Future Service Oriented Networks
www.fusion-project.eu

Deliverable D4.2

Updated specification of service-centric routing protocols, forwarding & service-localisation algorithms.

Public report, Version 1.0, 19 December 2014

Authors

UCL	David Griffin, Miguel Rio, Khoa Phan, Vasilis Giotsas
ALUB	Frederik Vandeputte, Luc Vermoesen
TPSA	Dariusz Bursztynowski
SPINOR	Folker Schamel, Michael Franke
IMINDS	Piet Smet, Lander Van Herzeele, Pieter Simoens

Abstract In this deliverable, we provide an update on the specification, design and implementation of the service resolution layer in FUSION. We first describe the architectural and policy aspects of intra-domain and multi-domain resolution. Next, we discuss the design of the different interfaces of the service resolver. In the last part of this deliverable, we report on the progress in the design of various service resolution algorithms, and present an overview of the current implementation status of the key components in the FUSION service resolution layer prototype.

Keywords FUSION, service-aware networking, interfaces, algorithms, prototype.

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

Revision history

Date	Editor	Status	Version	Changes
10/09/14	P. Simoens	TOC	0.1	Initial TOC
23/09/14	P. Simoens	TOC	0.2	Updated TOC + responsibilities
03/10/14	P. Simoens	draft	0.3	bullet point input merged
31/10/14	P. Simoens	draft	0.4	first input merged
24/11/14	P. Simoens	draft	0.5	first consolidated version
29/11/14	P. Simoens	draft	0.6	Update
5/12/14	P. Simoens	draft	0.7	Update
10/12/14	P. Simoens	draft	0.8	Update after consortium meeting
14/12/14	P. Simoens	draft	0.9	UCL input merged
15/12/14	P. Simoens	draft	0.10	Orange input merged
18/12/14	P. Simoens	release candidate	0.11	Editorial changes
19/12/14	P. Simoens	stable	1.0	

EXECUTIVE SUMMARY

This document is a deliverable of the "Future Service-Oriented Networks" (FUSION) FP7 project. It describes the functional requirements and interfaces of the service resolution layer in the FUSION architecture. The key challenge for this layer is the resolution and load-balancing of service requests to the best instance, given the existence of many different potential execution points and dynamically instantiated service instances with varying load. We refer the reader to D2.2 for an overall view on the architecture and the interfaces, and to D3.2 for a detailed discussion on service orchestration aspects.

The service resolution layer is conceived as a collection of peering resolution domains, akin to the Autonomous Systems (AS) in the Internet. In this deliverable, we detail the interfaces and functional aspects of both intra-domain and inter-domain resolution. We study the roles and objectives of the different entities and how the protocols of the service resolution layer must be shaped to support policy enforcement.

This intra- and inter-domain analysis results in a functional decomposition of the service resolver and a description of its interfaces. In the second part of this deliverable, we elaborate on the different messages that are needed for each of those interfaces. The syntax of all service resolution messages has been worked out, resulting in the novel Networked Services Resolution Protocol (NSRP) that is presented in this deliverable. NSRP supports service anycast for resilience and load balancing, direct invocation with edge filtered access for security, support for any name space and more complex query results to enable several types of applications. The syntax details are presented in the appendix of this deliverable.

We also present an initial study on service resolution for composite services. Depending on how much knowledge on the composite service structure is available in the resolution layer, a more optimal replica selection can be realized. This comes at the price of more complexity. In this deliverable, we only report exploratory work on this topic, which serves as input to identify the need for architectural re-evaluation and extension to support composite services, planned in year 3.

The algorithmic work has been progressing as well in the second year of the project. In this deliverable, we present two dynamic selection algorithms for mapping user requests to service instances. The first algorithm is targeting service resolution in a single domain. Upon request arrival, it constructs service availability graphs based on session slot advertisements and finds the shortest available network path. The second algorithm focuses on multi-domain resolution. The algorithm uses a Linear Programming approach to solve a multi-objective optimization problem, trading off user utility against network transit cost.

Lastly, we report on the implementation status of the service resolution layer components, as well as the simulation and emulation environment in which these components will be evaluated.

TABLE OF CONTENTS

1. INTRODUCTION

The main primitive of the FUSION service resolution layer is to resolve service requests to the locator(s) of (the) most appropriate instance(s) among many replicas of the service component running in execution zones throughout the Internet. This fine-grained resolution capability is performed on the grounds of network metrics, service performance metrics and server load.

As motivated in D4.1, the FUSION consortium makes the explicit choice to build the service resolution layer as an overlay network on the existing IPv4/IPv6 network stratum. The main primitive of this layer is the resolution of service requests from service ID to the transport network locator of the most suitable service replica. Clients subsequently establish a network connection via service-specific means, beyond the control of the FUSION service resolution layer.

Figure 1 presents a logical decomposition of the service resolution layer. The service resolution protocol combines network and service instance state to build forwarding tables in the service request resolvers. Instead of developing components for the monitoring of network, infrastructure and services, FUSION wants to leverage on the vast array of monitoring tools and frameworks available and only specifies the interface for the *exposition* of this state to the service resolution protocol. As shown, the network and instance state information also serves as input for service placement across execution zones. This logic is part of the orchestrator layer and is further discussed in D3.2.

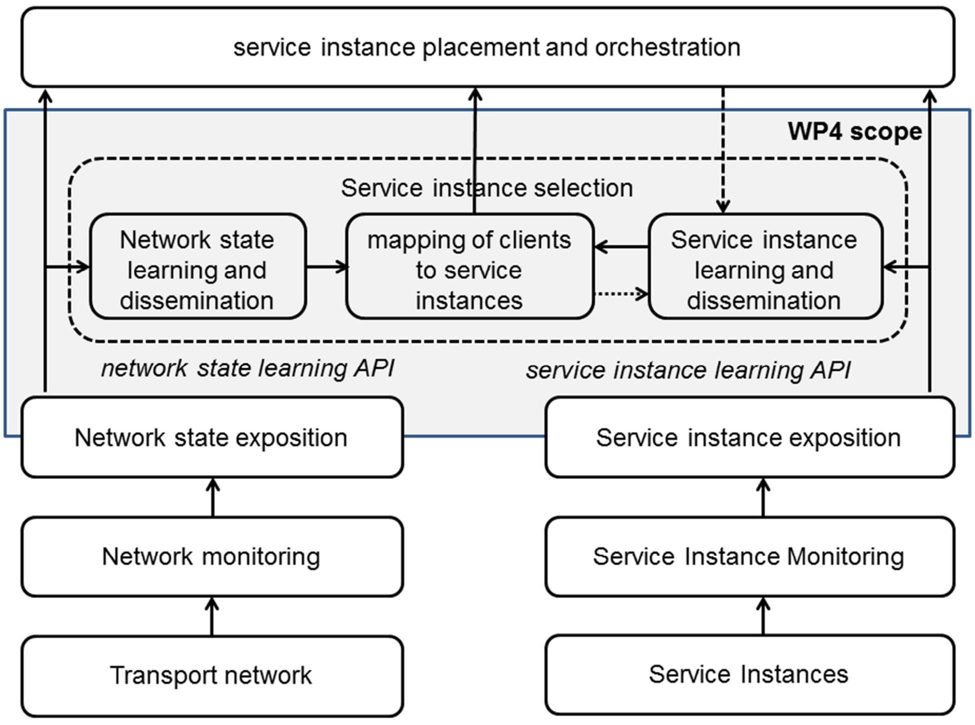
service instance placement and orchestration
WP4 scope
Service instance selection
Network state learning and dissemination
mapping of clients to service instances
Service instance learning and dissemination
network state learning API
service instance learning API
Network state exposition
Service instance exposition
Network monitoring
Service Instance Monitoring
Transport network
Service Instances

Figure 1: Logical view on service resolution layer

The FUSION service resolution layer is structured in multiple domains. Each service resolution domain comprises one logically centralized service resolver to which the gateway of attached execution zones reports service instance availability and state information.

The FUSION multi-domain service resolution is illustrated in Figure 2. Execution zones are grouped in several domains, according to their IP network attachment point. The service resolver has a detailed view on the underlying transport network that interconnects its execution zones, and the service availability in execution zones attached to that part of the IP network. Service instance state is exchanged between the different service resolvers, possibly subject to service resolution policies, so that each domain can build its own *catalogue* of service instances worldwide. Clients direct their service request to their service resolver. This resolver will first try to resolve this request to an instance running in one of the local execution zones. If no suitable instance is available, and if allowed by the service QoS requirements and deployment policies, the request will be resolved to an instance running in another domain.

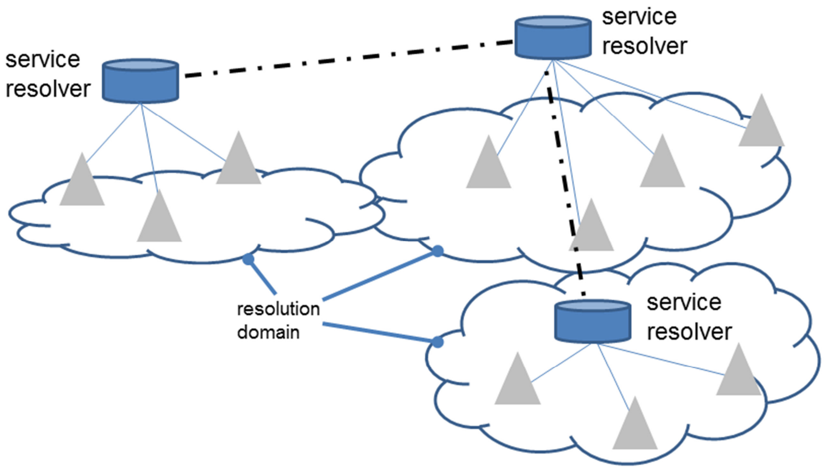

Figure 2: Service resolver collects monitoring information from instances running in execution zones in its domain and exchange this with other domains.

Users send a request to the service resolver of their domain. Requests that cannot be resolved in the domain, either because the service is not deployed or available, it can be resolved to the locator of an instance hosted in an execution zone of another domain. The resolution can be done either in the domain from where the request originates, either by forwarding the request to a service resolver in another domain. This multi-domain architecture enables advanced service resolution policies imposed by different stakeholders, as detailed in section 3.

As update of D4.1, this deliverable focuses on the service resolution layer of the FUSION architecture, including the design and specification of interfaces as well as the implementation of its constituting components (the service resolver and the zone gateway[1]). Its main contributions are:

- a functional description of service resolution in a single domain. This involves the communication and interaction between service instances, the zone gateway and their service resolvers
- a functional description of policy-based multi-domain service resolution
- a logical decomposition of the service resolver and discussion of each interface
- a detailed specification of the Networked Services Resolution Protocol that runs between clients and service resolvers, and between service resolvers. The exact syntax is presented in the appendix of this deliverable.
- progress report on the design, implementation and evaluation of different service resolution algorithms that have been developed
- status report on the prototype for the service resolution layer that will be built in the project

This deliverable is structured as follows. We first present a functional discussion on intra- and inter-domain service resolution. Then, we zoom in on each interface of the service resolution layer. In the last part, we present experimental results on service resolution algorithm and an update on the implementation status of the prototype of the service resolution layer.

[1] We refer the reader to deliverables D2.1 and D2.2 for an overall view on the architecture and the interfaces.

2. INTRA-DOMAIN SERVICE RESOLUTION

In this section, we elaborate on the internals of a service resolution domain. Multi-domain aspects are discussed in the next session.

2.1.1 Scope of a service resolution domain

A service resolution domain is defined as the collection of one (logically centralized) service resolver and the execution zones that directly report via their zone gateway service instance statistics to this service resolver. The resolver governs the resolution to instances hosted in one of its execution zones, although for scalability reasons it can allow service resolvers in other domains to handle requests to execution zones in its own domain.

Each service resolver builds its own database with network and service state information. While service state information is advertised by the zone gateway component of execution zones, network monitoring information can be collected via various means: queried from other service resolvers or via existing monitoring tools. Given the vast number of networking monitoring tools available, FUSION does not mandate a particular network monitoring framework. Instead, to accommodate as many deployment models as possible, we only define in a functional way the interface for network state exposition.

However, the FUSION consortium is studying one particular model where a service resolution domain coincides with the transport network of an ISP, advertised as a unique Autonomous System Number (ASN) in BGP. The consortium considers this option as the most realistic deployment scenario for FUSION service resolution, with each ISP operating one logical service resolver that peers with service resolvers of other ISPs. In this scenario, the service resolver has direct access to the network management and monitoring of the underlying transport network that interconnects the different zone gateways.

In light of the current state-of-the-art, we expect each IP router to be a potential execution zone that can run a limited number of service instances. For example, the Cisco IOX platform is enabling access to computing and storage resources within the network devices to host applications and interfaces as close to the network edge as possible [CISCO-IOX]. To have an initial expectation on the typical size of a service resolution domain, we crawled the CAIDA IP-router topology that is continuously collected and updated by a set of globally distributed probes [CAIDA]. This dataset contains the detected routers with their assumed geolocation. Table 1 shows the number of routers in this dataset for three European countries. Given the limitations of the collection process and the geocoding of IP addresses, the databases contains multiple routers with the same geolocation. In the third column of the table below, we provide the number of unique router locations in the database.

Table 1 – Number of IP routers per country, as crawled from CAIDA database

Country	No. of nodes	No. of nodes with unique geographic coordinates
Germany	1 294 539	7 034
France	1 804 905	10 696
Poland	228 893	1 859

We are well aware that this table only provides a rough estimation of the size of a resolution domain. On the one hand, the figures presented above are an overestimation since multiple ISPs/ASes are active in the same country, the dataset may contain false positives and the applied geolocation might be inaccurate. Also, ISPs can adopt individual rules as to where execution zones will be placed in their network. On the other hand, we expect the number of execution points to increase. For example, Nokia Siemens Networks has commercialized its Radio Access Cloud Server [NOKIA-LA], enabling to deploy services on eNodeB base stations.

2.1.2 Zone Gateway

Four stakeholders are involved before the instances of a service are announced to the service resolution: the service provider, the orchestrator, the zone gateway (as part of the zone manager) and the service resolver. In this deliverable, we focus on the interaction between the latter two: zone manager and service resolver.

2.1.2.1 Intermediate between service resolver and service instance

To allow clients to discover and connect to a particular service instance, the endpoint of that instance as well as instance state (e.g. load) are advertised by the zone gateway to its service resolver. This endpoint is the tuple (IP, port) that the service resolver will answer to a service request. Service resolvers can then advertise this information, possibly in aggregated or obfuscated form, to resolvers in other domains.

Rather than having individual instances report directly to a service resolver, the FUSION consortium makes the explicit choice to have the zone gateway in control of this process. This is motivated as follows:

- **Zone load balancing.** Load balancing across multiple instances is an essential component in any cloud infrastructure today. By exposing a single public IP, all traffic for service replicas is routed through this load balancer, allowing for elasticity through up- and downscaling. Moreover, common load-balancers perform additional zone orchestration tasks like detecting unhealthy instances, auto-scaling and SSL termination. Since instances will be provided with a private IP address, it is not possible to have service instances retrieve their IP and port via introspection and announce it themselves to the service resolution domain[2].
- **Zone orchestration policies**. The zone might be operated by a different entity than the service resolution domain and may want to hide the number of instances actually deployed, e.g. not to expose internal scaling policies. Moreover, by keeping the zone manager in charge of the instance state advertisement to the service resolver, its orchestration capabilities are enhanced. For example, the zone could apply an on-demand instantiation policy and advertise service availability although no actual instances are deployed until the time the service is actually invoked.
- **Resolver layer scalability**. Having each individual service instance injecting state into the service resolution layer would cause a large amount of traffic. The zone gateway can aggregate state information across all its instances and send a summary to the service resolver.

The zone gateway reports two types of information to the service resolvers:

- **Utility function**: this is a metric that is service-specific and reflects the user QoE in terms of network performance metrics. Higher values will result in more requests being resolved to that instance.
- **Session slots:** to accommodate for the large heterogeneity of service performance metrics, the FUSION consortium has decided to adopt the generic metric of session slots (as introduced in D3.2). Session slots reflect the total number of concurrent sessions a particular instance can handle.

[2] Some cloud providers, targeting the most demanding services, provide instances with a public interface that is directly connected to the Internet [SOFTLAYER].

2.1.2.2 Instance state aggregation

Multiple instances of the same service can be running in a single zone. Each instance will report monitoring information to the zone manager. Although in principle the individual instances can be exposed to the service resolution layer, in most cases it can be expected that only one public endpoint (e.g. IP/port of load balancer) is advertised, as this provides more flexibility for instance scaling. Consequently, it is the responsibility of the zone gateway to process the individual instance load information to one aggregated number of available session slots. The aggregation function to be applied is determined by zone orchestration policies and by the service provider, since the number of available session slots is not always the simple sum of the available slots advertised by each running instance. In practice, the zone gateway must account for the slots that can be provided by the maximum number of instances that could additionally be deployed. In case of composite services, the zone gateway needs to account for the sharing of (resource) session slots between multiple services. This is further explained in D3.2.

2.1.2.3 Instance state reporting

Session slots and utility functions only reflect a snapshot of monitoring information that will be rapidly out-of-date. To improve the resolution capabilities, we therefore mandate that zone gateways also advertise a histogram of service session times. This will allow the service resolution plane to forecast the evolution of session slots.

2.1.2.4 Trust relationship between zone gateway and service resolver

As explained in the previous section, for reasons of scalability and policy enforcement, the FUSION consortium has decided that the service resolver must receive service availability information from the zone manager and not directly from the service instances. The corollary of this decision is that the service resolver cannot assume a trust relationship with execution zones that are owned by a different party.

We have identified three different security threats that may impact the service resolution:

Security threat: fake service availability advertisement

What?

Execution zones must attest that they have obtained the binary of a particular service, otherwise malicious execution zones could advertise false service advertisements, leading to denial-of-service for the users as they are resolved to non-existing instances[3].

Mitigation?

One possibility is to only accept service registration from a predefined list of zone gateways. However, this provides no absolute protection against hijacked zone gateways.

Some ICN approaches use self-certifying names, where the receiver can verify the integrity of the downloaded content. However, as FUSION targets remotely running services instead of downloadable content, self-certifying names are not sufficient as they do not guarantee authentication.

One promising approach is to apply the Packet Level Authentication mechanism [PLA11]. The sender adds an own header to packets, containing the sender's cryptographic identity, certificate from the trusted third party, signature over the packet and other fields. Using this information, other nodes can verify the integrity and authenticity of the traffic origin. This mechanism can equally be used for

[3] This would lead to service unavailable errors (cf. 404 error code in HTTP). There is also the threat that clients receive the IP/port of a malicious instance. However, clients can always attest the authenticity of the instance they connect to using existing certificate-based security mechanisms like SSL/TSL.

signalling messages. Applied to FUSION, service resolvers could use PLA to challenge execution zones by sending a challenge to the public locator that is exposed by the execution. The execution zone must then respond with an authentication packet that has the challenged signed by the service at hand. Because such packets can only be generated by legitimate instances, the zone gateway proofs that it executes the running service by forwarding this packet unmodified to the service resolver.

Security threat: SLA violation

What?

Zone gateways have interest in announcing better service performance statistics (e.g. more session slots, better utility), as this will increase their chance of requests being resolved to an instance hosted in their execution zone.

Mitigation?

In contrast to the previous security threat on fake service advertisement, signing the report messages by the instance is not an option, as the zone gateway is responsible for aggregation and advertisement to the service resolver.

If clients report back on their RTT to the service resolver, these figures can be compared with the utility function and the network metrics that have been collected by the service resolver. If large and/or repeating anomalies occur that cannot be attributed solely to the status of the transport network, the service resolution plane could penalize such execution zones and resolve clients to other execution zones.

3. MULTI-DOMAIN SERVICE RESOLUTION

In this section, we identify the need for policy enforcement when the service resolution layer is composed of multiple resolution domains. Policies can also be enforced by other entities involved, like the zone manager, the orchestrator or the service provider. Based on study, we are able to provide the basic operational principles and protocol requirements for multi-domain resolution. This section discuss the multi-domain resolution from a more general perspective. More details concerning multi-domain service resolution are discussed in section 6.2.

3.1 The use of policies in service resolution

Throughout this section, a *policy* is understood to be a rule that governs the selection of a service instance. As will be specified later, such rules may apply directly to service requests or to the information exchange that takes place between service resolvers to build their own forwarding tables (we thus differentiate between resolution which operates on per-request basis from the construction of forwarding and resolution tables which operates at different time-scale and is not directly related to particular service requests).

In the following we discuss the main motivations that the different actors may have to enforce their policies and how these motivations impact the service resolution. Based on this we define top-level requirements for policy enforcement mechanisms applicable at the protocol level of the FUSION service resolution plane.

3.1.1 Aspects

In general, players that are involved in providing different functions defined by the FUSION architecture may be interested in having a degree of control over the process of service resolution. Their objectives and the desired degree of control may depend on many factors.

3.1.1.1 Roles

We first identify the different roles in the FUSION resolution. These roles will drive many motivations, important requirements and respective protocol-level mechanisms.

