Future Service Oriented Networks

www.fusion-project.eu

Deliverable D4.3

Final service-centric routing protocols, forwarding algorithms and service-localisation algorithms.

Public report[1], Version 2.0, June 2016

Authors

UCL David Griffin, Elisa Maini, Khoa Phan, Miguel Rio

Nokia Frederik Vandeputte

Orange Dariusz Bursztynowski, Andrzej Denisiewicz, Paweł Piotrowski

SPINOR Folker Schamel

IMINDS Piet Smet, Lander Van Herzeele, Pieter Simoens

Abstract This deliverable concludes the work on FUSION service resolution. We first characterize the geographic distribution of the data centres that are already deployed today and characterize the network between end-users and this existing infrastructure. We then perform a scalability analysis of the signalling overhead, using a discrete event simulation and several publicly available datasets and models. The results are compared with a more centralized, per-service model. This deliverable also presents a distributed resolution algorithm, which is validated through simulation. Finally, the deliverable details the architecture of the FUSION resolver and present a case study on WebRTC that demonstrates how FUSION resolution aspects can be applied in an ISP network.

Keywords service resolution, network measurements, system design, algorithms, WebRTC

[1] The public version of this report has some sections suppressed as they are currently under review for inclusion in scientific publications (or submission is planned thereof).

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
02/12/15	P. Simoens	TOC	0.1	Initial TOC
15/01/16	P. Simoens	Early draft	0.2	Bullet points
15/02/16	P. Simoens	Draft	0.3	Input on distributed resolution Input on geographical characterisation
15/03/16	P. Simoens	Draft	0.4	Input on network characterisation Input on service resolver design
15/04/16	P. Simoens	Draft	0.5	Scalability analysis added
10/05/16	P. Simoens	Stable version	1.0	Intermediate version
06/08/16	P. Simoens	Final version	2.0	Additional results and final conclusions

EXECUTIVE SUMMARY

This document is the final deliverable of WP4 “Service-centric Networking” of the FP7 project on “Future Service-Oriented Networks” (FUSION). The main objectives of this WP were:

1) to define, implement and validate a scalable service-centric networking stack
2) to develop optimized network management techniques for placement, duplication and relocalisation of services and content
3) to design and implement service-centric routers

Service resolution is only useful in case multiple service replicas are deployed throughout the Internet. We therefore study in section 2 the validity of the claim that fog/edge computing with distributed data processing capabilities located as close to the network edge as possible is required to provide low-latency services. The realization of this fog paradigm obviously entails significant capital and operating expenses to Internet Service Providers (ISPs) that could hinder the deployment of distributed cloud infrastructure and thus the adoption of FUSION resolution. However, our studies show that the already existing DCs may provide sufficient performance to deliver many high-performance applications, such as cloud gaming, to the vast majority of users worldwide. Thus, there are already today many feasible candidate sites to deploy instances on of demanding services.

With sufficient infrastructure available, two factors influence the placement and resolution decision: the performance of the infrastructure (studied in WP3) and the network metrics (scope of this work package). In section 3, we document the results of an extensive empirical study on network characteristics between end-users and this existing cloud infrastructure. This study focuses on the added value of ISPs assuming the role of service resolver, compared to an approach in which the resolution is based on over-the-top (OTT) monitoring information. One key question is the stability of routes between end-users and data centres over time, as well as how fast rerouting and interruptions can be detected by ISPs as well as OTT monitoring entities.

In section 4, we zoom in on the scalability of the resolution. Using discrete event simulation and models derived from realistic datasets, we study the overhead in terms of signaling information for two models. The first model is a per-service centralized model, while the second model assumes a distributed approach with generic resolvers deployed at strategic locations in the network.

Section 5 presents the internal architectural details of the designed FUSION resolver, fulfilling objective 3 described above.

In section 6, we present a case study on WebRTC, whereby FUSION resolution principles are applied into the operational network of Orange, a large European ISP.

The two annexes of this deliverable present the detailed results of the network characterization study and the resolver interface specification.

TABLE OF CONTENTS

1. INTRODUCTION

The main primitive of the FUSION service resolution layer is to resolve service requests to the locator of the most appropriate instance among the many service replicas 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.

This deliverable focuses around the following research questions.

Section 2: What distributed cloud infrastructure is available today?

One of the key premises of the FUSION project is the availability of many datacentres at dispersed locations. The performance of any service placement and instance selection algorithm is bounded by the number and distribution of execution zones. Obviously, there can be a chicken-and-egg problem: if there are only a limited number of execution locations available, this is a significant barrier for the development of next-generation demanding end-user services. Conversely, if there are no services needing distributed deployment, there is no incentive for network operators or cloud infrastructure providers to invest in additional cloud capacity, especially bearing in mind that the OpEx of distributed facilities is higher than that of only a handful of centralized sites. For this reason, it is important to gain insight in the infrastructure that is already available today and how well these sites are reachable by end-users.

