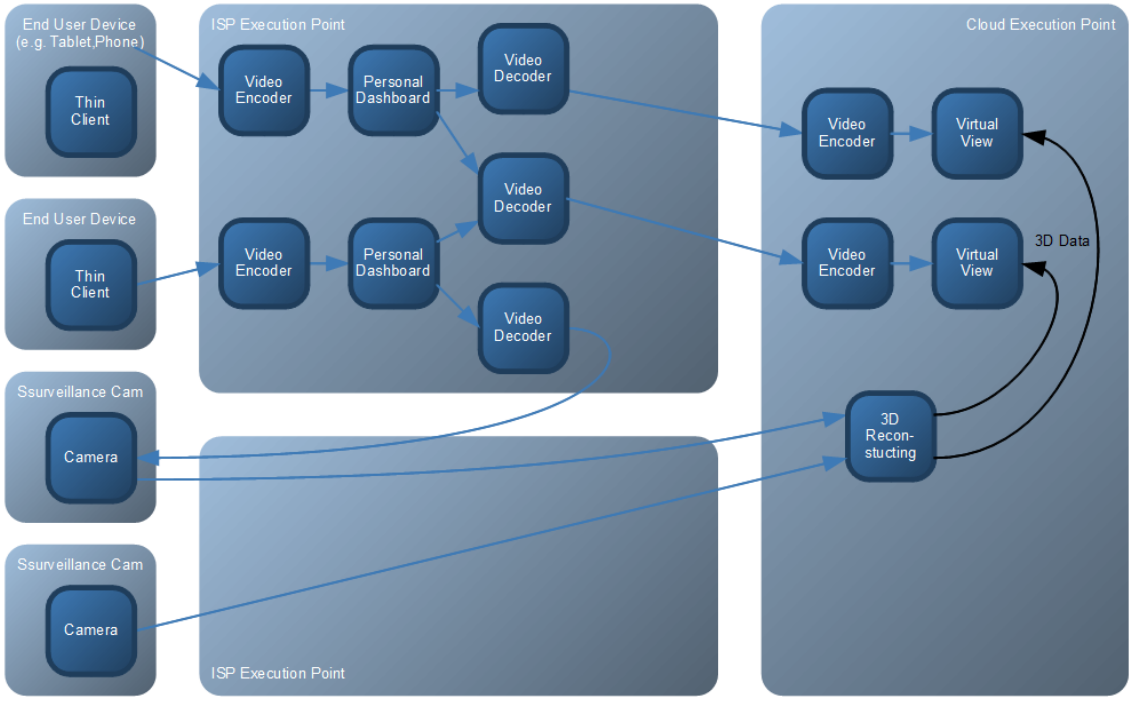
Multi-Player Cloud Gaming



Multi-Player Cloud Gaming on same ISP



First Responder



13. APPENDIX C: DETAILED MEDIA DASHBOARD USE CASE

The Media Dashboard use case is an example of an advanced 2D or 3D user interface that is rendered in the network as a FUSION service and that represents a next-generation electronic program guide. Next to the basic functions you typically expect from a Media Dashboard, like changing the channel, browsing through the TV guide, etc., it can also be a portal to other types of content, including personal videos and pictures, games, or other interactive FUSION applications.

Technically such a media consumption scenario may be implemented by a combination of multiple service software components, for example like the following. Of course, depending on the situation not all service components are necessary, or additional service components are used.