The main roles that can influence the inter-domain service resolution are listed below. We also provide exemplary motivations each role may have to enforce service resolution policies.

Originating service resolution domain – service resolution domain that is the home domain for a service requestor (client or service instance). Exemplary underpinning principle for a policy that this role may have is network traffic engineering and/or the preference to resolve requests to instances hosted in other domains to save data centre resources. Apart from defining own policies, an originating service resolution domain may have incentives to act on behalf of other roles (e.g., domain orchestrator) and achieve some objectives thus contributing to the overall fulfilment of these external policies. Exemplary motivations for the latter are:

- benefits from having agreements with clients, service providers and domain orchestrator, or remote (e.g., trusted) service resolution domains to provide adequate network performance statistics of the scope of its domain,
- benefits from having agreements with clients or domain orchestrator/service provider to provide service resolution based on end-to-end network performance

The above motivations are not exhaustive and their relative value will depend on the domain in question.

Terminating service resolution domain – service resolution domain having control over a subset of execution zones where instances of requested service are hosted. Exemplary underpinning principles in this case are similar (after appropriate adaptations) to those listed above for the originating service resolution domain. Nevertheless, it is possible that business models specific for this role (if any) may also result in incentives that will be unique to it.

Intermediate service resolution domain – service resolution domain that forwards service requests between the originating and terminating domains. The main reason for the introduction of such domains is to support aggregation of resolution information to improve the scalability of the resolution layer. Another reason may be to enable network-based selection in presence of transit network domains that otherwise would not be covered by the originating and/or transiting domain. Intermediate domains will mainly enforce policies requested (injected) by other domains to improve the quality of the selection process due to their enhanced view of network state. A closer study on this role, the validity of respective incentives and potential impact on service resolution mechanisms and also on the FUSION architecture is left for the next period of the project (note that aggregation resolvers are mentioned also in section 6.2).

Orchestrator and service providers. Service providers are expected to define a lot of policies and many of them can refer to service instance selection process, for example regarding service access rights based on user attributes. According to the FUSION model, service providers are positioned outside the service selection/resolution loop and are supposed to delegate policy enforcement to the orchestrator. In the following we thus consider orchestrators as aggregating points that enforce the policies on behalf of respective service providers. The following may be additional motivations for an orchestrator to participate in the resolution process:

- enforcement of specific/complex policies difficult to achieve otherwise, e.g. in a distributed way at the service resolution level
- hiding confidential information that is needed for policy enforcement.

We note that each party involved in the resolution process for a given request may wish to enforce its own policy for this request. In section 3.1.2 we generalise this to formulate corresponding requirements for inter-domain service resolution and in section 3.2 we propose a set of mechanisms that fulfil these requirements.

3.1.1.2 Objectives

We assume that policies, including their measurable objectives, are typically private in the sense that they need not be communicated explicitly to other parties (roles). Nevertheless, parties undertaking particular roles may have to rely on other parties to assure that the measurable objectives of their policy are met. To this end a party may decide to expose its expectations in the form of *requirements* and delegate the enforcement of these requirements to other party (parties). We refer to such requirements as publicly measurable objectives. An example of this mode of operation is when the orchestrator attempts to assure a certain level of QoS (e.g., maximum delay, minimum bandwidth, etc.) for a given type of service and to achieve this, the service resolution domain notifies the network control/management plane to reserve network resources necessary to satisfy the demand.

Conversely, non-public (or private) measurable objectives are not communicated *per se*, but instead parameters related to the service are exchanged between parties involved in service resolution to define the allowable scope for subsequent decisions. In this case, information describing the *capabilities* of particular domains to be used by other parties is exposed rather than explicit requirements for them to meet. This may include both non-functional characteristics such as network cost/performance or service cost/performance guaranteed/achievable in a given resolution domain, but also functional characteristics like client address/location or a recommended list of execution zones/service instances, etc. In case such capabilities are exchanged during the service request resolution process they are said to define *context* of the request.

It should be stressed that the two generic approaches just mentioned, i.e., requirement exposition *vs* capability exposition, are complementary with respect to each other. It is also important to note that, in the area of network performance provisioning, the FUSION consortium has decided to focus mostly on capability exposition mode of operation.

3.1.2 Basic requirements

Below, we propose top-level requirements related to the use of policies for inter-domain service resolution in FUSION. These requirements are partially derived from the discussion provided in section 3.1.1 above, but are also inspired by useful and quite universal functionalities employed in solutions known from other research domains as content delivery, e.g. [FPLS13] [PDBR14]. These requirements will be used in section 3.2 to identify a set of protocol-level mechanisms that enable policy-based service resolution in the FUSION architecture.

R1: *For each party (or role as of section 3.1.1.1), there must exist a possibility to enforce its own policies that have impact on the results of service request resolution.*

In principle, there are two complementary options to achieve this goal. One option is for a party to rely on other party(ies) and delegate the enforcement of desired policies to such external partner(s) (e.g., this could be done via dedicated interfaces using agreed notation to describe the rules to be implemented). The other option is for a party to get itself involved in the resolution process and enforce desired policies on its own.

R2: *Each party must have a possibility to protect confidential data (i.e., not to be shared with other parties) when enforcing their desired policies.*

One way to fulfil this requirement is to directly involve the interested party in the resolution process so that it can enforce policies on its own. In technical terms, this means that such a party should operate as a service resolver. This gives a clear indication of the mechanisms that should be available to the orchestrator.

R3: *There should be some degree of support for composite services.*

We note this requirement applies to intra-domain resolution as well, and that composite services define a complex case where policy enforcement may often require the use of centralised logic.

At present, we have identified two options of how service resolution of composite services can be handled in the orchestrator layer or in the service resolution layer:

- finding the appropriate orchestrator module that is aware of the composite service graph structure: such a module would work as a terminating service resolution with enhanced logic for orchestrating composite services. The main role of the service resolution in this case is to find this orchestrator module, which acts as an oracle, pass to it the parameters needed and deliver the oracle's response back to the requestor.
- finding a set of atomic service components under a given set of (service-dependent) constraints by the service routing layer itself. This requires that the service graph structure is injected in the composite service routing layers.

Section 9 contains an initial discussion on the complexity of resolution for composite services in both options. A deeper study of the role of service resolution in supporting composite services is planned for the next reporting period in the project.

3.2 Protocol-level mechanisms

We are interested in the mechanisms that the FUSION service resolution layer must provide to support different players in their policy enforcement tasks. In general, two types of such mechanisms can be distinguished on protocol level:

- out-of-band mechanisms apply to the information exchange between parties/domains that takes place outside of the service request resolution process. This information is disseminated independently from the execution of service request resolution tasks and allows service resolvers to build their own forwarding and resolution table that subsequently is used by them on per-request basis for the resolution purposes,

- in-band mechanisms (per-request mechanisms) are used during the phase of resolving individual requests sent by clients and are built into the service request resolution protocol. The information exchanged to resolve a given request can be reused for subsequent requests, too (similar to the caching of DNS responses by DNS resolvers).

In an ideal case the set of related protocol-level mechanisms should be limited. For reasons of scalability, this applies in particular to in-band mechanisms.

3.2.1 Out-of-band enforcement

Out-of-band enforcement mechanisms will typically consist in filtering the information to be exposed in the resolution protocol to neighbouring domains. For this purpose appropriate rules can be set in service resolvers to control the level of detail of the information exchanged. In such a case, a dedicated API can be used to allow external function (orchestrator and/or service resolution domain provider) to deploy desired policies in service resolvers. Although the details of such an API are out of scope of this report, it is nevertheless expected that most often they will have no impact on the routing protocol. Altogether, no specific protocol-level mechanisms will be required to deploy a wide variety of policies for off-path enforcement procedures. We note also that off-path enforcement as considered here supports requirement R1 as applied to the orchestrator and service resolution domain provider.

It may turn out that in case of certain off-path rules some form of standardisation may be needed, e.g. when rules for policy enforcement have to be implemented in a coordinated way across multiple service resolvers (service resolution domains). Such coordination among service resolution nodes may imply the introduction of appropriate protocol mechanisms. However, at current stage of development of FUSION we consider this option to be theoretical and postpone further studies in this area until a real need for the introduction of such coordination is identified.

Advantages: policies are enforced only in service resolvers so some (typically significant) part of the required processing effort can be done outside the fast (request forwarding) path. Also, the service resolution signalling can take shorter (optimised) paths compared to the case where the orchestrator has to be involved in the resolution process.

Disadvantages: depending on the policy, trust concerns may arise when implementing third-party policies; sensitive information may have to be exposed to the service resolution plane which may rise additional trust concerns.

3.2.2 In-band enforcement

In-band policy enforcement assumes that each interested party is involved in the resolution process on a per-request basis. To support this capability it is necessary to provide mechanisms for in-band invocation of policy enforcement parties and for in-band exchange of information to be handled by policy enforcement rules. In the context of FUSION we identify preferred schemes for these two types of functions.

3.2.2.1 In-band invocation mechanisms

In-band invocation allows to trigger an action of a party interested in applying its policies to a given request. We consider two basic mechanisms to enable this functionality, namely resolution hints and orchestrated request forwarding.

Resolution hints mechanism, similarly to routing hints proposed by other frameworks, e.g. [SAIL13] and [NDGK12], uses a dedicated API to allow service resolvers consult the orchestrator in deciding what action to perform regarding a given service request. On one hand, this mechanism potentially provides the orchestrator a great flexibility in enforcing desired policies. On the other hand, it requires the introduction of a dedicated information model that will be shared by the orchestrator and service resolvers. Although this solution may prove to be desirable in the long term, we prefer in the current stage of the project a more lightweight approach as described below. We note that the use of resolution hints allows to meet requirements R1 and R2 as applied to the orchestrator.

Orchestrated request forwarding simply assumes passing service requests to the orchestrator based on predefined criteria. In this case the orchestrator plays a role of a specialised service resolver and depending on the policy to be enforced it either forwards the request to the next-hop service resolver or sends it back to the resolver who triggered orchestrated forwarding. In the latter case, service resolvers that trigger orchestrated forwarding may have to implement additional mechanisms to cope with such requests being sent back by the orchestrator. Thus, the operation of this scheme resembles to the one found in IMS at the interface between S-CSCF and application servers to trigger services [WZKC13]. We note that orchestrated forwarding allows to meet requirements R1 and R2 as applied to the orchestrator (so in many cases it can be equivalent to resolution hints described previously); it may also contribute to fulfilling requirement R3 provided that orchestrator is capable of coordinating composite services (such a model would resemble the one adopted in [NGSO11] for the service composition function).

Advantages of orchestrated forwarding: fulfilment of requirements R1 and R2 regarding the orchestrator role so many trust concerns can be eliminated; ability to implement complex policies.

Disadvantages of orchestrated forwarding: service resolution signalling can take long paths and thus have a negative impact on service resolution delays; adds complexity to service resolvers and to orchestrator.

3.2.2.2 Information exchange

Information exchange involves two interrelated issues to be solved, namely the pattern for passing signalling messages among the parties and the general form of the information to be actuated upon by the parties involved.

The design decision in FUSION is to adopt a simple scheme for the selection of appropriate service instances based on the use of candidate (or ranking) lists that contain execution zone identifiers or service instance identifiers, possibly depending on the type of the message (request, answer). A general idea is to use a candidate list (ranking list) that is passed within the request/answer along the signalling path to negotiate the final choice of service instance among the parties. Given this scheme, the results of policy enforcement decisions are exposed during the resolution phase in the form of current composition of the candidate list attached to the resolution messages. Of course, additional rules for this negotiation may be needed, e.g., the list may not be extendible and at most one instance may be suspected to be chosen as a final result of the resolution. Appropriate conventions applicable to this process will be subject by the FUSION consortium to further studies. We note the use of candidate lists contributes to fulfilling requirement R2.

Advantages: relies on information that by definition is standard for FUSION service resolution so certain trust concerns can be limited; simple form, able to support a wide range of different policies; can be combined with different triggering schemes; helps to avoid the dissemination of excessive amounts of service state information in case it may change very frequently (late binding effect) or in case of low-popularity services.

Drawbacks: may be unable to express certain complex policies; can increase signalling traffic and rise processing requirements due to the use of (potentially long) candidate lists.

3.3 Principles of operation

In this section we present basic principles of the operation of FUSION service resolution layer in the multi-domain scenario. Throughout this presentation, the use of the mechanisms defined in section

3.2 is assumed to enable policy enforcement especially during the resolution phase. Therefore we take the particular perspective of service request resolution to organise the discussion that follows and consider its two following aspects:

- domain of resolution, i.e., the domain where the decision about the selected service instance is taken for a give request
- collection and exchange of information needed for multi-domain service resolution.

Both of the above aspects are discussed in the following subsections. We note that we assume relatively high level of abstraction here as the goal of this section is to provide basic concepts for multi-domain operation of FUSION service resolution layer. Selected interdomain aspects of service state exposition and query resolution are described in more detail in sections 6 and 7, while the description of the protocol adopted by FUSION for resolution purposes is provided in section 8.

3.3.1 Domain of resolution

We found the two simplest (extreme) cases of service resolution scenario to be when the resolution decision is taken either in the domain of the requesting client or in the domain of the best instance of the service being requested. These two cases are illustrated in Figure 3 and Figure 4 respectively.

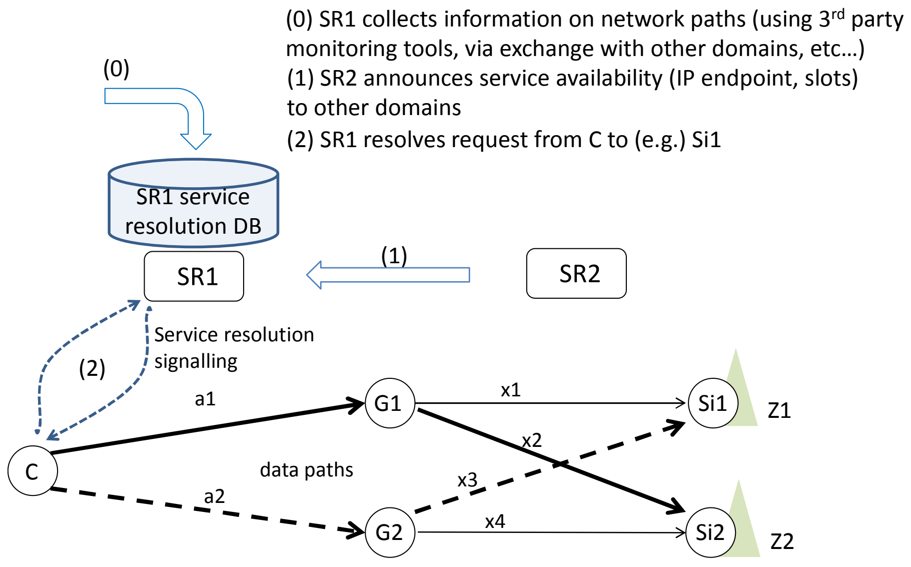
(0) SR1 collects information on network paths (using 3rd party monitoring tools, via exchange with other domains, etc…)
(1) SR2 announces service availability (IP endpoint, slots) to other domains
(2) SR1 resolves request from C to (e.g.) Si1
(0)
SR1 service resolution DB
SR1
(1)
SR2
Service resolution signalling
(2)
a1
G1
x1
Si1
Z1
x2
C
data paths
x3
a2
G2
x4
Si2
Z2

Figure 3: Resolution in client domain.

The main steps of resolution in client domain are shown in Figure 3. It can be seen that in this case the domain that hosts service instances (represented by service resolver SR2) is not involved directly in the resolution process (although domain SR2 disseminates the information on service availability and on session slots and thus indirectly impacts local decisions taken by SR1). It is also to be noted that in this basic scenario network-level information is gathered by resolution domains (SR1 in this case) on their own using all available means provided by the network exposition interface (see also section 5). The overall result is that domain SR1 is able to resolve to the best instance on its own based on the information gathered outside of the resolution process and possibly applying policies that are known locally.

Resolution in best instance domain (see Figure 4) defines the opposite case to the above and it leaves the final decision regarding instance selection to the terminating resolution domain (SR2 in this case). In general, the role of the originating domain SR1 in this scenario is not null as SR1 can decide upon the terminating domain in case when multiple terminating domains are available (not show in the figure). We note that in the latter case the decision taken by SR1 may well be policy-based. Moreover, message forwarding rules of the resolution protocol may allow the response from

SR2 to the client to also be forwarded through SR2. As we will see later in this section, this feature enhances the set of available options for policy-based service request resolution.

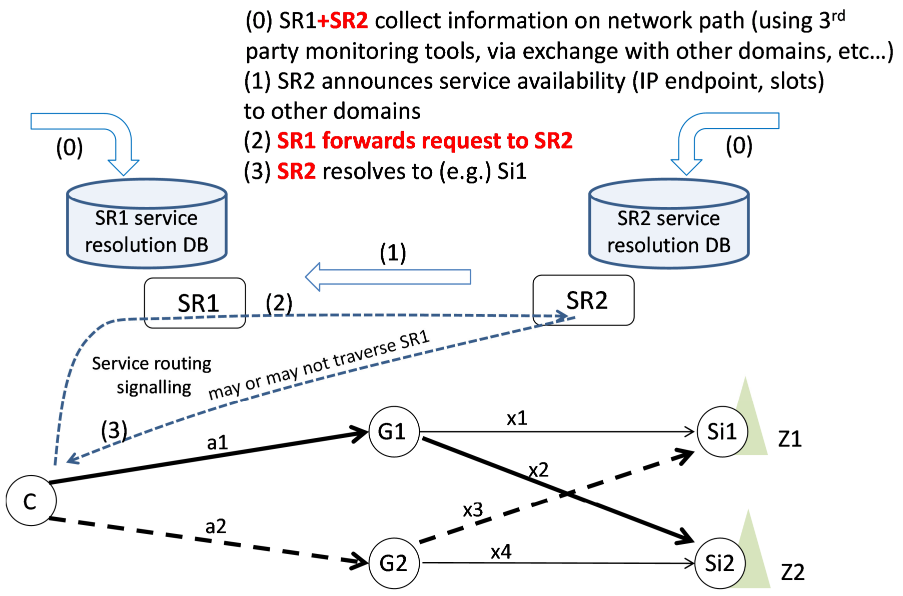
(0) SR1+SR2 collect information on network path (using 3rd party monitoring tools, via exchange with other domains, etc…)
(1) SR2 announces service availability (IP endpoint, slots) to other domains
(2) SR1 forwards request to SR2
(3) SR2 resolves to (e.g.) Si1
(0)
(0)
SR1 service resolution DB
SR2 service resolution DB
(1)
SR1
(2)
SR2
Service routing signalling
may or may not traverse SR1
(3)
a1
G1
x1
Si1
Z1
x2
C
x3
a2
G2
x4
Si2
Z2

Figure 4: Resolution in best instance domain.

The two basic scenarios described above can of course be generalised assuming the use of the candidate list mechanisms introduced in section 3.2. A generalised scenario where both originating and terminating domains as well as the orchestrator are involved in the resolution process is shown in Figure 5.

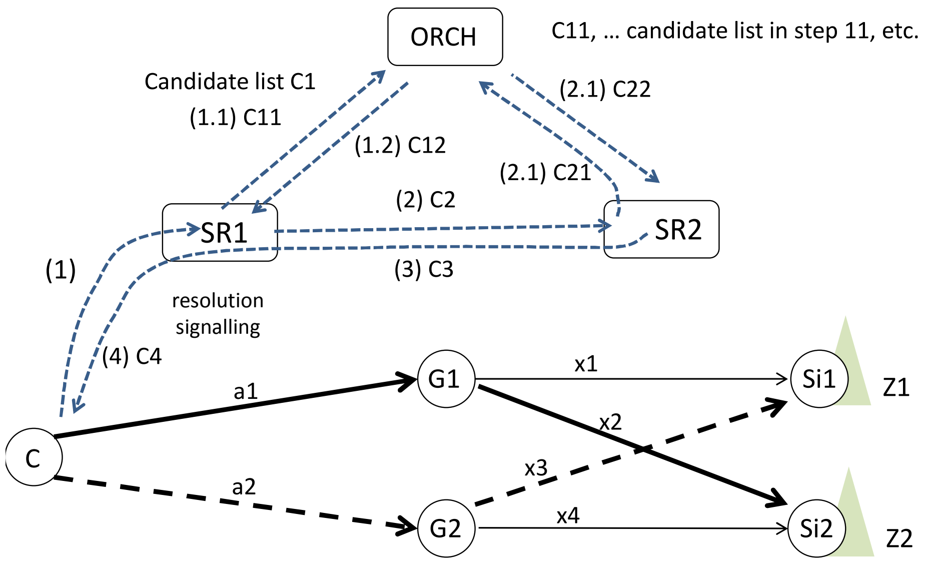

Figure 5: General resolution scenario with candidate lists.