Section 3: How important is network-awareness in the resolution process?

End-to-end network metrics such as latency, jitter and throughput can be monitored in an over-the-top manner (OTT). Commercial solutions like Cedexis or CloudHarmony provide benchmarks of CDNs and cloud providers worldwide. Statistics are crowdsourced by clients accessing HTTP pages with embedded scripts to measure network statistics to selected sites. The accuracy and timeliness of these datasets depends directly on the number of participating clients.

A key question is thus whether OTT measurement methods are sufficient for taking appropriate decisions to which service replica end-users should be directed. ISPs have a detailed insight in the performance of their own network, and on the BGP routing topology towards other Autonomous Systems (AS). This inter-AS routing is subject to changes (e.g. due to link failures) and traffic routing policies. In this section, we thus investigate the added value of having access to such detailed knowledge on the network topology and status.

Section 4: What is the most appropriate placement model for service resolvers from the perspective of signalling overhead?

In FUSION, a service selector is a service ID/name, so there is no overlap between services and it is feasible to have a single resolver for each service ID/name. There is thus an optimization problem to be solved: what is the most appropriate placement of resolvers. Distributed resolution, with each resolver responsible for a number of execution zones, may provide advantages in the amount of signalling information, while a centralized model is conceptually much simpler. In this deliverable, we formulate this trade-off and study which factors determine the position of the optimum in-between two extreme models.

Section 5: Architecture and implementation of the resolver

We present here the architecture and interfaces of the FUSION resolver. We include a discussion on how this resolver can access monitoring information.

Section 6: Distributed resolution algorithms

This section focuses on the key intelligence of the resolution plane: to decide which execution zone a user request will be redirected to. The polynomial algorithm allocates user service requests to execution zones based on utility while satisfying transit cost constraints. The method presented includes a scalable, low-overhead distributed model that is shown via simulations to lead to higher user utility compared to mapping user requests simply to the closest service replica.

Section 7: case study on WebRTC

WebRTC enables browser-to-browser real-time communication for services like voice calling, video chat and P2P file sharing, without the need of internal or external plugins. In this section, we present how the FUSION resolution architecture can be used to improve the Quality of Experience by selecting appropriate instances of TURN servers.

Before addressing this research questions, in the remainder of this introductory section we present the key functional concepts, final architecture and deployment considerations of the FUSION resolution subsystem.

1.1 Functional concepts

For the sake of clarity, we briefly recapitulate two key FUSION concepts in this section. More details on both concepts can be found in D3.3.

- A **session slot** is a lightweight abstract application metric for measuring the resource utilization and availability for a particular service instance. Specifically, it is a number representing the number of parallel sessions that can (still) be supported with sufficient QoS (i.e., deadline misses, latency, jitter, etc.) for a particular instance. For example, if an instance is reporting 4 available session slots, this means that it has still sufficient resources for handling 4 clients in parallel for the duration of the session. The number of instances and the allocated resources to each of these instances so as to provide sufficient service availability is governed by the orchestrator. Execution zones announce the current number of available session slots to the service resolvers to help drive the instance selection algorithms.
- **Utilities** are defined by the service provider in its manifest. Utility is described as a weighted combination *R* of metrics relevant for the service performance: infrastructure cost, network latency between clients and service instances, etc. The utility function is formulated using two different parameters. R_{min} denotes the value for which further reducing of R has no significant benefit on the service Quality-of-Experience. For example: for some voice services, reducing the latency below a given threshold will not further improve the perceived quality. The value R_{max} provides the maximum tolerable value of R: user-instance pairs with a value greater than R_{max} are not allowed.

1.2 Final FUSION Resolution Architecture

The final architecture of the FUSION resolution layer follows a **utility-based publish-subscribe** model, and is visualized in Figure 1.