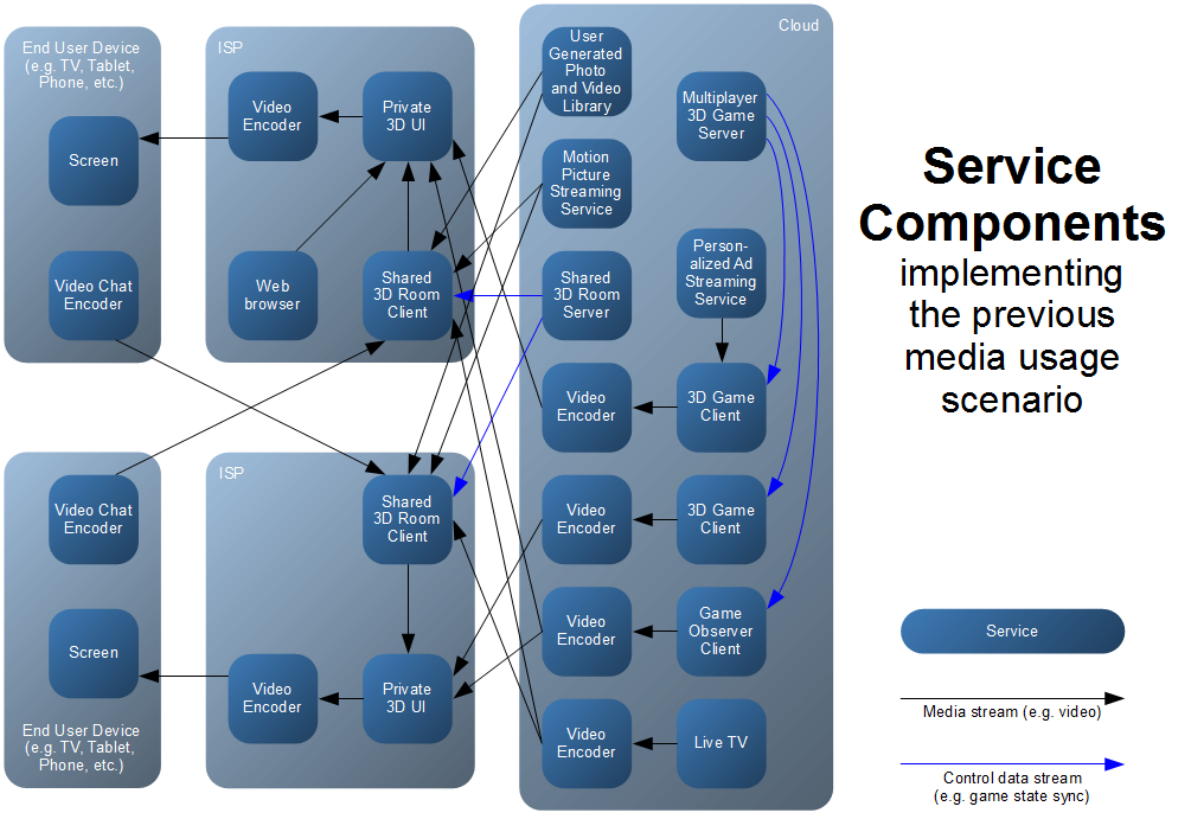


Figure 44: Media Dashboard Service Components

This diagram contains different possible service components:

- A private 3D UI contains the dashboard for one particular user, for example a 2D or 3D desktop for per-user functionality. For example, the top user in the diagram may play a multi-user video game in this private 3D UI.
- In this particular scenario the private 3D UI is rendered not on the end-user device (for example because it is a mobile or thin TV device which is not powerful enough). Instead the 3D UI is rendered in a service software component which is running not on a cloud server, but on a ISP server which is closer to the user and therefore providing more responsiveness. The user may transfer the 3D UI seamlessly from one end-user device to another, for example from his home TV to his mobile device because he has to leave his home.
- Transferring the video stream from the private 3D UI service to the end-user device requires encoding and decoding the video stream, which may not be statically integrated into the particular software, but implemented as separate services. In the above sample diagram the video encoder services are demonstrated.

- A shared 3D room client is a dashboard which is shared between multiple persons. This is for example the place where two persons can browse a photo library together, browse available VoD series episodes, or have a shared image viewer and VoD selections.
- The cloud hosts various service components for media consumption, e.g. VoD servers, image libraries or game servers, which transfer or stream their data to the dashboard service components.

Client and Backend Interactions

In Figure 45, the sequence diagram is shown of a client connecting to a running EPG service instance. Solid arrow lines represent message-based communication, whereas dotted arrow lines represent streaming communication; similarly, black lines represent in-band FUSION communication, whereas blue lines represent out-of-band non-FUSION communication. As can be seen from the diagram, we modelled the video feeds to be streamed outside of FUSION, though this may only be required for legacy clients using legacy video streaming formats.