In this scenario, resolution primitives are forwarded among the resolvers and additionally they are sent to the orchestrator. In each step a list of candidate service locators is exchanged between respective entities so that each of them may apply its policies to the list. The result of applying a policy is exposed in the form of the set of potential choices (candidates) available for the next resolution step. Actual conventions that should apply while changing the candidate list is an open topic, but we note that in other frameworks (e.g., SIP/SDP for codec negotiation) it is often assumed that the initial choice can be narrowed down and never extended in subsequent steps. We also note that the communication between service resolvers and the orchestrator can be based on either a dedicated API (routing hints) or the resolution protocol itself, but it seems that the actual form of this communication is not critical. What is important, this capability effectively resembles to redirection mechanisms in other protocols (e.g., HTTP and SIP).

3.3.2 Collection and exchange of information

The information to be advertised across service resolution domains can relate to services (e.g., service slots, histograms) and to network paths that can be used by connections between clients and service instances.

Session slot/histogram advertisement for the inter-domain scenario is discussed in more detail in section 6.

As explained in more detail in section 5, the primary model adopted by FUSION to cope with network level performance during service resolution and orchestration builds on the use of network distance information provided by systems external to the FUSION platform. Nevertheless, there may arise situations when service resolvers do not have sufficient information about network costs. For example, this may occur for certain services that are served in very distant domains and when available network proximity systems lack or do not provide this information. In such a case some form of support from FUSION service resolution layer for the exchange of related information may be helpful. However, this rises practical concerns as explained below.

Theoretically, the exchange of network proximity information within service resolution layer can be done either out of band or in band (during the resolution process). We note that from the service resolution perspective, to be useful such information has to be very specific. One possible solution assumes announcing metrics related to well-known gateways (or landmarks) that would serve other domains as reference point to enable them reasoning about actual end-to-end costs between clients and service instances. For example, referring to Figure 3, in order for resolver SR1 to be able to derive the end-to-end network path cost (e.g., delay) from client C1 towards service Si the following would be necessary:

(a) service resolver SR2 would have to announce to SR1 actual cost from each gateway G1, G2 to each zone (location) Z1, Z2,

(b) for each location Z1, Z2, resolver SR1 would have to know actual gateway that handles traffic forwarded from C towards Z1 and Z2, and

(c) only based on a) and b) could SR1 calculate the actual cost from C to Z1, Z2.

We note also that the detailed information to be exchanged will depend on the resolving domain and the direction of traffic flows. Overall, we believe that exchanging network level information within the service resolution layer in the multi-domain scenario increases implementation complexity of service resolution plane and may raise scalability concerns. Therefore we defer deeper studies on this topic until a sound use case is identified to justify the introduction of this option.

4. SERVICE RESOLVER

Figure 6 presents an overview of the different functional blocks of one service resolver, together with the interfaces exposed by these elements.

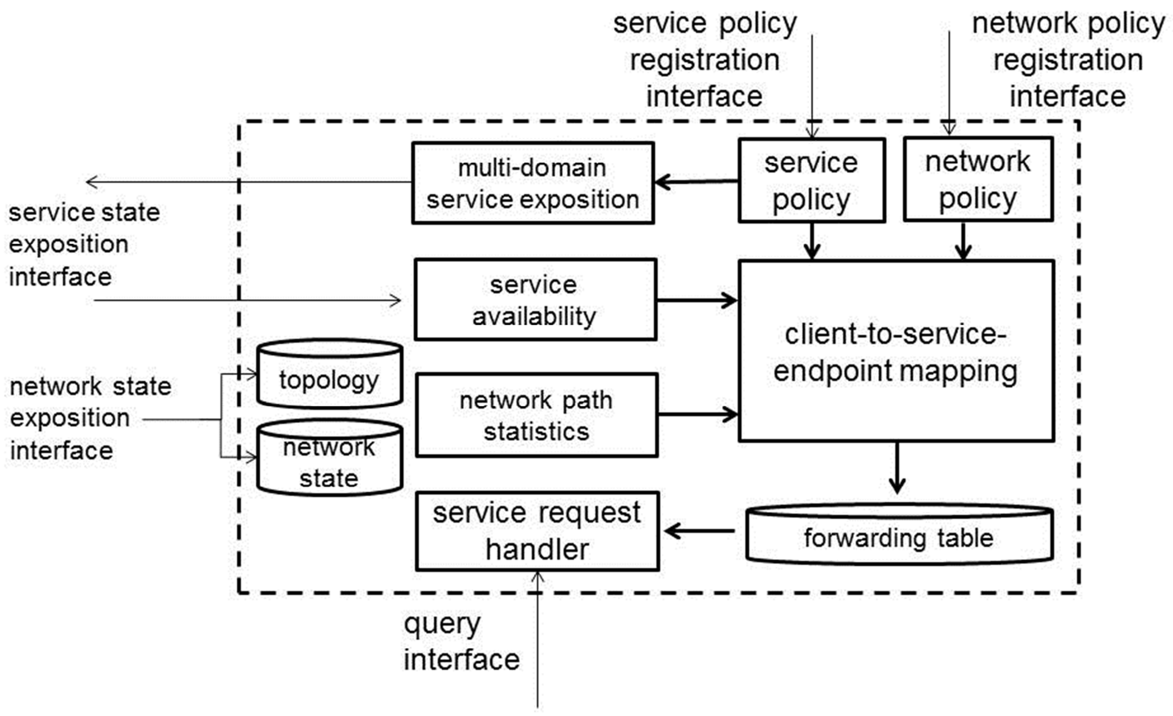
service policy registration interface
network policy registration interface
multi-domain service exposition
service policy
network policy
service state exposition interface
service availability
client-to-service-endpoint mapping
network state exposition interface
topology
network state
network path statistics
service request handler
forwarding table
query interface

Figure 6: Functional decomposition of the service resolver

The core functionality is provided by the service request forwarding table; which is queried when a service request is received via the query interface. For each tuple (client IP, service ID), the forwarding table specifies one of the following actions:

- **Resolution**: the service resolver will answer the query with an IP/port tuple of the selected service endpoint. This endpoint is used as destination address by the client when establishing a data plane connection to the service.
- **Forward**: The next service resolver to forward the request to, in case the request cannot be served within the service resolution domain.

The forwarding table is populated by the mapping component, which takes various inputs into account:

- **Network State:** this module builds a database that tracks network metrics on the end-to-end path between client IPs and execution zones and network topology information. It takes input from the network state exposition interface. This interface is discussed in section 5.
- **Service Availability**: services are identified by service ID. For each service, this module tracks the availability and current service load per execution zone. This interface is discussed in section 6.
- **Service Resolution Policy:** the orchestration layer (discussed in D3.2) can configure specific resolution policies via the corresponding interface. Service Resolution policies may impose restrictions on the network path between client and service instance, like geographic restrictions, minimum bandwidth or maximum latency. Policies can also be related to deployment and orchestration, e.g. favour specific zones, or on total service performance, e.g. maximum latency including network latency and service response time).

- **Network Routing Policy:** configured via the corresponding interface, these policies are related to the underlying transport network (for example traffic engineering).

Lastly, the service resolver may expose information on service instances running in its execution zones to service resolvers running on other domains. The advertisement of such information is subject to orchestration and/or resolution policies, as discussed in section 3.

5. NETWORK STATE EXPOSITION

In theory, service resolution nodes can use a broad range of network-level parameters for running resolution algorithms. Packet delay, loss rate, bandwidth and prices can be mentioned as the most popular metrics.

Estimation of most of those parameters requires the collection of network monitoring information in real or near-to-real time which itself is a non-trivial task. Collected data sets will typically comprise information about network topology and network state (router status and load, link status and utilisation) to build network maps of ISP domain. Topological and routing information within domain can be retrieved from IGP protocol. Annotations like link utilization and other metrics can be gathered using SNMP. BGP information can serve to find egress points for traffic while Netflow or similar tools can help identify ingress points so that actual network paths for all prefixes can be determined. Moreover, the above information can be augmented utilising performance maps of own domain obtained using passive probes or from active tools based on, e.g., OWAMP and/or TWAMP.

The above tasks apply in principle at the level of a given domain where the access to respective information can be enabled relatively easy for organisational purposes. Therefore, to complement the overall network view, performance maps obtained from other ISPs or third parties, e.g., RIPE Atlas, can be used to extend the coverage of network maps beyond the own domain of an ISP.

Theoretically, gathering and processing all needed information (from raw measurements to information directly used by the resolution procedures) can be done by the service resolution layer on its own. We note however that from the point of view of actual resolution procedure/algorithms network information presented at relatively high abstraction level is sufficient. Therefore, and considering the emergence of IETF standards for dissemination of network level information, we propose the use of ALTO protocol [APYA14] at the interface between FUSION service resolution layer and network monitoring for network state exposition.

Depending on policies and/or the feasibility of specific information, service resolvers may require that information directly used for resolution purposes be available at different levels of granularity. Desired objects to describe are execution zones, services, service instances, but also aggregates such as sets of execution zones or sets of clients. We note that ALTO provides great flexibility in terms of both type and granularity of information for annotated network maps. In particular, the concept of Provider-defined Identifier (PID) seemingly covers all the above requirements in terms of granularity. Similarly, ALTO cost maps define one-way connections between pairs of PIDs and provide a great freedom in defining the semantics of annotations. Although ALTO protocol is based on HTTP and JSON which may rise performance concerns, we believe this need not be a key aspect for FUSION. In fact, one can always adopt the implementation to cater for actual constraints and even define more lightweight notation similar to, e.g., DNS.

Lastly we note that similar information as the one needed for service routing should also be available to the orchestrator for deciding on service instance placement based on network metrics. Adoption of ALTO also in this case makes it a perfect solution for network state exposition within the whole framework of FUSION.

6. SERVICE STATE EXPOSITION

In order for the service resolution to make informed choices regarding service selection, service resolvers need to have several pieces of information: service availability per execution zone and network performance data. Service selection will run an optimisation algorithm which will try to maximise user quality of experience and minimise operators' costs. Here we deal with the service specific data.

6.1 By application components

After a set of service instances is instantiated in an execution zone, the zone manager is responsible to inform the local service resolver. This involved passing several items that will then be selectively propagated in the resolution system. A crucial concept is that of "session slot". This is a common measure to any service that quantifies "serving one user".

The following data is passed between the execution zone and the local resolver:

- Number of currently available session slots
- Number of currently used session slots
- A distribution, in form of an histogram, of service times of the sessions running in that execution zone. This will be potentially used for resolvers to forecast what will be near future usage of a given service and choose an instance that minimises blocking probability (and load balances more efficiently)
- The utility function of the service regarding quality of experience. This is a function of metrics like end-to-end delay and/or available bandwidth. The values of the utility vary between 0 (the best) and -1 (the worst).

6.2 Multi-domain service state exposition

In order for resolution to work across several domains resolvers need to organize themselves in an overlay. Here we explain how the process works:

1. Each domain has a logically centralized resolver that answers queries from the domain's clients, receives registrations from endpoints inside the domain and interacts with resolvers in other domains to exchange service instance availability. We refer to this exchange as routing.
2. A domain will typically be equivalent to today's Autonomous systems. If an AS is too big it can break it in several domains. Finally, domains can consist of federated domains between several organizations or providers where they have similar network characteristics to the rest of the world. We expect this last feature to reduce the total number of resolvers visible to the rest of the world.
3. After a new service instance gets registered in its local resolver, this information will be propagated to other domains. After this process finishes, clients will be able to resolve names to locators. We detail the routing process first and then the resolution process.
4. Resolvers can connect to any resolver in the Internet. They are not constrained by layer 3/BGP connectivity. This can happen for the exchange of routing information and forwarding of resolution queries. They need to find, through another mechanism (email, website, face-to-face) at least one address of another resolver. Resolvers will have an ID and a public key associated with them.
5. Routing consists of two distinct steps: Catalogue sharing and service subscription.
6. **Catalogue Sharing**: In Catalogue sharing each domain advertises that it has a new service available. This information consists of serviceID, utility function, representative locator and resolver locator. The utility function will define the characteristics of the service, the

representative locator will allow other locators to assess the potential performance of connections to instances running in that domain. The resolver locator will allow other resolvers to connect and subscribe to updates from that service. In case of larger domains, multiple locators can be advertised.

7. The Catalogue can be propagated directly (to resolvers that are directly connected), or forwarded through a multicast/spanning tree. The former raises less trust issues but the latter produces less traffic and allows for a resolver to only need to know of one other resolver. Note that this information is not very dynamic. An update is only sent when a new service is created, all the service instances have been deleted or there are significant changes in network connectivity (e.g. change of traffic engineering policy). It does not generate new updates every time a service instance is created.
8. The propagation of Catalogue information can further be optimized by not propagating information to domains where the service utility would be minimal (e.g. availability of a service that requires 10ms delay should not be propagated to far away domains).
9. **Service subscription**: Whilst receiving Catalogue messages, resolvers decide if they should subscribe to routing updates from a given service from a given domain. They will try to contact close domains before and obtain enough diversity of service availability. They will expand the subscriptions until enough instances are found. Note that potentially they will have to connect to every other resolver in a full-mesh. These should be in the order of 10,000s, which a modern operating system can cope with. Moreover, the routing functions of a resolver can easily be parallelized. Yet another way to reduce the number of connections/relationships would be the existence of resolvers run by neutral organizations that aggregate routing information and make resolving decisions on their behalf. These could be run by a myriad of types of organizations: top-level domain administrators, non-profit organizations, cooperatives or a new business entity. Note, however, that incentives issues would arise.
10. After subscribing to a given service, a resolver will start receiving updates from that service in that domain. This will include service slot information and the locator(s) for which the name is mapped. A flag detailing if the resolution can be done locally or remotely is also included (more on this below). These locators can be of different types (see Appendix A) including the locator of the resolver (and not final instance) in the case of remote resolution.
11. Service resolvers must either propagate service availability information (possibly after aggregation) received from their neighbours or register in the catalogue as “proxies to other domains” so that requesting domain resolvers can subscribe to them for routing updates in usual way.
12. By the end of the process, every resolver should be able to deal with any query for any existent service through the resolution process.
13. **Resolution:** Resolution takes place when a client wants to resolve a name to a locator. It can happen in two ways: Locally (in the client’s domain) or remotely (in another service resolution domain closer to the best service instance).
14. As mentioned before (in section 3.3.2), resolvers are equipped with the capacity of assessing network distances to any IP address in the Internet. This decision will take into account: the service utility function, the location of the user and the traffic cost in reaching all potential instances.
15. In local resolution (cf. Figure 3) the resolver selects the best service replica and returns immediately the resulting locator to the client. Local policies may be applied during this process.
16. In remote resolution (cf. Figure 4) the resolver chooses to which resolver to send the query and forwards it. At the other end the remote resolver will decide which instance to return which gets then returned to the user. It may apply its local policies for the selection as described in section 3. The origination resolver can also restrict the choices that the remote resolver returns based on its local policies.

17. Clients have the option of restricting even further the results (by specifying extra QoS constraints in requests) and to receive multiple locators. Applications can make use of these to achieve resilience, or parallelism.

6.3 By service endpoints/clients

Our architecture includes client quality of service experience as a feedback loop that can help for better service selection. This can improve the accuracy of service resolution, in particular because is server specific unlike many other network data, but it is not necessary for the resolution to take place. Resolvers have a plethora of other mechanisms to obtain this information.

After clients use the service they send performance information to their local resolvers. This consists of observations between locators for a given service and corresponding final metrics like goodput, loss and delay. The information is similar to that sent by RTCP [RTCP] but sent to the resolver rather than the end system. It should be noted that the use of QoE feedback is an optional feature and not essential for FUSION, but will provide a useful user point-of-view of actual performance to augment monitored network metrics.

There is an incentives issue here since this will require service developers to provide this information through our interface. We believe that more accurate resolution will be a strong enough incentive. We are also aware that QoE reporting can be faked. Resolvers may have to apply statistical tests to remove outliers.

7. QUERY

This interface implements the main primitive of the service resolution layer. Using this interface, clients can request the best matching service instance by name. The actual data communication is service-specific and completely agnostic to FUSION.

7.1 Service ID Namespace

The FUSION service resolution layer will resolve a service ID to an IP/port tuple that is publicly addressable. The service ID identifies the *endpoint* of a service, as defined in the service manifest by the service provider or orchestrator and advertised by the zone gateways.

Obviously, the service resolution will have multiple endpoints per service name, at least one per execution zone that hosts the service. But there is also no unique mapping between a particular locator to the corresponding service name. Consider an atomic service that is advertised as SID_A, but that is also defined as the endpoint of a composite service with identifier SID_B. In this case, the IP/port tuples of the instances of this particular atomic service will be mapped from both SID_A and SID_B.

The FUSION resolution layer operates on a flat name space for service IDs. Hierarchical name structures, as e.g. proposed by Van Jacobson [VST09], are based on organizational structures, folder structures, etc. However, such naming schemes introduce fixed interdependencies that are avoided by flat name spaces. In our view, service IDs should be independent from any hierarchical organization, as they will be used to refer to instances running in any zone of any service resolution domain.

However, although the service resolution layer does not assume any hierarchy in the service name, the use of hierarchy by service providers (e.g. starting each name with their domain name) is not prohibited but will be transparent to the service resolution layer.

Although flat name spaces do not allow for name aggregation by service resolvers, we argue that the impact of using flat namespaces on the scalability of the FUSION service resolution layer is limited:

- There is no resolution authority that must be queried for a particular service ID. Each service resolver that knows the locator of an instance that matches the service requirements may answer the request. The list of registered locators can differ per service resolution domain. This is in contrast with DNS, where each domain name is governed by a particular authoritative name server. Other DNS servers cache resolution answers, but ultimately the service ID is always resolved to the same IP address[4] mandated by the authoritative name server. The FUSION service resolution layer is conceived in a different way, where each service resolution domain has the autonomy to resolve service IDs to different instances.
- Flat names allow the use of hashing techniques for look-up operations in the forwarding tables of service resolvers. The population of these forwarding tables by the service request resolution algorithms is decoupled from the look-up operations and thus does not stall the request resolution.

7.2 Resolution policy

Some services will not be available in all domains and corresponding requests will have to be resolved to an instance in a remote domain. There are different options regarding the authority for service request resolution and the actions a service resolver can take to process a service request.

[4] When DNS load balancing is used, other name servers might cache only part of the aliases. However, all these aliases are provided by the authoritative name server.

Below, we discuss these different modes of operation that are supported by the FUSION service resolution layer, in decreasing order of generality:

1. Requests can be resolved by any service resolver in any domain

This is considered as the default operation mode of the service resolution plane. Service resolution domains advertise service IDs, locators (IP/port) and load information to other (remote) domains. For such services; service resolvers can resolve received requests to service endpoints advertised by any execution zone, even in other domains.

In this scenario, service endpoint advertisements can be cached by service resolvers in other domains which improves the stability of the service resolution plane. The main challenge to be overcome is that information on service availability may be inaccurate or out-of-date.

2. Requests can only be resolved in a particular domain

If a service resolver in domain A resolves a request to an instance hosted in domain B, the service requestor will in most cases subsequently connect to the selected instance and consume network and execution zone resources. For this reason, a service resolution domains might prefer to keep better control of the resolution of requests to instances within their domain.

The most straightforward way to achieve this is to simply not advertise service instances to other domains. However, this reduces the service consumption in contracted execution zones as no traffic from clients in remote domains can be attracted. Instead, the FUSION service advertisement interface must support this “only-local-resolution” policy by allowing to only advertise the availability of a particular service at a more abstract level without specific locators. This forces server resolvers to forward the request to the advertising resolution domain, as they have no information on the exact locators but are only provided with a list of candidates, as explained earlier in section 3.3.2.

3. Requests can only be resolved at the selected execution zone or by a service component

Some services might prefer handle instance selection internally, rather than relying on the FUSION resolution plane. One relevant use case is cloud gaming, where the performance of candidate instances must be evaluated against service-specific criteria (e.g. using evaluator service) and the service provider wants to keep control of intra-zone resolution.

In the most basic scenario, the FUSION resolution plane is only used to resolve the request to a locator of this service-specific orchestrator module. However, the FUSION resolution plane may provide additional functionality, e.g. by sending the request to the zone with the lowest latency.

7.3 Invoke

The default operation mode of service requests is to return the locator (or a list of locators) of the best instance by service resolvers. In this mode, the request message is never received by the service instance itself and the client establishes a data connection out-of-band of the FUSION service resolution plane.

Another mode of operation is to directly send a message to an instance of the specified service ID. This avoids an additional round-trip time and extends the functionality provided by the FUSION service resolution plane. An example scenario is a cloud gaming service that wants to perform its own service choreography. Using the invoke mode of the query message, the request is resolved to a suitable zone and directly delivered to the service endpoint. The service choreography component can then immediately return the list of locators of the selected components (note that such a choreography component could be thought of as a special service-specific resolver).

8. NSRP PROTOCOL DESIGN

This section provides an overview and motivation for the new Networked Services Resolution Protocol: NSRP

We analysed the potential advantages and disadvantages of deviating from DNS style based protocol towards a text based protocol like HTTP 1.1 (HTTP 2.0 also moved towards a binary based protocol). After careful consideration we decided to stick with a binary protocol for the following reasons:

- The data being exchanged contains only limited amount of text (the names being subscribed. The majority of the data consists of IP addresses, IP prefixes, ports, options and flags. Binary representation reduces the amount of data being transferred. This comparison also applies if we compress both options. Compressing text based data can achieve similar data sizes when large amounts of data are exchanged in the same message. This only happens occasionally in our architecture (typically in bootstrapping a resolver with service name/address data).