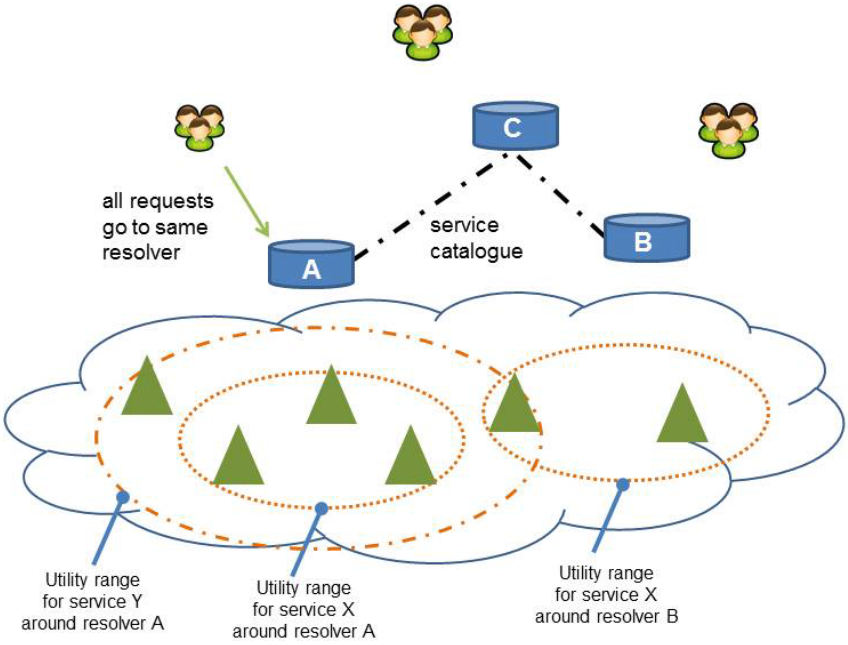


Figure 1 Conceptual view of the FUSION resolution layer

The resolution layer consists of one or more resolvers. Each service resolver is responsible for maintaining and managing information to create forwarding paths for queries to be resolved to execution zones (EZ) containing available instances of the requested service. Each resolver can handle requests for all services. When a client wants to resolve a service ID into a locator, it sends a query to the resolution system. Service resolution algorithms find the "best" instance amongst possibly many replicas distributed over the Internet.

These instance selection decisions are based on the merging of two factors: instance availability in the different EZ and network characteristics from the client to each one of them. These network characteristics may include QoS metrics, traffic policies, and so on.

1.2.1 Exchange of service availability information

The exchange of service availability information consists of two distinct steps: catalogue sharing and service subscription.

Catalogue Sharing: The Zone Manager[2] announces the service ID, the utility function and a representative locator to at least one resolver. This information is further injected into the catalogue which is shared between resolvers which organize themselves in an overlay. FUSION has not defined the exact means of communication between these resolvers, since existing approaches like multicast trees or distributed hash tables can be used. The service catalogue contains information on the services and contact information:

- list of service IDs per EZ. This list only contains the services that have been *provisioned* in this EZ and thus does not convey information on the actual *availability* of the service. For example, in the

[2] The Zone Manager is an agent of the orchestration layer that runs locally in an execution zone. We refer the reader to deliverable D3.3 for more information on the orchestration layer architecture.

case of on-demand provisioning (see D3.3), an EZ will list the service ID although no actual instance is running.
- IP address source of a given EZ to subscribe to for session slot announcements (see below)
- a possible second locator (e.g. port/IP tuple or ALTO locator) that can be used by resolvers to assess the utility of the service running in that EZ. This locator could e.g. refer to an evaluator service[3], or to the Zone Manager itself that includes the result of the evaluator services

Note that the information in the catalogue is not very dynamic as updates are only sent when a new service is created, all service instances have been deleted or there are significant changes in network connectivity (e.g. change of traffic engineering policy).

Service subscription: Whilst receiving catalogue messages, resolvers decide if they subscribe to updates from a given EZ. The feasibility of an EZ is determined based on the 'utility range' for a given service. This is shown in Figure 1: resolver A subscribes to three execution zones for service X, and to 5 execution zones for service Y – which has a larger utility range. Similarly, resolver B subscribes to two execution zones in its vicinity to receive session slot updates for service X. To obtain enough diversity of service availability, resolvers will try to contact close zones before, expanding the subscriptions to more distant zones until enough instances are found. After subscribing to a given service, a resolver will start receiving updates from that EZ on the availability of the service. The availability information is conceptualized as session slots.

For the sake of clarity, some details are not shown in Figure 1. For example, a resolver need not to subscribe to all execution zones that announce session slots of a specific service within the utility range of that service. It may opt to receive only updates from a subset, e.g. until enough session slots are aggregated, or to minimize inter-AS transit traffic costs for high-volume services.

1.2.2 Request resolution

Users send a request for a given service to the resolution subsystem, which returns a locator of the selected service replica. This locator can contain IPv4/IPv6 address, ports, protocol numbers and/or tunnel identifiers. 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 (e.g. TCP/IP tuple) 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.

At regular intervals, EZs update service resolvers on instance availability by reporting their available session slots. Instance selection decisions are thus based on the merging of three factors: instance availability in the different EZs, network characteristics from the client to each one of them and service requirements.

1.2.3 Resolver deployment

Users are grouped in *resolution domains*. Each resolution domain has (logically) a single resolver that handles requests for *all* services from *any* user in this resolution domain. Users can find the nearest resolver using the same DNS-based approach that is common in today's CDNs to redirect users to the nearest proxy [FKH15]. Note that DNS is only used for finding the nearest resolver, the resolution of a service ID to a particular instance in an execution zone is done by the resolvers themselves, which take into account session slot availability and network metrics. Alternatively, users can hard-code the IP location of the resolver, akin to how they configure their DNS server.