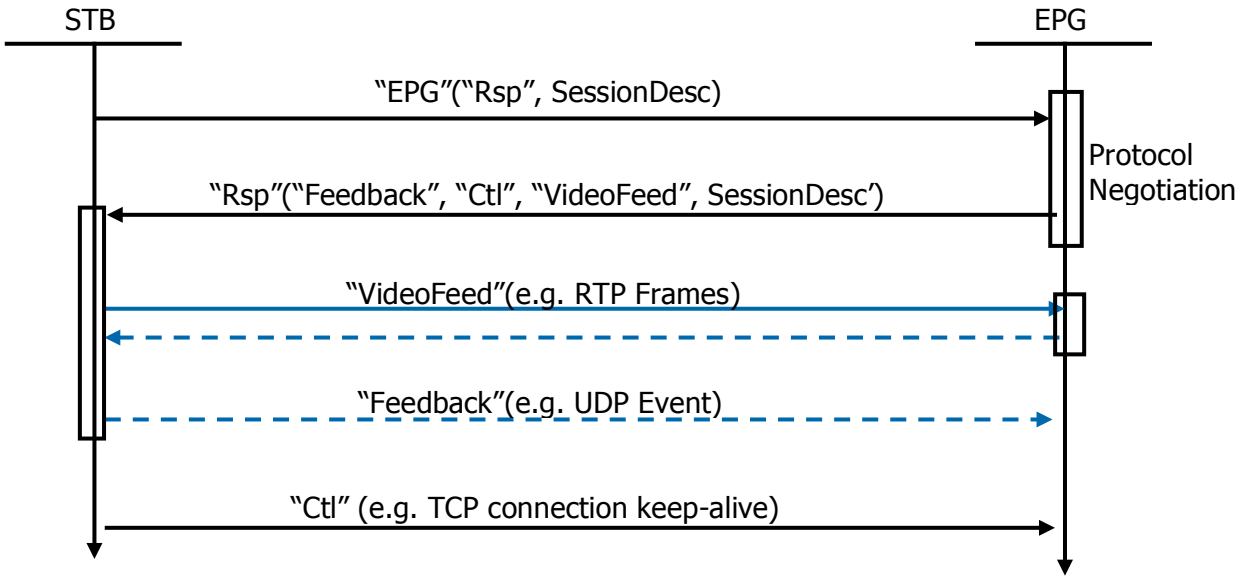


Figure 45: Client connecting to an EPG Service Instance

Basically, the client first issues a service request for the service with the name "EPG", along with a number of FUSION and non-FUSION service request parameters, describing the context of the session. These parameters may include the response channel over which to send back the result, the resolution and quality, an identification of the end user, etc. As a result, the (by FUSION) selected service instance will create and prepare a temporary session to handle the service request, and send back a response to the client. This response may include how and where to contact the EPG session, including the video feed and feedback channel. There may also be a control channel that acts as a keep-alive so that the EPG service session can detect when the client disconnects (abruptly), after which it may decide to terminate the session.

The service instance itself will typically also communicate with a number of input sources that provide the main content for the EPG service. This may include both FUSION sources as well as external sources, and both static and interactive content. In the latter, the EPG service may have to forward the feedback coming from the client to the other service, so that the client appears to be directly interacting with the other service (e.g., a game client). This is backend service diagram is depicted in Figure [ref].

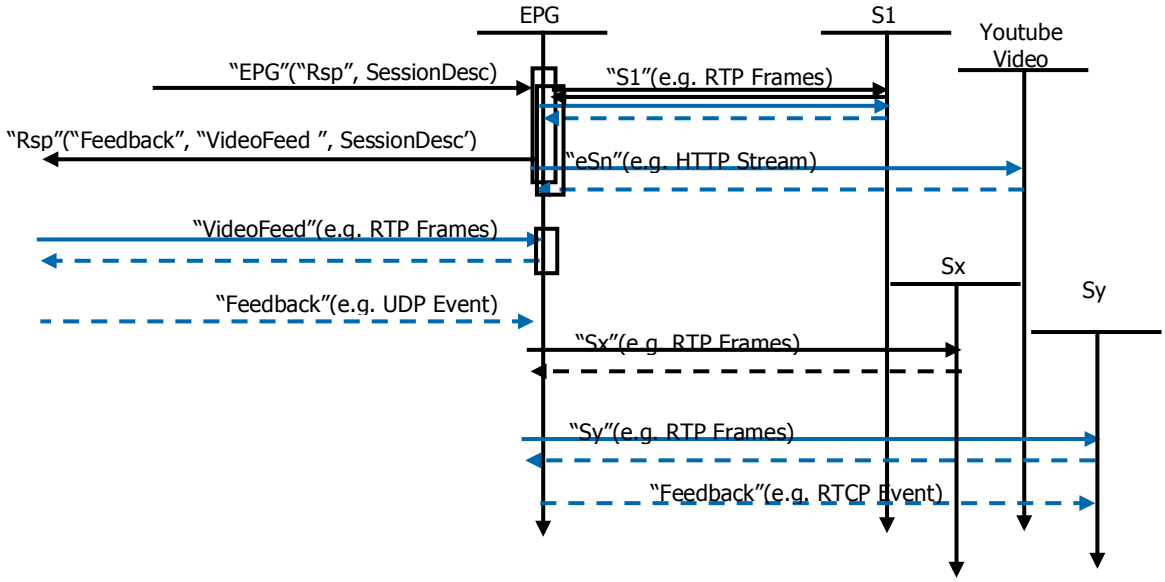


Figure 46: Backend Resources for EPG Service Instance