- Parsing can be a significant overhead for resolution servers. The amount of requests/subscribes can be significant which will increase the processing requirements of resolver software.. It will also introduce extra processing delay, especially if a message needs to be forwarded to several resolver.

- Less errors: As stated in [HTTP2], binary protocols "are much less error-prone, compared to textual protocols like HTTP/1.x, because they often have a number of affordances to help with things like whitespace handling, capitalization, line endings, blank links and so on. For example, HTTP/1.1 defines four different ways to parse a message; in HTTP/2, there's just one code path."

We also considered the use of REST/HTTP: Although REST/HTTP provide advantages for application developers they incur a significant overhead in data transmission and processing time with no benefits for name resolution.

The biggest disadvantage of a binary protocol is human readability and ease of debugging. Given that name resolution is conceptually placed between the application and network we don't see any need for human readability. Debugging can be solved with a purpose built tool like a Wireshark plugin.

The protocol consists of the following messages with several options which are further detailed in Appendix A.

1. REGISTER/UPDATE: Used by a Zone Manager to advertise to the local resolver the existence of one or more service instances. The message is sent to the local resolver and is continually updated
2. SUBSCRIBE: Used by resolvers to subscribe to service updates from other resolvers. It is also used to request Catalogue Information.
3. QUERY: Used by a client to convert a name to one or more locators. This message may be propagated to other resolvers.
4. DATA: Data containing mappings from name to locators. It is sent in response to a SUBSCRIBE (both Catalogue and Service subscription) or to a QUERY
5. QoE REPORT: Messages issued by the clients to report quality of experience. This should serve as input to service instance selection

6. HELLO: Initial message exchanged by resolvers to authenticate themselves
7. ERROR: Error message sent in response to a QUERY, SUBSCRIBE, REGISTER, QOE REPORT or NETWORK_DATA_REQUEST

Before we embarked in defining a new protocol we analysed if DNS could not be used to achieve our goals. Here we compare our protocol with two other possibilities: unchanged DNS and a potential extension of DNS that changes the server to server protocol but keeps intact the client to server part.

NSRP Features	**DNS**	**DNS in clients; modified server to server protocol**
Name to Locator Resolution of arbitrary name spaces	Not possible	Possible with some restriction on names (63 bytes between ".”s and only letters, numbers and the "-“ symbol.
Replica management, load balancing	Not possible	Possible but without ability of service being able to specify utility function or use histogram
More complex locators	Some combinations are possible (for example SRV RRs allow for port numbers). More complex result, like containing more than one IP address would require new RRs to be defined)	
Dynamic updates to client	Not possible	Not possible
Direct invocation	Not possible	Not possible
Client restrictions on results	Not possible	Not possible
Authentication at source	Not possible	Not possible

As can be seen a substantial set of features cannot be obtained with any of these two possibilities. A small subset, however, can be implemented by maintaining the client-to-server interface which be used as temporary deployment strategy.

9. RESOLUTION ASPECTS OF COMPOSITE SERVICES

Composite services are registered via their manifest, specifying the individual components to the orchestrator layer. In this section, we want to discuss the impact of composite services on the design and implementation of the components involved in service request resolution.

In section 9.1, we discuss the implications on the zone gateway. The zone gateway can assume a service resolution role when instances of each component are available in the same zone. We also highlight the advertisement of session slots for composite services.

Although we can safely assume that the zone gateway can interact with the zone manager and thus has access to the structure of the composite service, a more fundamental design decision is to what extent this service graph should be known to the service resolver, as to support composite services across execution zones. This is described in section 9.2.

9.1 Zone gateway

The zone plays an important role in the advertisement of composite services, as well as in the scenario of dynamic service graph deployment in a single zone, where multi-configuration service components are reused across multiple composite services. The latter functionality creates a more open service ecosystem with run-time chaining of components developed by service providers.

9.1.1 Advertisement of composite service availability

The zone gateway is responsible for the advertisement of service availability to its service resolver. In D3.2, we introduce the notion of *service* session slots. Instances of a service component not only report *resource* session slots (determined by the number of available resources and the number of running connections), but also service session slots, reflecting how the available resource session slots are allocated over the different configurations for this instance.

Hence, the service availability that must be reported to the service resolvers are better reflected by the number of service session slots. For atomic services, this number can be calculated as described earlier in section 2.1 by summing the numbers reported by each instance. The situation is however more complex for composite services. Consider a composite service that comprises service component A and service component B. In this case, the number of session slots to be reported is the minimum of the available session slots for service A and for service B. Another complexity is that one the serving of one incoming client connection could require multiple session slots (e.g. a composite service that requires two decoders). Clearly, the zone gateway must take into account the service graph that has been provided in the service manifest.

9.1.2 Zone-internal resolution

In Figure 7, four service components are involved in three composite services. Service component A has been configured with two session configurations (A-B and A-C). As explained in D3.2, the two service configurations of service component A has been mapped onto different ports. Depending on which port a client connection is received, service component A will request the zone gateway, who assumes the role of service resolver, for the locator of an instance of service B or an instance of service C. Hard-wiring actual instances would limit the orchestration flexibility and deployment resiliency (e.g. when a component fails), especially for long-lived sessions. Moreover, this would prohibit dynamic scenarios where components are added to and removed from the service graph at runtime.

In the remainder of this section, we discuss two different patterns of data connection between service components that must be supported by the zone gateway. The discussion is supported by Figure 7. It is important to stress that the arrows on this figure indicate the data flow between components. Service requests to the zone gateway are not depicted.

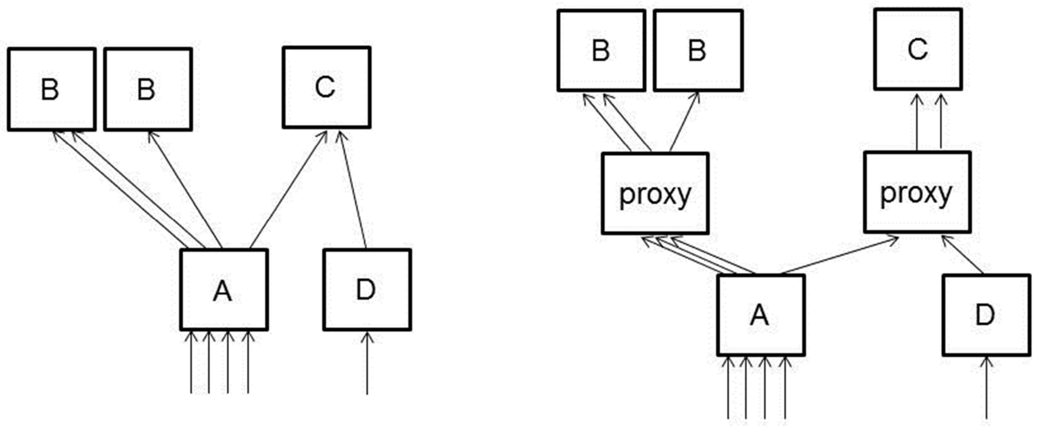

Figure 7: Patterns for the flow of data across service components

On the left side, service component A contacts the zone gateway for *each* session. Depending on which the port the client connection is received, component A will request the gateway for an instance of component B or component C. The main advantage of this scenario is that it allows for fine-grained resolution *per session* for a composite service. The zone gateway must only take into account the number of available resource session slots (although it must still map these to *service* session slots for advertisement to the service resolver). The management of *service* session slots is handled by the zone orchestration layer.

On the right hand side, we show a different orchestration pattern for connecting instances. In the *ambassador* pattern, proxy components manage the load-balancing between replicas of the same component. This pattern is heavily used in the emerging container-based virtualization solutions like Docker, where it is currently the *de facto* standard to connect two containers that run on different hosts of the same execution zone (data centre). In this scenario, the instance selection process comprises two steps:

- First, the zone gateway will return the proxy IP/port for the requested service component.
- Second, the data connection will be load-balanced to one of the instances. Load-balancing is handled per-service.

Given the relevance of this deployment scenario, we must support it in the FUSION architecture and implementation. The main challenge here is that the proxy only performs basic load-balancing and does not take into account service characteristics or the location of the requesting component. In the example of Figure 4, the proxy for service component C does not take into account whether the data connection is originating from an instance of service A or an instance of service D.

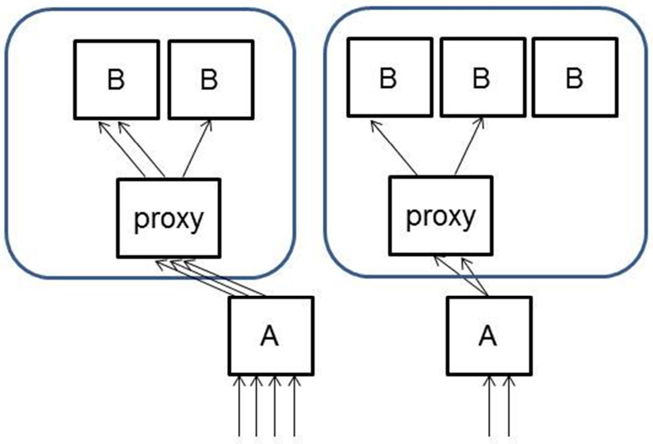

Figure 8: Multiple proxies for the same service

A possible solution is to group the different instances behind multiple proxies (e.g. one proxy per node on which the instance is deployed). In this case, the zone gateway takes an intermediate role and selects the most appropriate proxy, rather than the most appropriate instance. The drawback of this scenario is in the tighter coupling between the zone gateway and the zone orchestration layer.

9.2 Composite service resolution

As described in section 3.1.2, we discern different architectural options for composite service resolution:

- The service resolution layer returns a list of all instances
- The service resolution layer is unaware of the service graph. This means that each service component individually requests the next instance in the service graph, leading to potential suboptimal deployment.

In the latter case, the composite service graph must be injected into the service resolution layer (one possible option for doing this is to use redirections/routing hints sent to the orchestrator as suggested in section 3.3.1, Figure 5). This graph describes the individual components of the service (e.g. service ID) and their interconnection with a possible cost metric. Introducing the service graph into the service resolvers however entails bringing additional complexity for the forwarding process. For each component, the service resolution must select the optimal instance.

When the service request resolver receives a request for a composite service, it needs to create the corresponding *instance* graph from the *deployment graph*. An example is shown in Figure 9. In this example, the composite service exists of four service atoms A-D, with service atom A serving as the endpoint for clients to connect to.

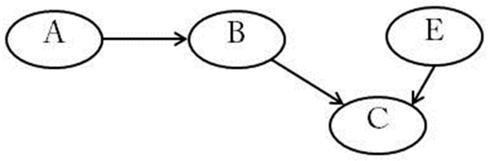

Figure 9: Composite service with four atomic services

When a request arrives for the composite service, the service resolver builds the service graph shown in Figure 10. At the time of the request, there are two available instances for atomic services A, B and D, while there is only one instance available of atomic service C. The service resolver must now determine the most appropriate subgraph, that contains exactly one instance of each atomic service. In general, shortest path algorithms cannot be applied since there is not necessarily a single source and single sink node. Moreover, the order and size of the instance graphs rapidly increase with the number of instances of each atomic service. If *N* instances of atomic service A and deployed, and *M* instances of atomic service B, then the graph will contain *NxM* edges since in principle each instance of B can be reached from any instance of A.

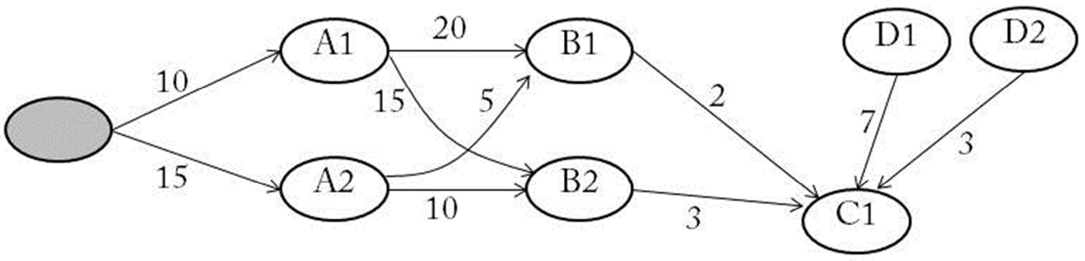

Figure 10: Service resolution must pick the best combination of instances

One possible heuristic is to keep the service graph transparent to the service resolver. In such a scenario, the service resolver would always return the instance with the lowest cost from the requesting client. Referring to Figure 10, the service resolver would first return instance A1 to the client. In turn, this instance would be routed to instance B1, and this instance would in turn be resolved to C1, resulting in a total instantiation cost of 28 (10 + 15 + 3). However, the lowest subgraph consists of (A2, B1, C1) and has an instantiation cost of 23 (15 + 5 + 3). This example shows that this heuristic may rapidly lead to increased instantiation costs, especially for more complex services.

10. INTRA-DOMAIN ROUTING BASED ON SESSION SLOTS

In this section, we describe the progress of a dynamic instance selection algorithm for a single resolution domain, which considers both network path cost and instance capacity cost (service session slots). Upon request arrival, our selection algorithm now aims to minimize the bandwidth usage by finding the shortest path to a server.

We also start the work on the interaction between the service resolution layer and the orchestration layer. If network congestion occurs, it is preferable to have the orchestrator redistributing the current number of running instances rather than scaling up additional instances.

10.1.1 Problem description

We are looking into a resolution service for a single domain which assigns each request to a network path and service instance. In the current status, we assume one single service resolver that has access to monitoring information on any instance running in an execution zone of this domain, as well as to detailed network path statistics, in particular the current bandwidth usage on links.

When no available path or service instance is found, the service request is rejected by the resolution service. The main objective of our research is to minimize the amount of rejected requests and serve as many users as possible.

To ensure that our selection algorithm only selects paths with available bandwidth and service instances with more than one remaining session slot, we construct a *service availability graph* upon request arrival. This graph only contains network edges with sufficient bandwidth for at least one additional service session. We then consider all service instances which have at least one session slot available and for which we can find a path to the client in the availability graph. This guarantees that at least one path with sufficient bandwidth can be found from the client to any of those execution zones. Using this availability graph we can implement selection algorithms in a simple manner.

Consider a network graph containing edges E, nodes N, services S and service instances M. Client nodes are denoted $N_c \subset N$ and execution zones are modelled as (server) nodes in the graph which are denoted $N_E \subset N$. The collection $M_S \subset M$ contains all service instances of service s ∈ S, which are running in the execution zones. $G_{j,s}$ is 1 if server node *j* is hosting a service instance of service *s*, else it is zero. Each edge $e \in E$ has a total bandwidth capacity B_e and $B_{e,t}$ denotes the available bandwidth on time *t*. Each service instance $m \in M$ has P_m session slots and can thus serve this many clients at the same time, while $P_{m,t}$ denotes the number of available session slots at time *t*. When a service request *r* is served, this incurs a bandwidth consumption W_r on the path between client and execution zone, and requires one session slot of the selected instance.

Upon request arrival at time t, our selection algorithm must find a network path where all edges have $B_{e,t} \geq W_r$ and $P_{m,t} > 0$. We aim to minimize the bandwidth usage by searching for the shortest possible paths.

10.1.2 Selection algorithms

1. **Shortest Path Only:** A first approach algorithm constructs a service graph containing all edges of the original network ignoring the available bandwidth capacity of the network. We then search for the shortest path between the client and the closest server node with an instance that has at least one session slot available. If that path does not have sufficient bandwidth available (considering all other active connections) for the actual bandwidth required for that service request, the request is rejected. This simple approach does not make use of alternative longer paths and creates many network bottlenecks which could be avoided by distributing traffic over several paths.

Input: Request *r* for service *s* from client *c*.
Output: Network path *L* from client *c* to service instance *m* running on server node *j*. Or an empty path if the request is rejected.
Pseudo code:

```
L ← EMPTY;
minLength ← INFINITY;
For each service instance m in Ms do {
    if ($P_{m,t}$ > 0) {
        d(c,j) ← shortest path between client c and server j hosting instance m;
        if (length(d(c,j)) < minLength) {
            L ← d(c,j);
            minLength ← length(d(c,j));
        }
    }
}
For each edge e in d(c,j) do {
    if ($B_{e,t}$ < W)
        return empty path; // reject the request
}
return L;
```

2. **Equal distribution:** We try to avoid bottlenecks by equally distributing the load induced on each edge and service instance. We start with the availability graph containing all edges and service instances in the network. For each incoming request, we look up the network path and service instance in the availability graph which induces the least variance in edge and instance load; this is the most even distribution. Using this equal distribution we aim to delay any bottlenecks until they can no longer be avoided because all available resources are drained. The disadvantage of this approach is the higher bandwidth usage due to the use of longer paths when a shorter path was still available.

   This algorithm goes as follows:

Input: Request *r* for service *s* from client *c*.
Output: Network path *L* from client *c* to service instance *m* running on server node *j*. Or an empty path if the request must be rejected.
Pseudo code:

```
L ← EMPTY;
bestScore ← INFINITY:
// construct the Service Availability Graph SAGraph
SAGraph ← contains each edge e with $B_{e,t} \geq W_r$
Q ← list of all service instances m ∈ Ms with $P_{m,t}$ > 0 for which we can find a path to client c in SAGraph

if (Q is EMPTY) {
    return empty path; // no path from client c to instance m available or no service instances with $P_{m,t}$ > 0 were found.
}
```

```
// Consider each service instance as final destination of request r
For each service instance m in Q do {
    serverUsages ← EMPTY;
    For each service instance n in Q do {
        serverUsage ← $P_n – P_{n,t}$;
        if (m equals n) {
            serverUsage  = serverUsage - 1; // m must also process request r
        }
        serverUsage  = serverUsage  / $P_n$;
        serverUsages.add(serverUsage);
    }
    slotVariance = variance(serverUsages); // slot variance if m were to process request r

    // now we calculate the network bandwidth variance for k-shortest paths
    K(c,j) ← k-shortest paths between server j (hosting m) and client c in SAGraph
    For each path p of K(c,j) do {
        networkUsages ← EMPTY;
        For each edge e in SAGraph do {
            edgeUsage ← $B_e – B_{e,t}$;
            if ($e \in p$) {
                edgeUsage = edgeUsage – $W_r$; // e must carry request traffic
            }
            edgeUsage ← $B_{e,t}/B_e$;
            edgeUsages.add(edgeUsage);
        }
        edgeVariance = variance(edgeUsages); // bandwidth variance if request r were to be
forwarded on path p

        // use slot variance and edge variance to find a weighted total variance
        totalVariance ← $\alpha$ * slotVariance + (1-$\alpha$) * edgeVariance;
        if (totalVariance < bestScore ) {
            bestScore  ← totalVariance;
            L ← p;
        }
    }
}
return L;
```

Shortest Available Path: this algorithm minimizes the bandwidth usage while using the service availability graph to find alternative network paths and service instances in case of congestion. Upon request arrival, an availability graph is constructed containing edges with sufficient bandwidth for that request. Next, we find all service instances with at least one slot available and for which there is a path to the client in the service availability graph. The request is assigned to the service instance which is closest to the client in the availability graph. Using this algorithm, every request is assigned to the closest service instance when no bottlenecks occur. When the shortest path is congested then the shortest alternative path is used. When the closest instance has no more session slots available, then the request is assigned to the closest alternative.

When there are service instances available but there is no path to the client in the availability graph, then all network resources are in use and we have a network bottleneck. When there are no service instances available then we have a server bottleneck. In any other scenario the Shortest Available Path approach will assign the request to the network path and service instance which minimize the bandwidth usage.

Input: Request *r* for service *s* from client *c*.
Output: Network path *L* from client *c* to service instance *m* running on server node *j*. Or an empty path if the request must be rejected.
Pseudo code:

```
L ← EMPTY;
minLength ← INFINITY:
// construct the Service Availability Graph SAGraph
SAGraph ← contains each edge e with $B_{e,t} \geq W_r$
Q ← list of all service instances m ∈ Ms with $P_{m,t} > 0$ for which we can find a path to client c in SAGraph

if (Q is EMPTY) {
        return empty path; // no path from client c to instance m available or no service instances with $P_{m,t} > 0$ were found.
}


// Consider each service instance as final destination of request r
For each service instance m in Q do {
        // since we only consider instances in Q we already know Pm,t > 0
        d(c,j) ← shortest path between client c and server j hosting instance m in SAGraph;
        if (length(d(c,j)) < minLength) {
                L ← d(c,j);
                minLength ← length(d(c,j));
        }
}
// d(c,j) only contains edges from SAGraph so we already know $B_{e,t} < W_r$ for each edge in d(c,j)
return L;
```

The Shortest Available Path selection algorithm assigns requests in an optimal manner as long as the total demand does not exceed the available resources. When a network or server bottleneck occurs and all alternatives are in use, selection alone will not suffice to improve the current condition. Network bottlenecks can be solved by deploying instances on different locations in the network and server bottlenecks can only be solved by adding additional instances or upscaling the existing instances. Therefore we developed a dynamic placement algorithm which attempts to reduce the amount of rejected requests by changing the current instance placement and configuration.