[3] Evaluator services are managed by the FUSION orchestration layer to assess service performance in a given execution environment (e.g. availability of hardware acceleration, multi-tenancy collision, etc.). We again refer to D3.3 for a profound explanation of the concept.

The resolution overlay can grow organically. In an early phase, orchestrators could act as resolvers to ensure reachability of their managed services. Over time, other parties could attach resolvers to the resolution overlay. One option, which we study more in the next sections of this deliverable, is that resolvers are operated by (larger) ISPs, who might want to impact the resolution process for various reasons:

- on-demand instantiation of services in micro-datacenters at the edge of the network, requires detailed information on the available resources as well as the network attachment point of the user. It is unlikely that ISPs expose this information to 3rd parties.
- reduction of inter-AS transit traffic costs by deploying some high-volume services in the own network, or by taking into account these traffic costs when selecting between suitable replicas in other ASes.

To keep full control of the load on some instances, resolvers may decide to hide the actual locators of services in a number of EZs and replace them with an ALTO Provider-defined Identifier (PID) before injecting the service announcement in the service catalogue. The PID is a representative locator for e.g. a subnet, an AS, a metropolitan area, that allows other resolvers to assess the potential performance of connections to instances running in that resolution domain. ALTO is an emerging IETF standard for dissemination of network level information between different business entities [6]. Operators expose cost maps, assigning cost values (e.g. routing cost) to one-way connections between PIDs. Other resolvers can then evaluate the feasibility of service replicas exposed by one resolver, without having full knowledge of the internal network or the operator policies. Using a mechanism akin to DNS forwarding, in these cases the service request will be forwarded between resolvers and only the final resolver will return the actual locator to the requesting client.

2. GEOGRAPHICAL CHARACTERISATION OF DATACENTRES

In this section, we formulate an answer on how well the geographical distribution of today's datacentres matches population density. In section 3, we present results on our measurements of network metrics to a subset of the datacentres worldwide.

The website datacentermap.com contains a registry of datacentres worldwide. By crawling this website, we were able to collect information about 3116 datacentres, see Figure 2. Statistics on the population per city were collected from geonames.org (see section 4.2.1 for more details).

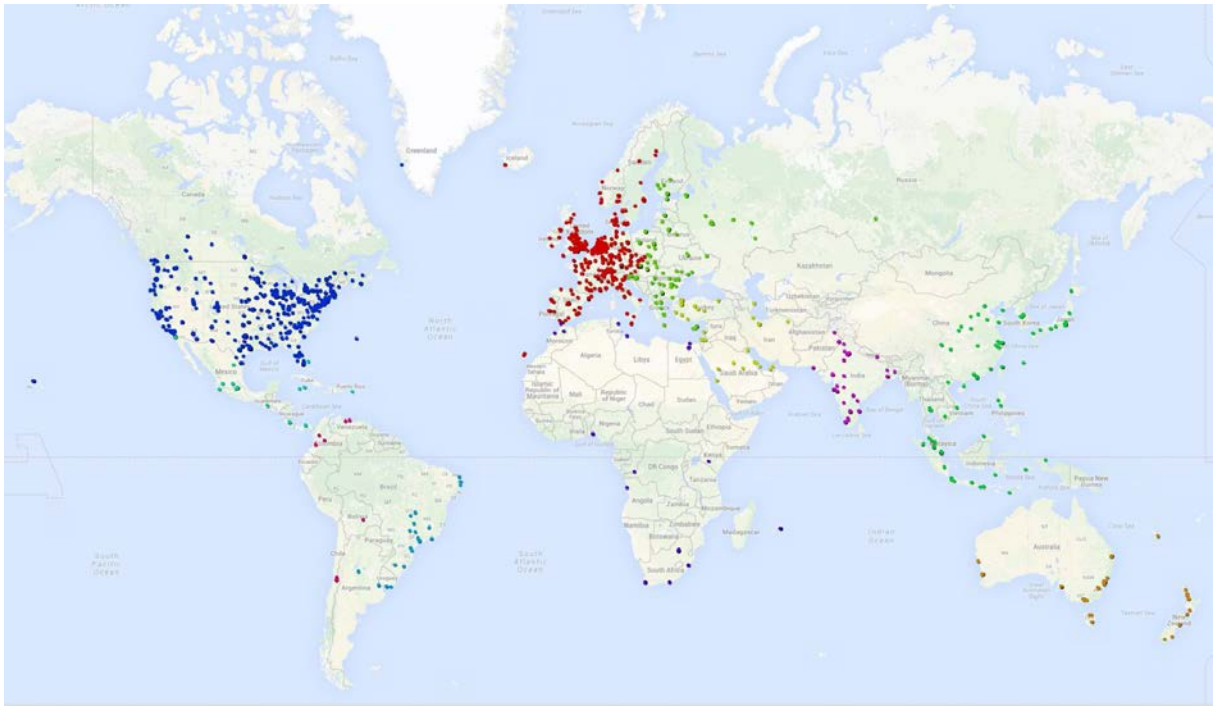


Figure 2 Geographical location of the 3116 datacentres obtained from datacentermap.com

Internet routing geography between users and data centres was modelled by three segments: user location to the capital city of the country where the user is located; capital city of the user's country to the capital city of the DC's location (this is zero if the user and DC are located in the same country); capital city of the country where the DC is located to the DC location. This models that interdomain routing is usually though public Internet Exchange Points, typically located in capital cities, or through direct peering between ASes, which is also done in large points of presence, usually in capital cities. The great circle haversine distance was calculated for each of the three segments and the three segments were summed for each user and DC pairing. Detailed network latency measurements to datacentres are discussed in the next section. Network latency, in terms of round-trip-time, can be estimated from haversine distance using a conversion factor of approximately 55km/ms, as determined by the analysis of measurements of global Internet traffic [LAC13]. This conversion factor accounts for queuing delays in intermediate switches and routers.

Our model shows that 100% of users can reach at least one DC within 50ms and just under 60% can reach a DC within 1ms. It should be noted that this model assumes RTT latency is core network latency only, measurements of service-level RTT, including access networks, internal DC networks and latency of the software stack, and modified figures based on actual measurements are shown in section 3.

The main observation from the modelling reported in this section is that a large fraction of the worldwide population has already a large number of datacentres within a relatively small RTT. With the expected deployment of datacentres at the extreme edge of the network, this number will further increase.

For tactile services, such as those envisioned by 5G networks with a response time of 1ms or less [FET14], the existing DCs may indeed not be sufficient and additional micro-DCs within ISP-provided locations may be required to keep latency below 10ms. On the other hand latency-tolerant services, such as document editing, can be deployed in a handful of centralized locations.

However, even for latency-tolerant services there may be prohibitive bandwidth costs associated with shipping large quantities of data to centralized locations for applications, such as remote video processing or large-scale data analysis, and the use of nearby DCs for bandwidth-hungry applications is an additional driver for replicating service instances in multiple locations, closer to the users and data sources.

For the majority of applications that lie between these two extremes and require a response within 80-100ms, including audio-visual applications such as video conferencing and cloud gaming a deployment in a number of the existing DCs is sufficient to meet performance requirements.

A detailed analysis of the availability of datacentres to users around the globe within a certain latency has been suppressed in the public version of this report as this content is currently under review for publication in a scientific journal.

3. NETWORK CHARACTERISATION

In the previous section, network latency between users and DCs was modelled by mapping distance to latency using the analysis of measurements of global Internet traffic [LAC13]. In this section, we conduct an in-depth analysis of the network metrics. In particular, we address the following research questions:

- How can we model the end-to-end latency between end-users and datacentres?
- What is the stability of network connection between a specific end-user and a datacentre?
- How to take into account transit costs for inter-AS traffic during the placement and resolution?

We make the study both from an over-the-top perspective (section 3.1) as well as from an ISP viewpoint. We zoom in on the possible role of the ISP in the resolution process. Both the inter-domain (section 3.2) and intra-domain aspects (section 3.3) are considered. The ISP-related results are drawn based on measurements from the operational network of Orange Poland, AS5617. More specifically, the analysis in this section has the following goals:

- to ascertain if the degree of variability in IP network routing can justify the involvement of ISP to support FUSION resolution and orchestration functions
- to provide initial data from operational networks that subsequently can be used to develop a model of the network

This section concludes with an analysis of the inter-AS traffic costs in section 3.4

3.1 OTT perspective: measurements and modelling of end-to-end latency

The main research question addressed in this section is whether using the haversine distance[4] between end-user locations and datacentres is a good approximator for the observed end-to-end latency. The rationale behind this question is that it is economically and technically infeasible for OTT resolvers to obtain measurements for every (end-user location, data centre) pair. Therefore, having a latency model would be very useful for FUSION resolvers that are operated by OTT parties which have no access to detailed monitoring information.

CloudHarmony™ (www.cloudharmony.com) monitors datacentres worldwide and provides APIs to access this data. In addition, it provides functionality to test network latency from any location. This can be done by running a Javascript 'compute-latency-speedtest.js', which sets up a connection to a server-side component running in 209 datacentres worldwide. For each of these nodes, this script runs 12 latency measurements by downloading a very small Javascript from the server (e.g. http://atlanta.vultr.cloudharmony.net/probe/ping.js?3kdocre) and measuring its latency. The script produces an average of the last 10 measurements to exclude warming-up effects, e.g. of the underlying TCP connection. We have conducted measurements in the period starting December 21, 2016 and ending February 22, 2016. These measurements were performed by embedding the client-side Javascript in a small html page that was loaded from local storage by the PhantomJS headless browser (phantomjs.org). The results presented below are thus reflecting the roundtrip response delay for an elementary service.