10.2 Interaction between resolution and orchestration layer

When a network bottleneck occurs there are still available service instances to process requests but these instances are not reachable due to congestion in the network. By placing this instance in a different location, away from the congested network area, it becomes reachable again and more requests can be processed. Similarly, when no instances have available session slots we can upscale instances to process more requests or deploy additional instances. This requires an interaction between the resolution and the orchestration layer. We solve network bottlenecks by finding an appropriate location to deploy instances and we solve server bottlenecks by upscaling and deploying additional instances.

This algorithm goes as follows:

Input: The current service deployment configuration, including service instance locations G and session slot capacities P.
Output: a list V of new deployment configurations, including service instance locations G and session slot capacities P.
Pseudo-code:
Case: most rejected requests were caused by network blocks
Explanation: *we assume that popular service instance(s) or other background traffic are creating congested hotspots in the network and we mitigate this by deploying instances in other areas in the network*

```
For each service instance m in M do {
        $Pop_i \leftarrow W_r$ * serviceTime * amountOfRequests;
}
Q ← list of all service instances ordered by decreasing popularity $Pop_i$
For the first k service instances m in Q do {
        s ← service of which m is an instance of;
        K ← all server nodes j where $G_{j,s} = 1$; // current active servers for service s
        L ← all server nodes j where $G_{j,s} = 0$; // current inactive servers for service s
        j ← server in L which is averaged furthest away from all servers ∈ K;
        $G_{j,s} \leftarrow 1$; // deploy an instance of service s on server j
        V.add({G, P}); // add the deployment configuration G and session slot configuration P to possible solutions

        y ← least popular service instance of service s based on previously calculated $Pop_i$
        $G_{y,s} \leftarrow 0$; // remove the least popular instance to try and reduce server costs
        V.add({G,P}); // add the deployment configuration G and session slot configuration P to possible solutions
}

// V now contains 2*k new solutions which should be evaluated
```

Case 2: most rejected requests were caused by server blocks:
Explanation: *we upscale service instances which were unable to process the amount of requests while downscaling overprovisioned instances to reduce costs*

```
// First, we upscale the under provisioned instances to increase processing capacity
For each service s in S do {
        A ← average amount of rejected requests per second for service s;
        if (A > threshold) {
                m ← service instance of s which received the most requests;
                B ← throughput m; // average amount of requests processed per second by m
                slotsRequired ← A/B;
```

```
            $P_m \leftarrow P_m$ + slotsRequired;
        }
}
V.add({G,P}); // we add the new configuration after upscaling all the under provisioned instances



// Next, we downscale the overprovisioned instances to reduce costs
For each service s in S do {
        A ← average amount of rejected requests per second for service s;
        if (A < threshold) {
                y ← service instance of s which received the least requests;
                usedSlots ← average amount of requests processed per second by m
                $P_m \leftarrow$ usedSlots + $\alpha$; // we only provision as many slots as received on average. α is a
safety margin to avoid under provisioning this instance.
        }
}
V.add({G,P}); // we add the new configuration after downscaling all the overprovisioned instance.

// V now contains 2 new possible configurations
```

10.3 Evaluation

To evaluate these solutions, we run a simulation for each generated configuration. Each simulation uses the expected demand pattern as input. At the end of the simulation we are able to measure the amount of rejected requests as well as the total monetary cost of that deployment setup. We then use the most cost-effective configuration to configure the network.

10.3.1 Simulation Platform

Service selection and instance placement algorithms in FUSION must consider network constraints (*edge bandwidth*) and server constraints (*session slot availability*), both of which depend on the network topology and location of available execution zones. In order to evaluate the proposed algorithms on multiple network topologies with different characteristics, we developed a simulator to quickly perform these evaluations. This simulator combines existing frameworks to handle every aspect of the evaluation, from topology generation to event-driven simulation and presentation of results. Using algorithm hooks, users are able to run simulations using newly developed algorithms. We now describe the different aspects of the FUSION simulator.

10.3.1.1 Generating a network topology

Network topologies are generated using Brite [BRITE], a universal topology generator. This aspect of the generator takes a configuration file as input which describes the generation model to be used (*flat or hierarchical AS/Router topologies*) as well as network characteristics (*node count, node connectivity, edge bandwidth, …)*. The result is a list of network nodes and a description of their connectivity. Generating two topologies using the same configuration file might result in different topologies, though, with the same characteristics.

We use four different topology types to evaluate the developed algorithms. These topology characteristics are described in Table 2. SS50 and SD50 represent two smaller topologies with sparse and dense connectivity respectively. LS200 and LD200 are two larger equivalents with the same network characteristics. Using these four variations we study the performance of all developed algorithms under different network conditions.

Table 2 – Network configuration of each topology type used in simulations

	SS50	SD50	LS200	LD200
#service resolvers	50	50	200	200
#clients	5	5	20	20
#servers	3	3	3	3
Average outgoing edge degree	1	5	1	5
Avg. diameter	12	6	19	8
Brite model	Waxman AS	Waxman AS	Waxman AS	Waxman AS

The Brite model used to generate sample topologies is always Waxman AS. To illustrate, Brite uses the parameters presented in Table 3 to generate sample topologies of LS200 using this Waxman model. The parameter *m* represents the average outgoing edge degree from Table 2, while the amount of service resolvers from Table 2 equals N in Table 3.

Table 3 – Brite parameters to generate sample topologies of LS200

Parameter	**Value**		**Description**
Name	3		#Router Waxman = 1, AS Waxman = 3
N	200		#Number of nodes in graph
HS	100		#Size of main plane (number of squares)
LS	10		#Size of inner planes (number of squares)
NodePlacement	1		Random = 1, Heavy Tailed = 2
GrowthType	1		#Incremental = 1, All = 2
m	1		#Number of neighbouring node each new node connects to.
alpha	0.15		#Waxman Parameter
beta	0.2		#Waxman Parameter
BWDist	1		#Constant = 1, Uniform =2, HeavyTailed = 3, Exponential =4
BWMin	10		Minimum edge bandwidth
BWMax	100		Maximum edge bandwidth

10.3.1.2 Constructing a FUSION overlay

The topology generated by Brite has no knowledge of FUSION execution zones or clients. Therefore we convert the generated topology into a FUSION graph model which supports execution zones, service resolvers and clients. This graph model requires a user-specified configuration file containing the amount of clients, execution zones, services and how many instances of each service must be placed.

Initially, all nodes of the Brite network topology are added as service resolution nodes. Next, the simulator will assign the required amount of execution zones and clients to the network nodes. In order to increase the importance of the network characteristics we spread out execution zones and clients as much as possible, creating longer average paths towards the execution zones.

10.3.1.3 Initial service placement

The simulator attempts to place service instances on the assigned execution zones. This is the first algorithm which a user is able to specify as the initial placement has a big impact on the simulation results. Currently supported methods include: (1) randomized placement, (2) placing an equal amount of instances on each execution zone and (3) analysing an event tracelog to determine the best placement for the first batch of requests, assuming this placement is close to ideal for the entire tracelog. Next, we use the expected load to assign an initial amount of session slots to each service instance. The expected load can be derived from a tracelog provided by the user or must be specified as input parameter in case no tracelog is available.

10.3.1.4 Running a simulation

Once we've constructed a topology containing execution zones, clients and services, we prepare a simulation run. We use CloudSim [CBD11], a framework for modelling and simulating Cloud Computing infrastructures and services, to perform the event-driven simulations. We extended CloudSim's data centre objects with logic required to send, process and forward service requests. CloudSim also contains a centralized broker object which we extended with the FUSION orchestration logic. This broker contains knowledge of the network nodes (service location, available slots, edge capacity, …) as dictated in the graph model.

This modified CloudSim executable requires (amongst others) the location of the tracelog, location of the file containing the previously constructed graph model, a selection and placement algorithm as well as the amount of times the placement algorithm must be executed.

The CloudSim broker reads the tracelog provided by the user to schedule each event on the CloudSim simulation. When the tracelog contains events for more clients or services than the number specified by the user in the configuration file from step 2, then the broker will aggregate events in the tracelog. *N* clients in the tracelog are aggregated into *C* equally sized groups, where *C* is the amount of clients specified in the configuration file. The same logic applies to the services found in the tracelog. This allows a tracelog to be used as input for simulations with any amount of clients or services.

10.3.1.5 Sending a service request

Every scheduled event from the tracelog triggers one of the client nodes to send a service request for the attached service name. This request is forwarded to the broker which executes the service selection algorithm provided by the user. This algorithm must select an execution zone with one or more session slots available for that service as well as a path from that client to the selected zone with enough bandwidth to carry the traffic for that session.

The algorithm returns both the path and destination zone to the broker which then reserves these network and server resources for that session. The bandwidth on the selected path and session slot consumed by that session can not be used by consecutive requests until the client or server end the session. If no path with sufficient bandwidth or server with available slots for that service can be found, then the algorithm must alert the broker which will reject the request.

10.3.1.6 Evaluating the service placement

When reading the tracelog to schedule service request events, the broker also calculates the time interval in between consecutive runs of the placement algorithm specified by the user. After each interval the broker will schedule a placement evaluation event which pauses the current simulation and triggers the placement algorithm to run. Any placement algorithm is able to use the broker's database of measurements collected during the simulation up until the algorithm is executed. Using this knowledge, a placement algorithm is expected to generate alternative service placements by adding, deleting or changing the location of service instances, as well as upscale or downscale the session slots of any instance. The output is the amount of generated solutions as well as the location of the files containing the state of each solution.

The simulator will then execute a partial simulation for each solution as starting placement, processing only requests which occur between the current time and the next scheduled placement evaluation. At the end of each partial simulation the results are gathered and used to calculate the profit gained if this placement were to be used until the next placement evaluation. The broker selects the best solution, makes the required changes to the current graph model to match the new service placement and continue the main simulation.

10.3.1.7 Gathering results

The simulator gathers statistics about the generated load, resources used, costs made and profit gained. This is done for each simulation run and multiple runs are combined into averaged statistics. This allows us to evaluate algorithms on multiple topologies using different configurations to find the overall best performing algorithm.

10.3.2 Service selection

The main goal of our algorithm is to minimize the bandwidth usage to reduce required resources and to maximize the amount of requests per second we can process to increase profit. We evaluate each selection algorithm by running a simulation on a network with 100 service resolvers, 5 execution zones, 3 client nodes and 4 services. We deploy 2 instances of each service on execution zones, which creates a total of 8 service instances in the network.

We use a tracelog of Video-On-Demand traffic patterns, obtained through measurements by Orange France, to drive the simulation and generate request patterns for the clients. This tracelog contains data measured over 31 days and includes requests for 5700 objects coming from 9000 different clients. Figure 11 illustrates the amount of requests generated per hour for exactly two days in the tracelog.

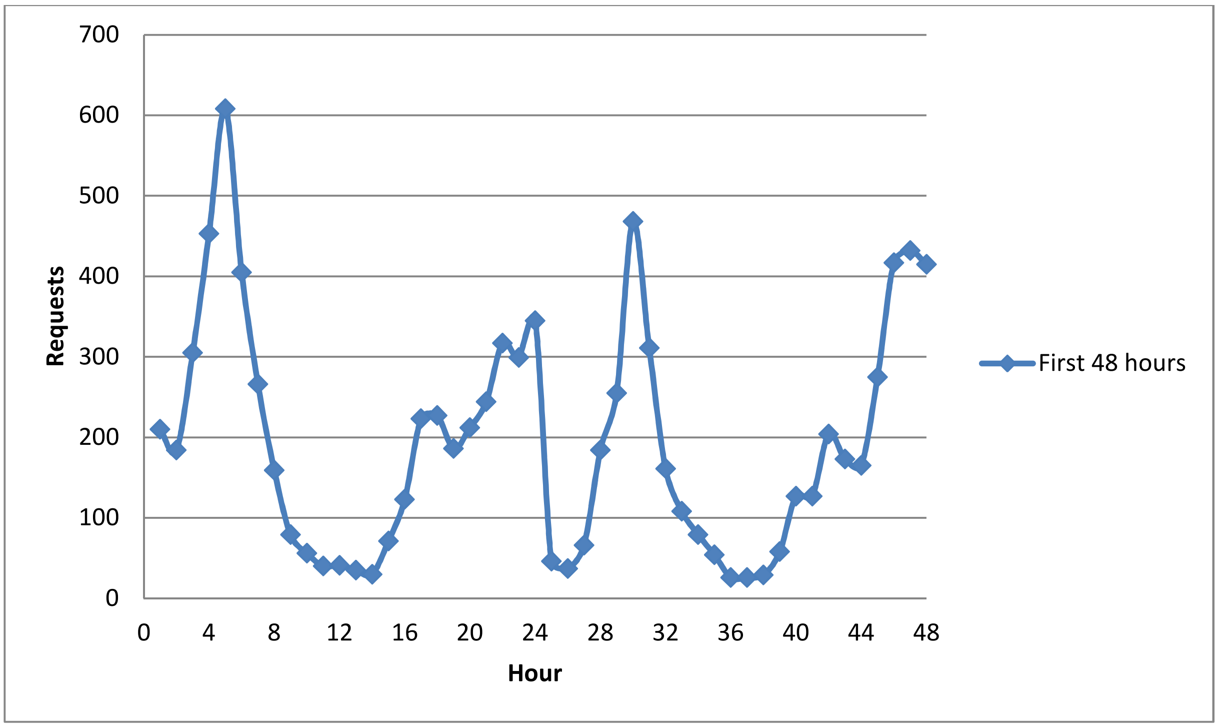

Figure 11: CDN requests per hour of two days measured by Orange France

Our generated networks only contain between 5 and 20 clients while the tracelog contains 9000 client records. In order to use this tracelog for any number of clients, we group the total amount of 9000 tracelog clients into X groups, where X denotes the amount of desired clients in simulation. For SS50 we require only 5 clients so all requests from the first 1800 client records will be seen as demand of client 1 in simulation, the next 1800 clients represent client 2, etc. We use the same logic to group the 5700 trace objects into the desired amount of simulation services.

The result of this simulation includes the amount of rejected requests as well as the bottleneck which caused a request to be rejected. We wish to minimize both the amount of rejected requests and the bandwidth usage.

Table 4 shows the performance of each algorithm. Using the Shortest Path Only algorithm we process the least requests, as expected, due to alternative network paths to servers not being used. The large amount of rejected requests renders this algorithm less suitable for practical use. The other two approaches perform almost equally well. Closest Available finds the shortest path with sufficient bandwidth available towards a service instance with available session slots. The Equal Distribution does also finds paths with sufficient bandwidth but does not prioritize the shortest path. Instead, it aims to have equal load on each path and service instance until all resources are full. Although both approaches allow an almost equal amount of requests to be processed, the Closest Available approach requires less bandwidth usage as it prioritizes shortest paths first.

Table 4: service selection results for all three selection algorithms

Shortest Path Only	Rejected due to bottleneck in:			
	Network	**Server**	**Processed requests**	**Bandwidth used**
	43458	20913	43841	1157875
Closest Available	**Rejected due to bottleneck in:**			
	Network	**Server**	**Processed requests**	**Bandwidth used**
	25061	26198	57066	2934105
Equal distribution	**Rejected due to bottleneck in:**			
Alpha α (used in variance)	**Network**	**Server**	**Processed requests**	**Bandwidth used**
0.0	24259	27081	56927	3154347
0.5	24551	26861	56849	3192334
1.0	25093	26251	56949	3039816

10.3.3 Service placement

The service placement algorithm in the orchestration layer comes into play when the available resources are insufficient to handle the current load, for example when too many requests are being rejected due to bottlenecks. A service placement algorithm aims to maximize the amount of requested requests by changing the service placement and server resources. If this was our only concern, we could deploy a service instance on each available execution point and configure each instance to have infinite session slots. While upscaling resources or deploying additional instances might increase the amount of processed requests, it also comes with a monetary cost. A trade-off must be found between the cost of deployed services and the profit made from processing requests.

For these reasons we developed a cost model which accounts for the following costs:

- Cost per Gbps: how much must the network provider pay for each Gb consumed on the network per second?
- Provisioning cost: the price per second to provision a service instance on an execution point. This does not include session slot costs yet and the instance can not process requests without slots.
- Cost per session slot: the price for the resources assigned to a service instance to process one request connection.
- Profit per request: the money received for each processed request.

By deploying more resources we can increase the amount of processed requests but this also induces more resource costs. It is up to the service placement algorithm to consider both profit and costs to find an optimal deployment. The solution which generates the most profit after deducting all costs is the best option and will be used to reconfigure the service deployment.

Analysing the performance of our first approach algorithms is a difficult challenge as we generate a set of new deployment configurations and use an experimental simulation to try and evaluate the quality of these solutions. Further study is required to identify the problem in the current deployment solution and find the best solution which optimizes profits.

The reaction time of the service placement algorithm can significantly improve the profit made; the faster we remove bottlenecks in the network, the less requests will be rejected because of it. However, reconfiguring the service placement more frequently reduces the amount of information gathered on the quality of the active placement. This can lead to poor placement decisions and induce higher costs.

Table 5: service placement performance in function of frequency

	Rejected due to bottleneck in:				
Evaluations during sim.	**Network**	**Server**	**Accepted**	**Bandwidth**	**Profit (€)**
0.0	16942	29464	62623	3615191	9074
1.0	17959	25256	66220	3714794	9693
2.0	17548	22534	69390	3729564	10341
3.0	17468	20948	71373	3685925	10797
4.0	18243	19868	71695	3762294	10808
5.0	18261	20777	70822	3642196	10787
6.0	17058	19474	73271	3783065	11146

Table 5 contains simulation results in function of the service placement frequency. The more frequent we reconfigure the service placement the faster a bottleneck can be removed but also the less information our algorithm has to base its decisions on. We observe that the amount of accepted requests increases as we reconfigure more frequent, although the performance improvement is less than linear. The more frequent we reconfigure our deployment, the less profit we gain from this, up to the point where we barely notice any difference when reconfiguring between three or five times during simulation.

Although these first results still require further study, we can already see that there is room to improve for the service placement algorithm.

10.4 Future work

Our current placement approach evaluates placement configurations by running simulations. This is a time-consuming process which means the reaction time of the placement algorithm is limited. On top of that, there is no guarantee that the generated configurations contain a solution for the problem which caused the bottleneck.

In future work we will develop a method to accurately detect the cause of network or server bottlenecks. We then create a model to find and generate solutions to solve that problem. We will also evaluate new algorithms which combine both upscaling/downscaling and adding/removing instances, so that one solution can mitigate both server and network bottlenecks.

11. DYNAMIC INTER-DOMAIN RESOLUTION

In this section, we first present a general mathematical model for the dynamic server selection problem. Then we show how to use this model in a practical application - the Source-to-Screen (S2S), a real-world cloud-based video processing service, according to the FUSION architecture.

In general, the goal of the server selection is to direct user requests to suitable execution zones (EZs). In this work, we define a utility function corresponding to user satisfaction. In detail, we use latency between a user and an EZ as a metric to measure that user satisfaction. We focus on a multi-objective optimization problem in which we first guarantee *max-min fairness* between users and then *maximize* the total utility of all users. We also consider a trade-off between the data transit cost (between resolution domains) and the performance (total utility) of users. We model the problem as a linear programming formulation.

11.1 Problem description and mathematical model

11.1.1 Problem description

- Inputs: estimated user requests, network performance model (e.g. latency, link bandwidth), data transit costs and resource constraints (e.g. number of available session slots at each EZ).
- Assumption: we use a centralized model in which each service resolver (SR) has all input information. Time is divided into fix-length windows, and the SR runs the optimization formulation at the beginning of each window time. Based on the results, we know how to map user requests to EZs.
- Objective: maximize the performance (total utility) of users while achieving max-min fairness between users. The objective also considers the trade-off between the performance and the data transit cost.