This methodology reflects the way an over-the-top resolver would estimate the network connection between an end-user and a given datacentre. Only end-to-end measurements and tools are used, possibly provided by specialized 3rd parties like CloudHarmony.

[4] The Haversine distance is also referred to as the great-circle distance: the shortest path between two points on the Earth's spherical surface.

Below, we first provide more details on the location of the probes and of the datacentres. In section 3.1.3, we analyse the measurement results.

3.1.1 Probes

We conducted tests by deploying the PhantomJS browser in 24 nodes that are part of three academic testbeds: InstaGENI, PlanetLab Europe and Kreonet. Those testbeds were accessed via iMinds' jFED tool (jfed.iminds.be). Geographical coordinates of the InstaGENI and Kreonet nodes was retrieved via the metadata provided by the testbed authorities themselves. For PlanetLab, we did not use the metadata provided on the website, instead we leverage on more accurate geographical locations that were kindly provided by Dan Komosny et al. [KPP15].

Continental distribution of the probes:

Europe	12	Asia	2
North America	9	Oceania	0
South America	0		

The geographical location of the nodes is visualized in Figure 3.

Figure 3 Geographical location of the probes, which were part of the PlanetLab Europe, InstaGENI and Kreonet nodes.

3.1.2 Datacentre locations

We were able to measure the distance from each probe to 209 datacentre locations, distributed over the different continents as follows:

Europe	57	Asia	26
North America	106	Oceania	16
South America	4		

Figure 4 provides a visual representation of the location of the data centers, according to the latitude and longitude provided by CloudHarmony™

Figure 4 Location of the CloudHarmony™ cloud sites that were queried

3.1.3 Worldwide latency measurements

In Figure 5, we shown the (distance, average latency) pairs for the results collected from a probe installed in the Virtual Wall facilities of iMinds, Belgium to each of the 209 datacentres. We have differentiated by the continent of the measured data centre.

We observe the following:

- Latency to data centres on the same continent is much lower than the latency to inter-continental data centres
- We seem to observe two linear trend lines to the North-American data centres. We conjecture this is due to two different intercontinental links
- All data centres in South-America lie in the same city (San Jose, Brazil). The latency to each of these individual data centres varies with a factor of more than two.
- Latency to the Asian data centres is subject to a very large spread.
- Oceanian data centres show the highest latency.

Mutatis mutandis, similar observations are made for probes in other continents.

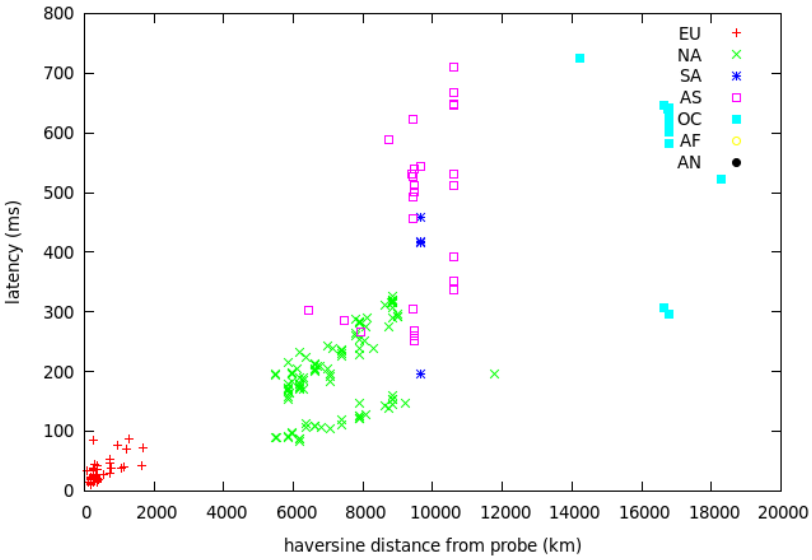
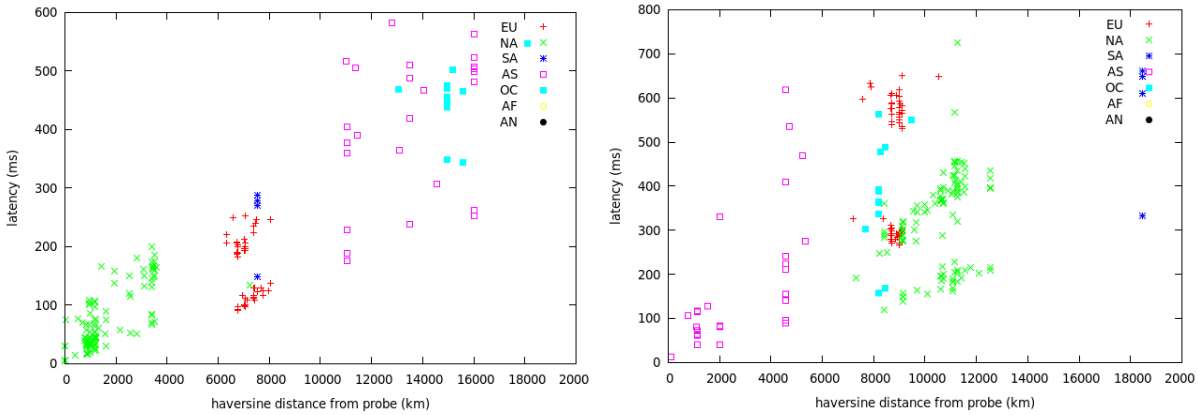