11.1.2 Utility function

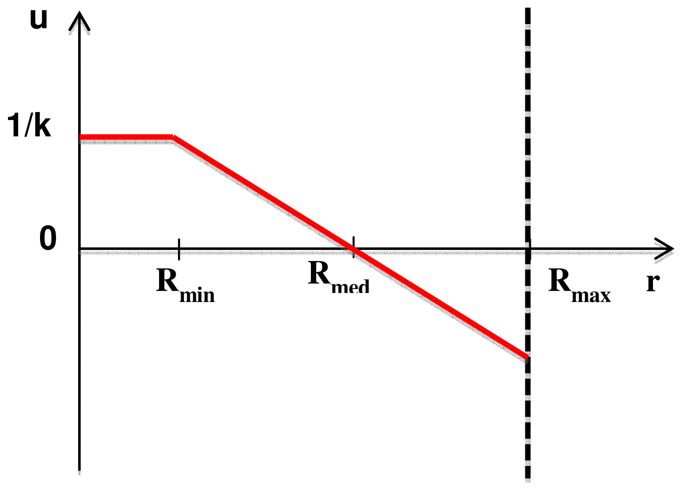

Figure 12: Utility function vs. response time

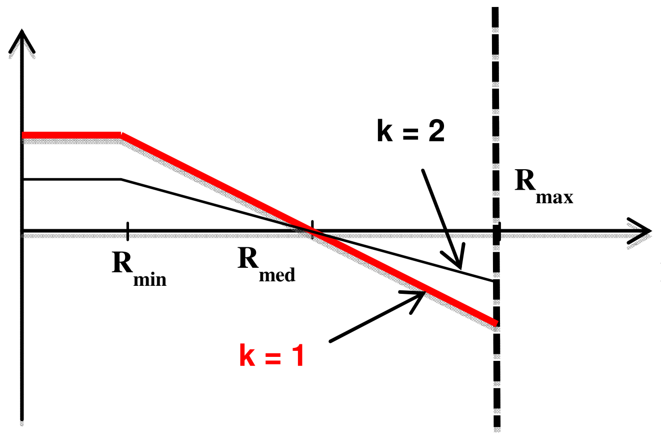

Figure 13: Utility function with different "k"

As shown in Figure 12, the utility function should express the following meanings:

- If $r \le R_{min}$, users are very happy. Depending on service type, we can choose an appropriate value of R_{min}. For some services, even if reducing more the response time cannot improve QoE, therefore the utility keeps the same value if $r \le R_{min}$. For example, voice over IP requires $R_{min} = 20\,ms$ [Wiki]). However, for other services like Web, the shorter the response time is, the better QoE the users get, so in this case we can set R_{min} = 0.
- If $R_{min} < r \le R_{med}$: the utility value is positive, meaning that the users are quite happy. However, the user satisfaction is reducing when the response time is increasing. We call R_{med} as the expected response time.
- If $R_{med} < r \le R_{max}$: the utility value is negative but the QoE is still in an acceptable range.
- If $R_{max} < r$: the utility value is $-\infty$ meaning that we consider the service request is blocked.

To satisfy the above requirements, we can define a utility function as follows (similar to but more complicated than the one in [NOMS08]):

$$u(z) = \frac{R-z}{kR} \qquad (1)$$

where:

$R = R_{med} - R_{min}$ and

$$z = \left\{ \begin{array}{ll} 0 & \text{if } r \le R_{min} \\ r - R_{min} & \text{if } R_{min} < r \le R_{max} \\ -\infty & \text{otherwise.} \end{array} \right\} \qquad (2)$$

A constant $k \ge 1$ is used to indicate the importance level of the user request. Large value of *k* means that the request is less important. By changing the values of *k, R_{min}, R_{med} and R_{max},* we can control the shape of the utility function (Figure 13).

11.1.3 Optimization Formulation: Linear Program (LP)

The objective of the optimization formulation is to achieve max-min fairness between users while maximizing the total utility. The objective also considers the trade-off between the performance and the service deployment cost. The algorithm works in two steps:

- Step 1: the objective function is to maximize the minimum user utility. In this step, we guarantee that the solution achieves max-min fairness between all users.
- Step 2: after the first step, let the value of the objective be *$U_{max\text{-}min}$*. Then, in the second step, we add the following constraint to the formulation:

min (u) >= $U_{max-min}$

By adding this constraint, the second step guarantees that the solution we find will be at least as fairness as in the first step. In addition, with the new objective: $max\ [\alpha \sum u - (1-\alpha) cost]$, the solution maximizes the total utility while considering a trade-off with the deployment cost (α is a parameter to say the relationship between the total utility and the deployment cost in the optimization objective).

Inputs:

- Estimated user requests: $\boldsymbol{d_{ij}}$: User *"i"* requests *"d"* session slots from service *"j"*.
- $\boldsymbol{b_{ij}}$: link bandwidth required by user *"i"* to get service *"j"*.
- $\boldsymbol{l_{it}}$: latency (response time) between user "*i*" and execution zone (EZ) "*t*".
- $\boldsymbol{C_t^j}$: available capacity (session slots) of service "j" at EZ "*t*".
- $\boldsymbol{BwCost_{ut}}$: unit bandwidth cost between user "i" at EZ "*t*".
- $\boldsymbol{R_{min}^{ij}}, \boldsymbol{R_{med}^{ij}}, \boldsymbol{R_{max}^{ij}}$: three response time thresholds defined in Figure 12.
- $\boldsymbol{R^{ij} = R_{med}^{ij} - R_{min}^{ij}}$

Main variable: $\boldsymbol{x_{it}^j} \in [\boldsymbol{0}, \boldsymbol{1}]$ - fraction of user *"i"* connects to EZ "*t*" to get service *"j"*. We assume that user "i" is a set of individual users that are grouped by near geography location.

LP formulation of Step 1:

$$U_{max-min} = max\ U \tag{3}$$

s.t.

$$\textstyle\sum_{t\in EZ} x_{it}^j = 1\ \ \forall (i,j) \in D \tag{4}$$

$$\textstyle\sum_{i\in user} d_{ij} x_{it}^j \le C_t^j\ \ \forall j\ \in service, t \in EZ \tag{5}$$

$$r_{ij} = \textstyle\sum_{t\in EZ} l_{it} x_{it}^j\ \ \forall (i,j) \in D \tag{6}$$

$$z_{ij} \ge 0\ \forall (i,j) \in D \tag{7}$$

$$z_{ij} \ge r_{ij} - R_{min}^{ij}\ \forall (i,j) \in D \tag{8}$$

$$u_{ij} = \frac{R^{ij} - z_{ij}}{k_{ij} R^{ij}}\ \forall (i,j) \in D \tag{9}$$

$$U \le u_{ij}\ \forall (i,j) \in D \tag{10}$$

$$cost_{transit} = \textstyle\sum_{t\in EZ} \sum_{ij\in D} BwCost_{it}\, b_{ij} x_{it}^j \tag{11}$$

$$cost_{transit} \le COST \tag{12}$$

$$x_{it}^j \in [0,1]\ \forall (i,j) \in D, t \in EZ \tag{13}$$

Explanation:

- Objective function (3): guarantee max-min fairness where U = min(utility) (as constraint (10)).
- Constraint (4): all the requests of a user *"i"* for a specific service *"j"* have been served.
- Constraint (5): each EZ has a limited number of available session slots dedicated for a service type. This may happen that some special services (e.g. need GPU processing) can only be

deployed at some specific EZs. This constraint is used to make sure the number of available session slots of EZ is enough to serve user requests.

- Constraints (6) are used to computed the average latency for the user "i" to get the service "j". Assume that the connection between users and EZs are a full mesh, it means that each user can connect to any EZs. For the inputs of the formulation, we simply remove all pairs of (user, EZ) that have latency which is larger than R_{max}^{ij}. This step guarantees that the latency for any user "i" to connect to any EZ "t" to get service "j" has to be less than R_{max}^{ij}.
- Constraints (7) – (8) ensure that $z_{ij} \geq 0 \; if \; r_{ij} \leq R_{min}^{ij}$, otherwise $z_{ij} \geq r - R_{min}^{ij}$.
- Constraint (9) is used to model the utility function (1). Note that when $r_{ij} \leq R_{min}^{ij}$, *z* can be at any value that is greater or equal to 0. However, thanks for the objective function, we would like to maximize the utility (9). It means that the formulation will choose a minimum value of *z*, or in other word, *z* is set to 0. Similarly, the formulation will choose $z_{ij} = r_{ij} - R_{min}^{ij}$ when $r_{ij} \geq R_{min}^{ij}$. Moreover, as mentioned in the constraints (6), a feasible solution does not allow any response time that is larger than R_{max}^{ij}. In other word, we can say that when $r_{ij} > R_{max}^{ij}$, the utility function is *–infinity*. In summary, the constraints (6) – (9) model exactly the utility function as defined in (1) (or in Figure 12). In the formulation, a user is group of individual users. For simplicity, we consider all group users are the same. If users are with different group sizes, we can multiply the right hand size of (9) with a parameter corresponding to the size of the group user "i". This helps to distinguish the group of users with different number of individual users.
- As shown in Figure 13, in the negative region, with the same value of response time r_{ij}, the request with low "k" gives lower utility (we call it a bad request). As the objective function is to maximize the utility of the bad request, the formulation will try to increase the bad request's utility by setting small value of r_{ij} for it. This intuitively means that the request with low "k" is more important.
- Depending on the users and the service type, we can set the appropriate values of the important level "k". Then, by playing with "k", R_{min}, R_{med} and R_{max}, we can control the shape of the utility function (Figure 13).
- Constraint (11) and (12) limit the transit cost between service routing domains. As shown in [ZZG+10, XL13], the linear transit cost that we use in constraint (11) is a good approximation for the 95-th percentile transit cost.

LP formulation of Step 2:

$$max[\alpha(\textstyle\sum_{ij \in D} u_{ij}) - (1-\alpha)COST] \qquad (14)$$

s.t.

(4) – (13)

$$U \geq U_{max-min} \qquad (15)$$

Explanation:

In this step, we add the constraint (15), where U is the minimum utility of users and $U_{max-min}$ is the objective value from step 1, to ensure that the solution should be at least as fairness as the max-min fairness in step 1. We keep the same constraints (4) – (13) and change the objective function as (14) where α is a parameter to say the relation between the total utility and the cost. If we set $\alpha = 1$, we get the solution with maximal total utility (while the cost is only restricted by the constraints (12)). On the other hand, if $\alpha = 0$ the solution we get achieve max-min fairness while minimizing the total transit cost.

There is always a trade-off between the utility and the cost. In case the service provider know how to choose the value of α, they simply run the formulation as step 2 above and get the optimal solution. However, if it is hard to estimate α, we have another way to give hint for the service provider to find a suitable solution. In general, given a solution, we can plot its cost and utility on a 2-D plane (Figure 14).

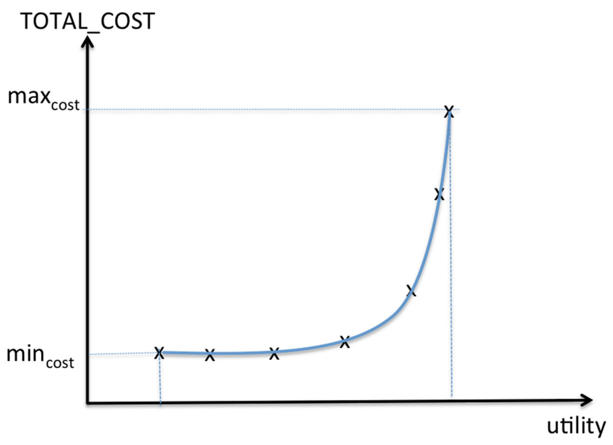

Figure 14: Trade-off between utility and cost.

By setting $\alpha = 1$ and $\alpha = 0$, we can find two solutions called max$_{utility}$ and min$_{cost}$, respectively. From max$_{utility}$, we can get the corresponding cost called max$_{cost}$. Then, we set many STEP_COST points in between the range [min$_{cost}$, max$_{cost}$]. Next step, in constraint (12), we set the value of COST to be equal to each STEP_COST point. We set $\alpha = 1$ and find the corresponding total utility for each value of the STEP_COST point. Depending on the granularity of the graph and how much time we can pay for computation, we can choose a suitable number of STEP_COST points. Finally, we get a trade-off relationship of the cost and the utility as in Figure 14. Based on this figure, the service provider can easily choose a solution with their desired trade-off.

It is noted that the optimization formulation above is a linear programming model; hence it can be solved efficiently. The number of variables x_{it}^{j} in the LP problem is $|I| \times |T| \times |J|$ where $|I|$ is the number of users, $|T|$ is the number of EZs and $|J|$ is the number of service types. Since $|T|$ and $|J|$ are usually much smaller than $|I|$, the worst case complexity of the LP problem will be $O(|I|^{3.5})$ [LPWIKI].

Relationship between server placement (WP3) and server selection (WP4) algorithms:

The objective of the service placement is to find which service instances to be deployed in which EZs. Given this deployment, we know the actual session slots at each EZ. The server selection phase is based on these real resource constraints to allocate user requests to appropriate EZs. In general, the formulations that we used in the service placement and the service selection are quite similar. We list in below the commons and the differences between them.

Similarities between service placement and service selection:

- They share the same idea of utility function.
- The algorithm works in two steps to achieve max-min fairness between users while maximizing the total utility. The algorithm also considers the trade-off between the performance and the cost.
- The constraints in the optimization formulation are similar, except the ones relating to the cost (constraints (11-12)).

Differences between service placement and service selection:

- The service placement considers the deployment cost of service instances in EZs while the service selection cares about the data transit cost between multi-domains.
- The capacity constraints (5) in service deployment are about the capacity of the hardware resource at EZ. In service selection, these are the constraints on the available session slots of service instances in EZs. In general, the capacity of the service selection is less or equal to the one of the service deployment.

The service placement is executed less frequently than service selection. In the service placement, we assume that a user can connect to any EZs. However, in the service selection, to reduce the input size for the optimization, we can assume that each user can only connect to a subset of EZs.

11.1.4 Preliminary results of the LP model

We present in this section some preliminary simulation results of the LP model for the service selection. We use the input dataset of the Massive Open Online Course application. In general, the dataset includes 2508 data centres distributed in 656 cities on over the world. To map them into the FUSION architecture, we call each city as an execution zone. The user demand is modelled as Poisson process and is proportional to the population of the cities. We manually set the number of total available session slots so that they are enough to serve all user demands. The latency between users and execution zones are collected based on the Haversine distance, the shortest distance between two points around the planet's surface. We define the three latency thresholds R_{min} = 20 ms, R_{med} = 50 ms and R_{max} = 150 ms. All of these data and the cost can be collected from https://github.com/richardclegg/multiuservideostream.

11.1.4.1 Trade-off between the total utility and the transit cost

In step 2 of the algorithm, assuming that we do not know value of α, then we draw a graph to show the trade-off between the total utility of all users and the transit cost, which is explained in Figure 15.

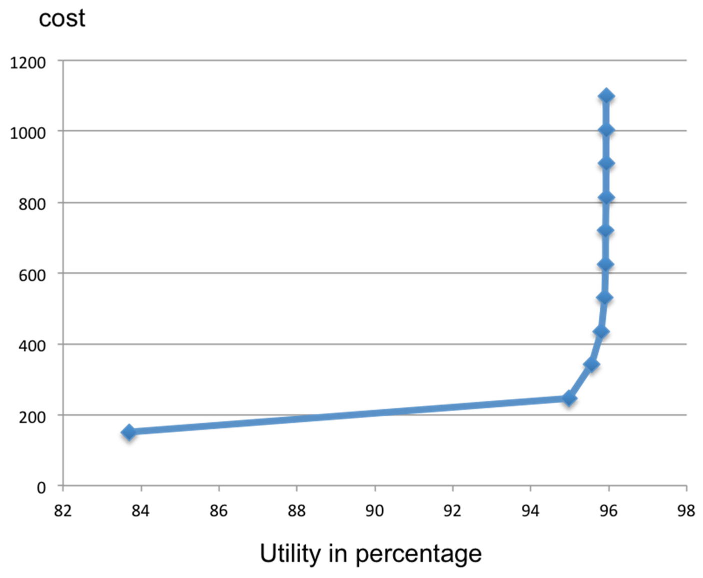

Figure 15: Trade-off between utility and cost

The utility is computed in percentage in which 100% means all users can get their best QoE (latency is less than R_{min}). As shown in Figure 15, the best utility we can achieve is 96%, which means that approximately 96% of the users can have latency which is less or equal to R_{min} = 20 ms. From the Figure 15, with the budget (cost) of 420 (units), we can get the maximum utility. On the other hand, with a limited budget of 180 (units), 84% of users can get their best QoE. Using this graph, the service provider can see the trade-off between the user utility and the cost and then they can choose the best operational point.

11.1.4.2 The benefit of max-min fairness

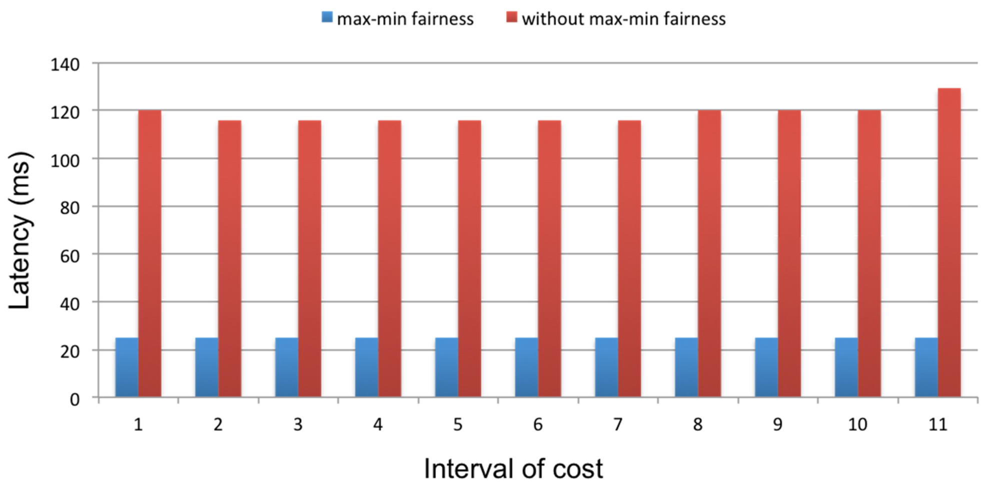

Figure 16: Latency of the worst user with and without max-min fairness

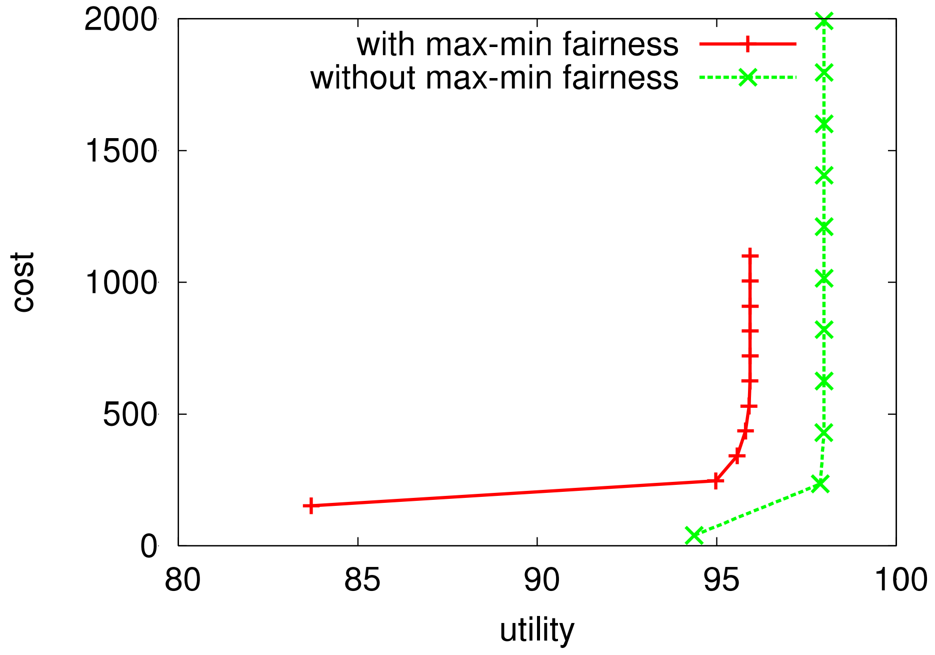

Figure 17: Total utility with and without max-min fairness

Figure 16 shows the comparison between the case if we consider max-min fairness (step 1 of the algorithm) and the case we simply maximize the total utility (ignore step 1 in the algorithm). The x-axis is the interval of cost ranging from the maximum and the minimum cost, which correspond to each step in Figure 15. As shown in Figure 16, with the max-min fairness constraint, the worst user still can get a good QoE (latency is less than 25 ms) while without max-min fairness, some users suffer from high latency (but it is always less than R_{max}). However, with max-min fairness, it can lose some benefits on the total utility and the cost. Figure 17 shows the comparison between the total utility with and without max-min fairness. The solution without max-min fairness can provide the best QoE for 98% of users, while it is 96% with the case of max-min fairness. On the other hand, with the minimum costs of 151 (units) and 40 (units), the total utility can be achieved are 83% and 94% with and without max-min fairness, respectively. Based on these simulations, we can see the advantages of max-min fairness in guaranteeing good QoE for all users with a small penalty compared to the case without max-min fairness.

11.2 Server selection within Source-to-Screen (S2S) application

11.2.1 Source-to-Screen application

To evaluate FUSION with a practical application we adapted Source-to-Screen (S2S), a real-world cloud-based video processing service, according to the FUSION architecture. S2S is provided by the British Association of Films and Television Arts (BAFTA) as a subscription-based service to film producers around the word who want to transcode their video files in formats compatible for video streaming. A typical service transaction involves three steps:

- The upload of a video file from a user to the S2S servers
- The processing of the input video file to produce a transcoded version of it.
- The download of the transcoded video file back to the user.

The uploading and downloading time depends on the size of the video files. Figure 18 shows the distribution of files sizes for the uploaded videos. The average file size has been 4.2 GB and the median 1.9GB.

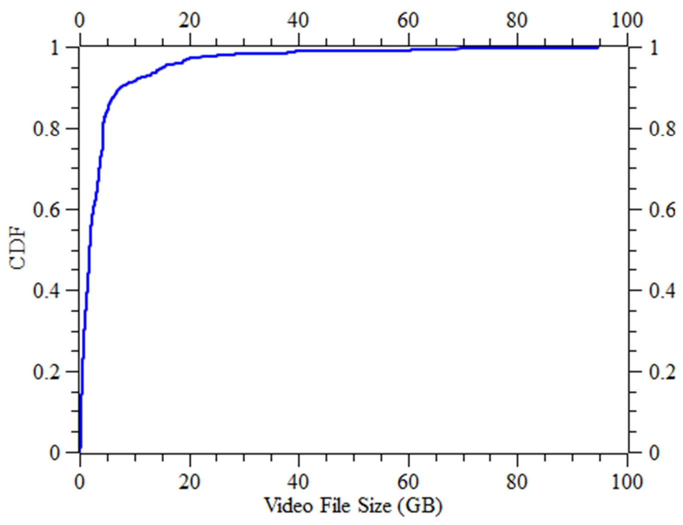

Figure 18: CDF of video file sizes uploaded to S2S

The processing time depends on the frame rate and the duration of the video files. Figure 19 shows the distribution of total processing time.

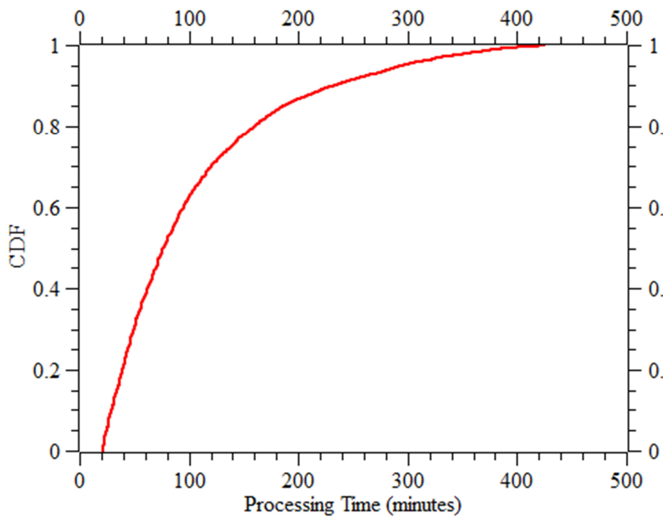

Figure 19: CDF of the video processing time

Figure 20 shows the user distribution. In 2014 S2S received 23,943 transcoding requests from 1,700 different users across 48 different countries. About 50% of the users reside in the UK and in the US.

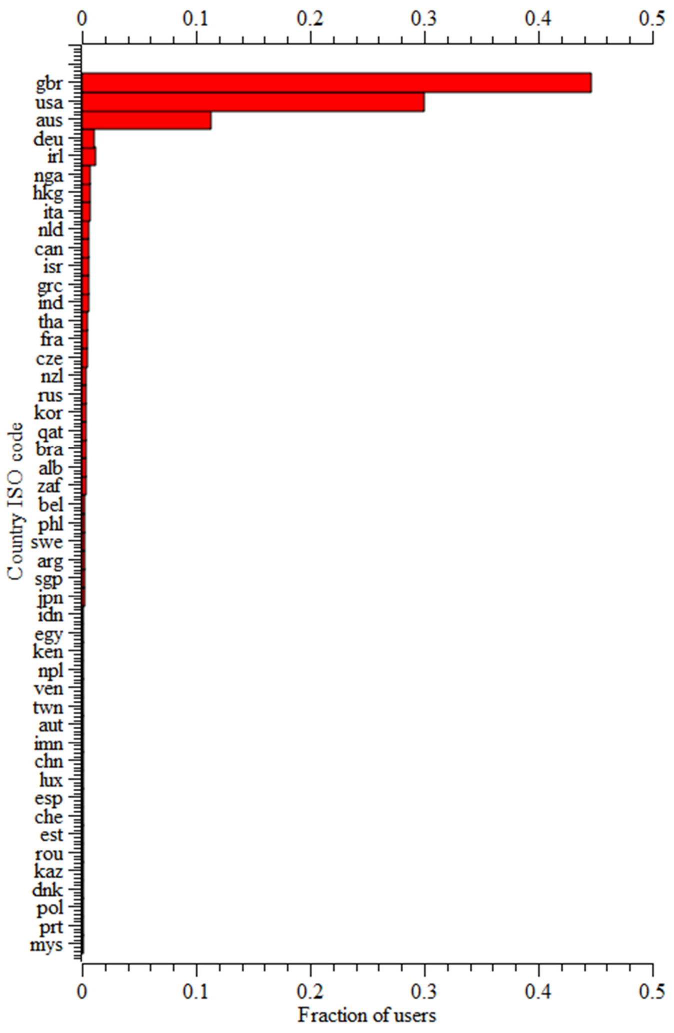

Figure 20: User distribution

11.2.2 Implementation of S2S within FUSION architecture

We implemented S2S on top of Amazon Web Services (AWS) cloud computing infrastructure, which provides the necessary components to capture the requirements of the FUSION service routing architecture. Figure 21 shows a high-level representation of the FUSION architecture.

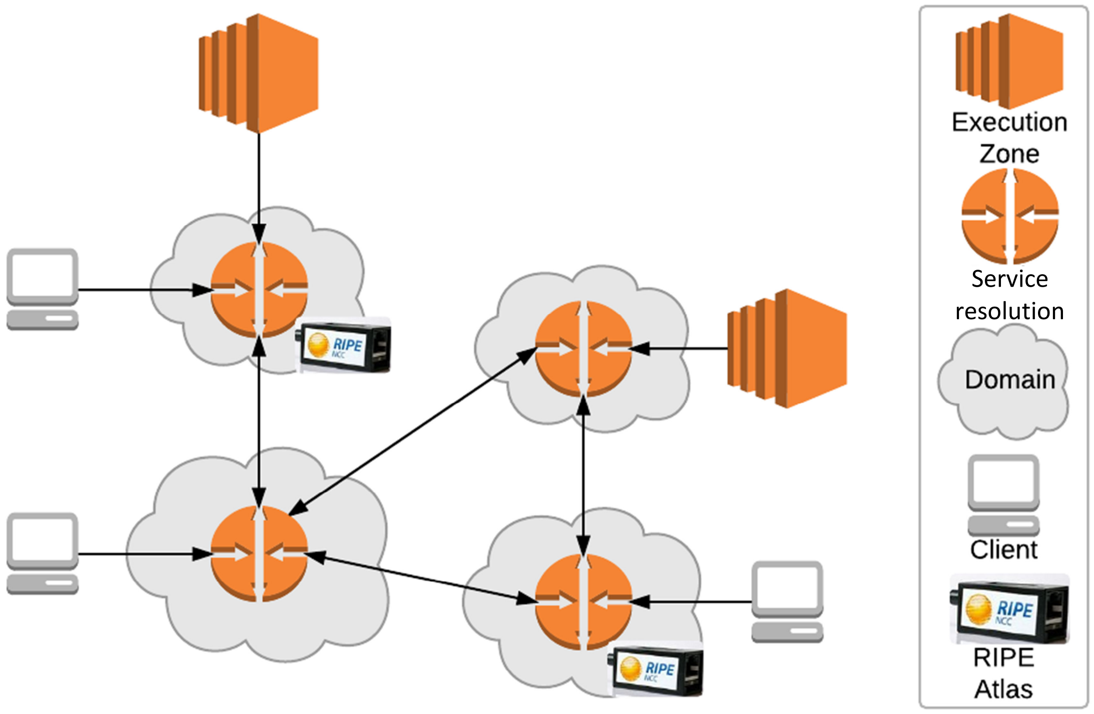

Figure 21: High-level mapping of S2S to FUSION architecture

Each **execution zone** is a collection of physical computational resources in a specific location, such as a data centre. Each *execution zone* hosts a number of service instances that can handle a number of requests in parallel, depending on the allocated resources. The number of possible parallel requests corresponds to a number of *session slots* that are available.

The **service resolutions** are responsible for routing client requests to the appropriate execution zones that can serve the request more efficiently. Service resolutions need to maintain information on the available session slots in each execution zone and monitor the network conditions between the clients and the execution zones.

To implement the FUSION's service routing we use the following AWS components:

- **Elastic Computing Cloud (EC2)** offers the ability to lunch virtual servers at different geographical locations called EC2 regions. Therefore EC2 region is a concept similar to FUSION execution zones. Currently there are 9 different EC2 regions shown in Table 6 EC2 regions. Each EC2 instance within a region corresponds to a service instance in FUSION. Each region has a limit of maximum number of EC2 instances that can be launched. The cost of each EC2 instance differs by region.

Table 6 EC2 regions

ap-northeast-1	Asia Pacific (Tokyo)
ap-southeast-1	Asia Pacific (Singapore)
ap-southeast-2	Asia Pacific (Sydney)
eu-central-1	EU (Frankfurt)
eu-west-1	EU (Ireland)
sa-east-1	South America (Sao Paulo)
us-east-1	US East (N. Virginia)
us-west-1	US West (N. California)
us-west-2	US West (Oregon)

- **Cloudwatch** is a monitoring service for EC2 instances that allows monitoring metrics such as CPU utilisation, data transfers and disk usage. Cloudwatch is used by our service resolution implementation to calculate the available session slots at each execution zone.
- **Route 53** is a DNS service that allows implementing different types of routing including latency-based routing, and geo-DNS. We use Route 53 to route the initial client requests to the closest service resolution.

To implement the network monitoring required by service resolutions we use RIPE Atlas, a global network of probes that measure Internet connectivity and latency. Figure 22 shows the coverage of RIPE Atlas. We can see that Atlas provides a particularly good coverage for the countries where the majority of the S2S reside. We collect periodically measurements from Atlas probes within ISPs from where requests are received and we store the results in a MySQL database. The database is accessible by all the service resolutions and they use it to pull data on network latency between a client and each EC2 region.

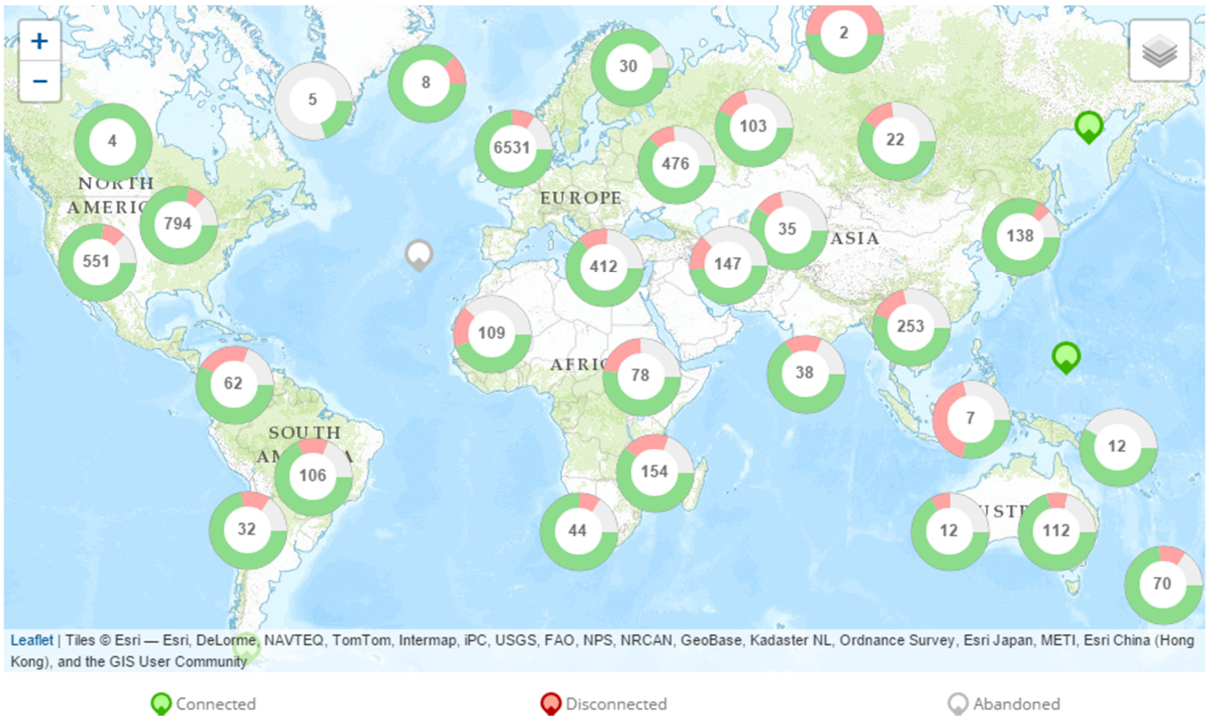
NORTH AMERICA
SOUTH AMERICA
EUROPE
AFRICA
ASIA
AUSTRALIA
4
794
551
5
8
6531
30
2
476
103
22
35
138
412
147
109
253
78
38
62
7
12
106
154
32
44
12
112
70
Leaflet | Tiles © Esri — Esri, DeLorme, NAVTEQ, TomTom, Intermap, iPC, USGS, FAO, NPS, NRCAN, GeoBase, Kadaster NL, Ordnance Survey, Esri Japan, METI, Esri China (Hong Kong), and the GIS User Community
Connected
Disconnected
Abandoned

Figure 22: Coverage of RIPE Atlas vantage points

In AWS there is no concept similar to service resolution. To implement service resolution, we allocate an EC2 service instance at each AWS region to have the role of the service resolution. Initially a client RESOLVE request is routed to service resolution at the geographically closest region. Each service resolution receives from cloudwatch data on the available service slots of its own region and exchanges this information with the servicer resolutions at the other regions.

Figure 23 shows the overall implementation architecture.

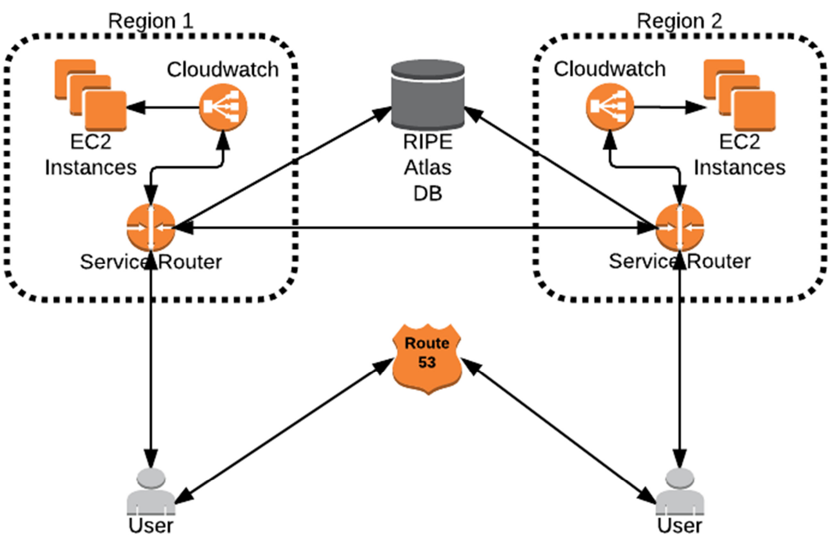

Figure 23: Implementation architecture based on AWS.

11.2.3 Impact of service resolution policies on performance and cost of S2S

Figure 24 shows the results, in terms of overall latency and cost (processing cost on Amazon AWS), of selecting servers based on lowest cost, lowest network latency and randomly. Overall service latency, in these plots, has been calculated as the time from when a user submits a video file for transcoding to when the results are delivered back over the network. As the S2S service, as currently implemented, transcodes an entire file before delivering the result to the user an approximation has been made for a real-time transcoding service. The minimum number of frames that can be transcoded at once has been determined, depending on the type of video encoding. The total number of frames in the submitted video has been divided by this minimum to determine the number of chunks in the submitted video that could be processed in a real-time fashion. The total file upload time, total transcoding time and total file download time has been divided by the number of chunks to calculate the service latency plotted in the graph.

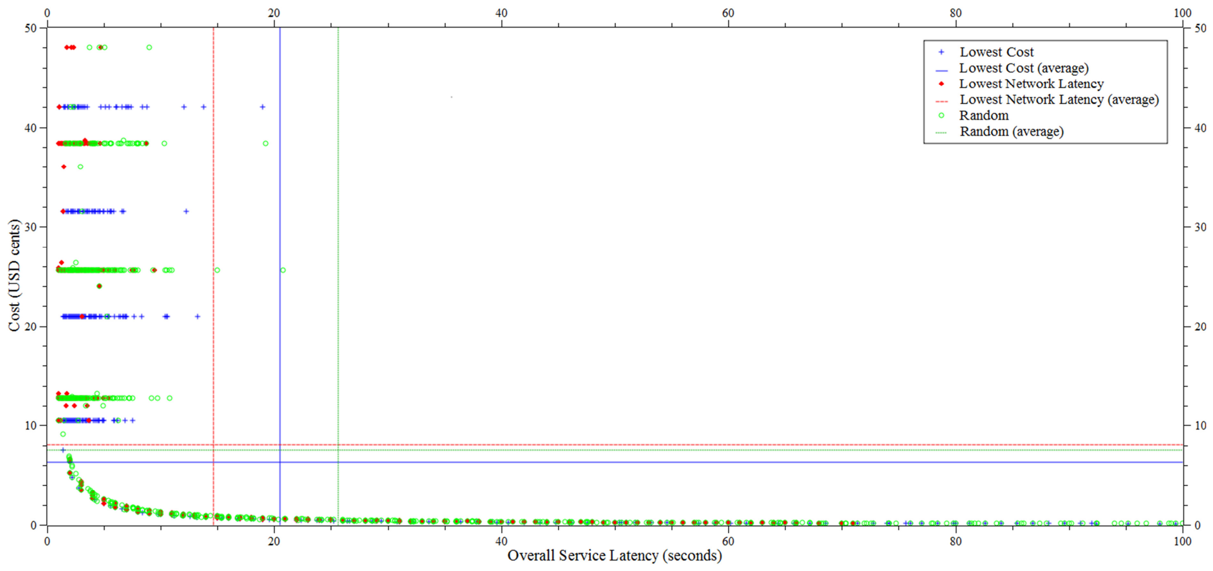

Figure 24: Impact of service resolution policies on S2S performance and cost

The graph demonstrates that service resolution policy affects both cost and performance of the S2S service. Selecting on lowest cost reduces mean costs but at the expense of service latency and vice versa. The next step is to evaluate the performance of the optimised service resolution policy using the optimisation formulation as presented earlier in this section.

We have implemented the server selection optimization model using CPLEX solver at each service resolution. The inputs for the optimization are collected by AWS components. For instance, the number of available session slots is collected from the Cloudwatch, the network metrics are from the network monitoring (RIPE Atlas). In addition, we have costs of data transit from different EZs based on different EC2 regions. After finding solutions, the server selection model stores the results in a forwarding table, which is used by the service resolution to direct client requests to appropriate EZs.

Evaluation results comparing optimised service resolution based on a combination of service and network metrics with the single metric resolution policies (Figure 24) are currently being collected.

12. SERVICE RESOLUTION PROTOTYPE

12.1 Emulation platform

To evaluate the different service resolution algorithms, we are building an emulation platform on the Fed4FIRE infrastructure [F4F]. The final plan is to showcase a running prototype of multiple service resolution domains, with execution zones running on geographically distributed locations. The operational details of the emulation platform are explained in WP5. The table below lists the current status of different steps that must be taken towards this WP4 prototype.

Action	**Comment**	**Status**
Topology	Network topology	Available: • BRITE generator • CAIDA database Planned for 2015: • Ongoing discussions at Orange Poland to expose network topology details to the consortium
Service demand	Popularity of services over time (e.g. diurnal patterns, longer-time trends) and place.	Available: • Poisson distribution • CDN trace from Orange France Planned for 2015: • Ongoing discussions at Orange Poland to expose demand patterns.
Service Placement	Placement of services in distributed locations, connected by network.	Available: • Virtual Wall with network emulation Planned for 2015: • Deployment on PlanetLab and/or geographically distributed experimental facilities; using the Fed4Fire framework
Request and resolution protocols	Message type and syntax	Available: • first version of interface specification • first version of message syntax • support for atomic services Planned for 2015: • incremental updates and refinements • support for composite services

12.2 Zone gateway

The zone gateway is responsible for the advertisement of service availability to the service resolvers.

12.2.1 OpenStack extension for VMs

At this moment in the project, there is a first prototype available. This prototype has been integrated in OpenStack IceHouse. In the current implementation, the sum of all available session slots of running instances, plus the number of session slots that could be offered in addition by deploying the maximum number of instances allowed.

This implementation allowed us to functionally evaluate the service resolver. The tight integration with OpenStack, because of the use of its Ceilometer monitoring framework, is however a drawback. Moreover, container support (especially Docker) is not available in the mainline software tree OpenStack and is not to be expected before the Kilo release in mid-2015.