Figure 5 Scatterplot of (haversine, latency) measurements from the Virtual Wall infrastructure to 209 datacentres worldwide.

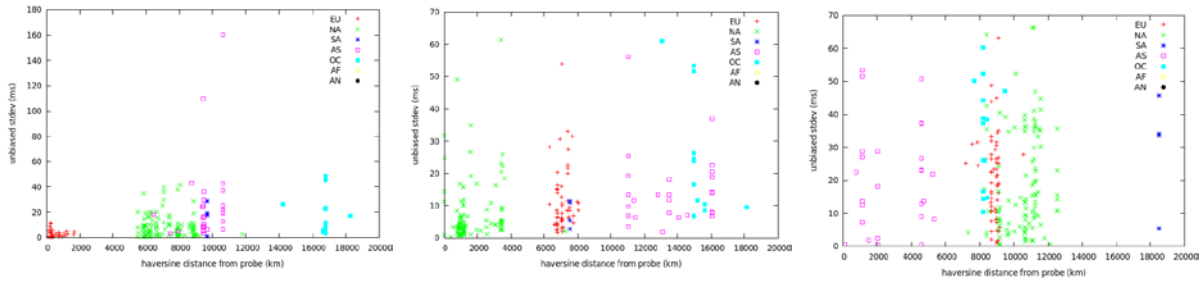
In Figure 6, we show similar graphs for the measurements from a North-American probe, located in the INSTAGENI testbed facilities at Georgia Tech University, as well as for an Asian probe, located in the KREONET facilities at Daejeon. Mutatis mutandis, we can draw the same conclusions. Interestingly, the huge variation in latency towards Asian data centres is also observed for the Asian probe, when there is only intra-continental traffic. In addition, from the Asian probe, we see a clustering of both European and North-American data centres.



a) probe at Georgia Tech University, USA b) probe at Daejeon, South-Korea

Figure 6 Scatterplot of (haversine, latency) measurements from a probe in North-America and Asia

Besides the average latency, also the jitter is an important metric to estimate the connection quality. Figure 7 contains scatterplots of the standard deviation on the latency measurements for the same probes located in Ghent, Georgia Tech and Daejon. The main conclusion is that the standard deviation is independent from the haversine distance.



a) Virtual Wall b) Georgia Tech Univ c) Daejon (Kreonet)

Figure 7 Scatterplots on standard deviation of average latency

3.1.4 Conclusion

From the analysis in this section, we can draw the following conclusions:

- There is already a lot of DC infrastructure that is accessible by clients in Europe with a (roundtrip) latency of less than 100 ms. Even without deploying new node, we thus argue that there is already today sufficient hosting infrastructure to motivate the need for a FUSION resolution.
- In Europe, an OTT resolver can apply a linear regression model that is based on the Haversine distance of a client and a DC to estimate the roundtrip latency.

From these conclusions, one might argue that there is no need to have access to detailed monitoring information to act as a resolver. Therefore, in the next section, we study the ISP perspective on the routing information to assess the added value of including this information in the resolver.

A more detailed set of measurements of probe to data-centre performance and its analysis has been suppressed in the public version of this report as this content is currently under review for publication in a scientific journal.

3.2 ISP perspective on inter-AS routing updates

The main goal of this section is to assess the added value of using inter-domain routing information by the FUSION resolver. This section thus studies the case where an ISP has deployed his own FUSION resolver, that however also resolves requests from its users to services deployed in execution zones in other AS-domains. To this end, we have made three different types of inter-domain experiments which are summarised in subsections 3.2.1, 3.2.2 and 3.2.3 respectively.

3.2.1 Dynamics of inter-domain paths

This experiment investigates the degree to which inter-domain paths change from the perspective of a given autonomous system. If path length changes were significant that could indicate also non negligible changes of inter-domain delay, and could be a good base for triggering optimisation algorithms to correct service resolution tables used by the resolver. During the study, we investigate a number of long-term statistics related to the length of inter-domain paths which we define along the discussion (in the Annex). The source of information is the stream of BGP updates as seen by all border routers of the AS and we apply the analysis to this stream. As we use BGP information, path length is expressed in terms of transit domains.