For this reason, it was decided to start from scratch with a new implementation of the zone gateway, that is independent from OpenStack.

12.2.2 DCA-compatible version

As explained in D3.2, the Data Centre Abstraction (DCA) layer is an adaptor layer between the high-level FUSION zone manager and gateway; and the underlying DC and its corresponding DC management and orchestration (MANO) layer.

At the time of writing, we are still in the functional analysis phase. Design requirements identified so far are:

- operating on Docker containers
- support for zone-internal service discovery, preferably building on existing tools like Consul.io or Apache Mesos. This will probably need an extension of the current DCA layer
- interfacing with zone manager for composite service resolution support, especially the aggregation of session slots of individual instances that belong to multiple composite services
- support for zone-internal composite service resolution, using the ambassador pattern (as discussed in section 9).
- integration with load-balancers like HAProxy. This is challenging in the context of composite services, where the resource session slots of an individual instance must be mapped to session slots of multiple composite services.

12.3 Service resolver

The currently available prototype is built in Java, to allow for function evaluation. Over the next year, we plan the investigation of the scalability of this prototype for the size of the intended emulation set-up on PlanetLab, possibly moving to other server technologies if needed (e.g. node.js).

12.4 Prototype Evaluation

We have deployed an initial version of a service resolver, operating in a single domain and with a *shortest available instance* heuristic. In this first version, the service resolver has a full and up-to-date view on the network topology, the network load and the service instance load. This is realized using the discrete event-based simulation framework MASON. Each time a request arrives, an internal state manager adjusts the state accordingly (e.g. decreases the available number of session slots).

The prototype algorithm has been evaluated starting from the router-level topology that is continuously collected and updated by CAIDA [CAIDA]. We reduced this dataset to the nodes that are geolocated in France. IP-routers having the same geographical coordinates where aggregated and only links between the reduced subset were maintained. Finally, the link cost was set to the geographic distance between the routers. The resulting topology comprised 77 nodes and 98 edges.

Subsequently, we have randomly divided the nodes in 16 client nodes and 61 server nodes (execution zones). Service requests are generated from these client nodes following a Poisson distribution (mean λ) with exponentially distributed request interarrival times. Each client node thus contributes equally to the demand load, which we advocate by the fact that we simulate the network of one country where people are likely to have the same cultural preferences and diurnal patterns.

For each service a number of replica instances is deployed on randomly selected execution zones. Each request is assigned to a particular instance by a centralized service resolver, who selects the closest available instance. The session length was set identical for each service.

Simulations were carried out on the Virtual Wall emulation environment of iMinds [VWALL].

12.4.1 Average network cost

We first evaluated the impact of increasing demand for a single atomic service. The network cost is calculated as the cost of the link between the client node and the selected instance. Each service was deployed randomly on different execution zones, with a varying number of session slots. The service connection time was fixed to 10.

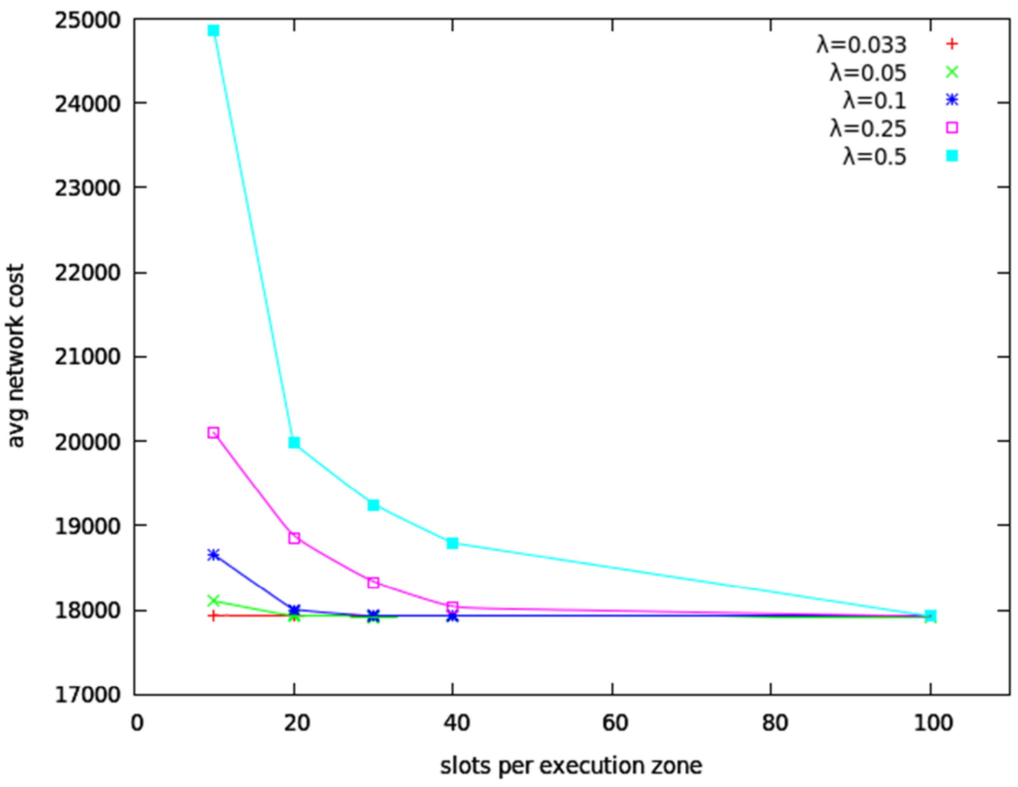

Figure 25: When each execution zone where the service is deployed announces more requests, the average network cost of connections between clients and servers decreases.

Figure 25 illustrates the trade-off between network cost and deployment cost. When each execution zone can handle only a limited number of parallel connections, the network cost increases with the demand because client requests are resolved to instances in more distant execution zones. When each execution zones announces abundant session slots, each request can be handled by an instance in the closest execution zone even in high demand scenarios.

In the evaluated configurations, there were only rejected requests in the scenario with the highest demand and the lowest capacity (λ = 0.5, 10 slots per instance). When the load increases, the network cost will saturate to a level where each instance is continuously occupied.

12.4.2 Average cost per client node

Figure 26 depicts the average network cost of requests per client node. The network cost from each client nodes increases with the demand, but the relative cost increase between the high-demand scenario (λ = 0.5) and low-demand scenario (λ = 0.033) varies greatly from 21 % (node 453) to 216 % (node 265). With other service placement algorithms, this difference might be smaller.

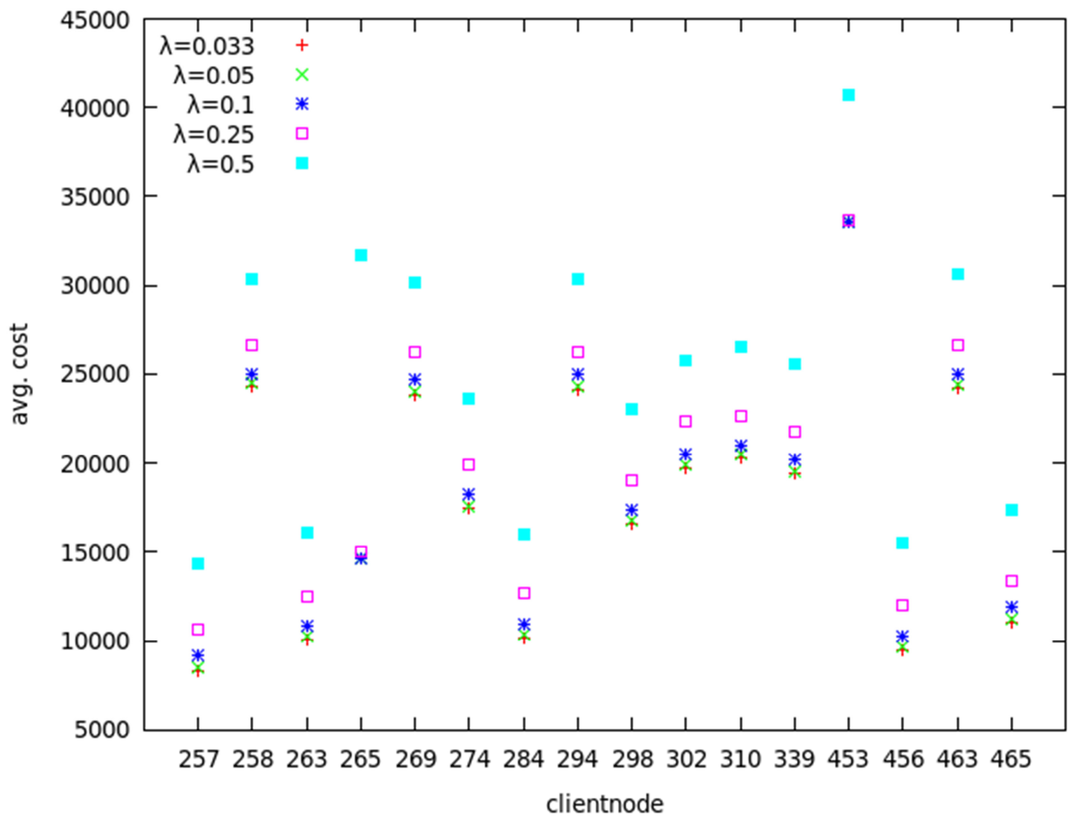

Figure 26: Each execution zone provides 10 session slots.

13. CONCLUSION

This deliverable presents the status of the design and implementation of the service resolution layer in the FUSION framework. This layer is conceived as an overlay for the IP stratum and provides name-based selection of instances; based on network metrics and service instance availability.

In this second year, we have progressed on the definition of a multi-domain organisation of the service resolution layer. We have identified the different policies that naturally arise in a setting with different stakeholders. From this study, we have defined all interfaces of the service resolution layer. The functionality is been translated into a comprehensive new protocol: the Networked Services and Resolution Protocol. This protocol is not only used for service requests; but also contains messages for exchange of resolution information and organisation of the resolution overlay.

An initial prototype implementation is available for a single domain, and the design of resolution algorithms has started.

For year 3 of the project, we have identified the following topics:

- The different roles in resolution policy enforcement and their potential impact on service resolution mechanisms and the overall FUSION architecture.
- The main research idea that will be followed here is to put limits on the distance (in terms of network hops and/or service routing domains) on the forwarding of service availability updates.
- Scalable mechanisms and algorithms for creating the intra-domain topology of service resolvers. Especially, we will consider whether there should be a (logical) full mesh between the different domains, or a spanning tree must be established. The trade-off between overhead and efficiency will be evaluated for different aggregation mechanisms of information on service availability and load.
- Service resolution algorithms that efficiently load-balance between execution zones based on a forecasting of the session slots and histogram of session length. The algorithms should be able to run in a distributed (inter-domain) environment. Heuristics will be developed that trade off network performance with session slot availability.
- Expanding the service resolution layer functionality through interaction with the orchestrator. This involves case where services want to do their own resolution , as well as composite services with dynamic service graphs. The main research challenge is to identify how much structural information can be embedded in the resolution layer without introducing complexity that would harm the scalability or performance of this layer.
- Develop a large-scale prototype of a multi-domain service resolution layer. Two set-ups are targeted: one with emulated network topology on the iMinds Virtual Wall, and one that runs on the actual Internet (PlanetLab).

14. REFERENCES

[APYA14] R. Alimi, R. Penno, and Y. Yang, “ALTO Protocol,” IETF Draft, draftietf-alto-protocol-27, March 2014.

[BRITE] Medina, A., Lakhina, A., Matta, I., & Byers, J. (2001). BRITE: An approach to universal topology generation. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2001. Proceedings. Ninth International Symposium on* (pp. 346-353). IEEE.

[CAIDA] http://www.caida.org/data/internet-topology-data-kit/

[CBD11] Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., & Buyya, R. (2011). CloudSim: a toolkit for modelling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, *41*(1), 23-50.

[CISCO-IOX] Cisco Fog Computing with IOx, http://www.cisco.com/web/solutions/trends/iot/cisco-fog-computing-with-iox.pdf

[F4F] Fed4Fire website. http://www.fed4fire.eu

[FPLS13] Frank, B., Poese, I., Lin, Y., Smaragdakis, G., Feldmann, A., Maggs, B., Rake, J., Uhlig, S., & Weber, R. Pushing CDN-ISP Collaboration to the Limit. SIGCOMM Computer Communications Review. 43(3), 34-44, 2013.

[HTTP2] Why is HTTP2 binary? http://http2.github.io/faq/#why-is-http2-binary

[LPWIKI] Linear Programming. http://en.wikipedia.org/wiki/Linear_programming

[NDGK12] E. Nordström, D. Shue, P. Gopalan, R. Kiefer, M. Arye, S. Ko, J. Rexford, and M. J. Freedman. Serval: An End-Host Stack for Service-Centric Networking. In Proc. 9th Symposium on Networked Systems Design and Implementation (NSDI’12), San Jose, CA, April 2012.

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

[NOKIA-LA] Nokia Siemens Networks – Liquid Applications. http://networks.nokia.com/portfolio/liquid-net/intelligent-broadband-management/liquid-applications

[NOMS08] D. Carrera, M. Steinder, and J. Torres, “Utility-based Placement of Dynamic Web Applications with Fairness Goals”, in NOMS 2008.

[MASON] Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, Keith Sullivan, and Gabriel Balan. “MASON: A Multi-Agent Simulation Environment**”**. In *Simulation: Transactions of the society for Modeling and Simulation International.* 82(7):517-527.

[PDBR14] L. Peterson, B. Davie, R. van Brandenburg. Framework for Content Distribution Network Interconnection (CDNI), IETF RFC 7336, 2014.

[PLA11] D. Lagutin, “Security: Packet Level Authentication and Pub/Sub Security Solution.”

http://wiki.fp7-pursuit.eu/uploads/d/db/PURSUIT_Aalto_Course-Lecture_8-Security_2-4.10.2011.pdf

[RTCP] RTP: A Transport Protocol for Real-Time Applications. RFC 3550.

[SAIL13] D.B.3 – Final NetInf Architecture http://www.sail-project.eu

[SOFTLAYER] SoftLayer. http://www.softlayer.com/

[VanJacobson] Jacobson, V., Smetters, D. K., Thornton, J. D., Plass, M. F., Briggs, N. H., & Braynard, R. L. (2009, December). Networking named content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies* (pp. 1-12). ACM.

[VWALL] http://www.fed4fire.eu/testbeds-and-tools/virtual-wall.html

[Wiki] http://en.wikipedia.org/wiki/Latency_(audio)

[WZKC13] Y. Wang, X. Zheng, L. Kang, B. Cheng. Research on Initial Filter Criteria of IP Multimedia Subsystem. In Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering (ICCSEE 2013), Hangzhou, China, 2013.

APPENDIX A: NSRP MESSAGE SYNTAX

NSRP consists of a set of binary messages and extensions. The general format of the messages can be seen in Figure 27.

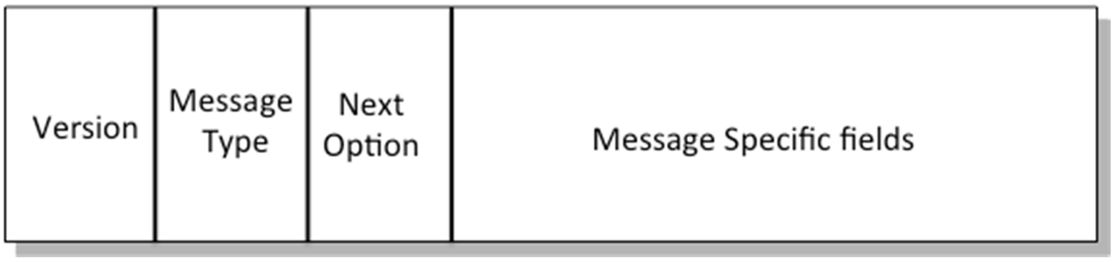

Figure 27: General NSRP message

Where:

- Version (4 bits): is the version of the protocol
- Message type (4 bits) indicates which type of message follows the generic header.
- Next Option

The majority of NSRP messages deal with resolution from names to locators. Names can have arbitrary size and belong to any name space of format. NSRP treats them transparently.

Locators can have several components: IPv4 or IPv6 addresses, port number, protocol number or prefix size. They can also be a tunnel end-point. These combinations are indicated by the following set of flags:

i. IPv4 or IPv6
ii. Port (included or not)
iii. Protocol (included or not)
iv. Tunnel (yes or no)
v. Prefix (yes or no)
vi. Multiple locator endpoint. This means that more than one locator is going to be used to identify the same service instance. For example a game service can have an audio server, a video server and a IM server.

Messages

There are 8 types of messages. Here we include a brief description and their fields:

1) REGISTER/UPDATE

   This message is sent to the local resolver to announce the availability of a service instance. It should be sent periodically to refresh the state in resolvers.

   Fields:

   - Size of Name (8 bits) in bytes
   - Name: Name of the service this instance is replicating
   - Send to source FLAG (1 bit): if equal to 1 then resolution should be sent back to the source of this message. This forces implementation of remote resolution.
   - Information Time to live (8 bits): Time in seconds this message is valid for. If not refreshed information should expire. If equals to zero, means to deregister an instance
   - Number of Locators

For each Locator:
- Locator type
- Locator
- Preferences

2) SUBSCRIBE

This message is used by service resolvers to subscribe for updates to services from other service resolvers. Each subscription has an expire time (Subscription Time to Live) and any updates should be forwarded until that time arrives. Subscriptions should be renewed before that.

Fields:

- Subscription ID
- Type of Element (Name, Catalogue, wildcard)
- Size of Element
- Element
- Subscription Time to Live

3) QUERY

This message is used to request resolution of names (1) Generated by a client that wants to resolve a name and (2) possibly forwarded by a service resolver to another resolver in the case of interdomain resolution.

Fields:
- Query ID
- Type of Element (Name, wildcard)
- Size of Element
- Element
- Query Time to Live
- More than one replica (flag)

4) DATA

A DATA message contains resolution for one name or a catalogue entry.
If it's catalogue entry (Subscription ID reply should be of a Catalogue Subscription there should be two locators: the representative locator and the resolver locator). The utility function of the service can be included as an option.

Fields:

- Subscription ID
- Number of Locators
- Locator Type
- Locators
- Preferences

5) QOE REPORT

QoE reports are sent from clients to their local resolver

Fields:

- Size of serviceID
- ServiceID
- Type of Source Locator
- Source Locator
- Type of Destination Locator
- Destination Locator
- Number of observations
- Type of Observation (Delay, Goodput, Loss)
- Observation

6) HELLO

This message is used by resolvers to build the subscription overlay. It is sent by resolvers to authenticate themselves before messages get exchanged

Fields:

- Resolver ID
- Signature

7) ERROR

This message contains an error and additional data for the error

Fields:

- Error Code
- Size of extra data
- Extra data

OPTIONS

The following options are defined:

1) Session slots: This allows a Zone manager to describe available and used service slots
   - Available session slots
   - Used session slots

2) Histogram of service: This allows a Zone manager to describe an histogram of service times that characterizes how the service is being used
   - Size of histogram
   - Histogram

3) Utility Function: This option allows a service to define an utility function. That is how varying QoS (delay, throughput, loss) affects the user utility of the service
   - Size of utility function
   - Utility function

4) Security: Instance authentication – This option allows a Zone manager to authenticate that is authorise to instantiate a particular services.

   This option allows a Zone manager to proof that he can announce instances of a given name

5) Security: Server describes permission – This allows a Zone manager to describe who can access the service. This is sent to all the resolvers so that they only resolve if authentication is successful. This should be used with the "Send to source" flag so that final resolution is made by the Zone manager (and potentially interact with the underlying network to switch on permissions.

6) Security: Client permission: This is the client counterpart to option 5. Client authenticates at the edge that it can access the service

7) Transparent Data (sent to instance, sent to client): This implements the "Invoke" functionality. Data is sent to the instance and this can return data back.

8) QoS constraints: This option allows a client to constraint the result(s) it receives. For example it may restrict to instances within X milliseconds.

9) Client information: IP address: This option allows a QUERY message to be sent "by proxy" for a node but indicating that the proximity corresponds to another IP address

10) Client information: GPS information
    This option allows a client to include GPS information in order to help the selection of closest instance

11) Routing hints: This option allows a client to restrict the returned instances to a subset. This happens when resolution is done remotely.

A mapping of which messages can carry each option is shown in Table 7.

Table 7: Options mapped to Messages

	Register	**Sub.**	**Query**	**Data**
Session slots	X			X
Histogram	X			X
Utility Function	X			X
Security: Instance Authentication	X			X
Security: Service permission	X			X
Security: client permission			X	
Transparent Data			X	X
QoS Constraints			X	
Client information: IP address			X	
Client information: GPS location			X	
Routing restrictions			X	

ERRORS

The following errors are defined for the error message

1) Name not found
2) Instance meeting QoS constraints not found
3) Access Authentication failed
4) Register Authentication failed
5) Resolver not responding
6) Badly formed packet
7) Timeout