Based on BGP updates from other ASes, ISPs have detailed information on the path taken between their domain towards execution zones. These paths may change over time, because of link failures or updated traffic policies. In this section, we analyse the variability of the length of inter-domain paths (paths between non-neighbouring ASes). It is motivated by the conjecture that such inter-domain route changes can result in path length variations sufficiently large that they cause significant changes of inter-domain delay that may affect service quality. Confirming this conjecture could provide a

strong argument in favour of using the information from inter-domain routing to improve the FUSION resolution and thus the quality of service offered in a given ISP domain.

We note that although a lot of related work has been published by many authors, we've decided to run our own study to rely on up-to-date data available from own AS, for which some data mining could be done using tools from operational teams, and given that we were interested in specific parameters not studied elsewhere. A detailed discussion of the results of the study is contained in ANNEX 1, section 9.1. The main conclusions from this analysis are summarised in the following.

- A general trend known from earlier studies keeps valid: larger networks (with shorter IP address masks) experience much lower BGP update frequencies, i.e., have much more stable routing paths than smaller networks (with longer IP address masks). This means that centralized monitoring of inter-domain events related to services located in small subnetworks can easily face scalability problems.
- The majority of updates (up to 90% of all BGP notifications) relate to subnetworks with 24-bit mask. In fact, subnetworks with masks of 10 to 16 bits long account for about 10% of the visible updates, while the largest subnetworks are responsible for not more than 1% of all updates received by the AS. In terms of the intensity, we note that 1% of updates corresponds to around 30 updates per second. Considering that these figures account for the whole globe and ISP-owned FUSION resolvers could probably be tuned to services located rather in the vicinity of the domain (e.g., same continent), one can expect that the numbers FUSION resolvers would have to tackle in practice could be much lower than those provided above because information from ASes in other continents can be disregarded.
- The statistical analysis of BGP updates (see sec. 9.1.4.3) suggests that **a reasonable minimum reaction time of service resolution to *enable* service instances for resolution** based on BGP information is in the order of 120-240 seconds (we note that according to [RIPE], a similar time is also needed for routes to converge during a typical event). This is because assuring that a route is consistently active need some guard time. **Much shorter reaction times to *disable* existing routes (and disable related service instances from resolution) can be achieved**. This is because the detection of even the first BGP notification is sufficient to trigger a disabling action for related routes and corresponding service instances. It has to be noted that filtering out routing events assuming time scales in the range mentioned above (and even much larger) probably does not eliminate any interesting destinations as potential points where service instances can be run (see sec. 9.1.4.3).
- From the analysis of the average and the standard deviation of the path length changes due to BGP updates, we conclude that **the knowledge of the length of inter-domain paths alone is not sufficient for service resolution based on inter-domain delay**. So it becomes clear that the knowledge of inter-domain paths should be linked to direct measurements of delay if service resolution has to be based on the delay criterion. Moreover, for the latter to make sense, there must be sufficient variability of end-to-end delays (confirmed by direct measurements).
- Additionally, inter-domain delay changes should visibly coincide with inter-domain routing events. The degree to which the two latter requirements are satisfied is investigated in subsections 3.2.2 and 3.2.3.

3.2.2 Inter-domain delay evolution

We remind that in section 3.1 only very-long term averages of inter-domain delays were taken into account, while the analysis in 3.2.1 was not conclusive in terms of delay variability, because inter-domain paths (expressed as the number of transit domains) in general are quite stable along time. The goal of the experiment summarised in this section is to evaluate the **actual level of variability of inter-domain delays**. The main focus now is on **fluctuations of delays in relatively long time periods**, i.e.,

disregarding the structure of very-short time scale changes/instabilities. We note the latter is addressed in section 3.2.3.

The experiment consisted in measuring the delays from a given location in Orange ISP network and one location in other ISP network to several DCs worldwide (in other ASes).The set of DCs was the same as the one used in section 3.1. The difference wrt section 3.1. (apart from using only two probes) is that we increased the frequency of probing so that each DC was sampled every 6 minutes to achieve a better resolution of measurements. This increased resolution was also necessary to explore more closely short-term impact of link failures in the network (seen as BGP update storms - caused by "large events"[5]) which is studied more closely in section 3.2.3.

A detailed discussion of the results of measurements is given in ANNEX 1, section 9.2. The main conclusions from this analysis can be summarised as follows.

- Most of the DCs we have analysed offer sustainable RTT over quite long periods of time, but for many of them there are periods when the RTT increases or is highly variable at different time scales (sometimes on the order of hours, sometimes several minutes).
- Typically, even if some DCs experience increased RTT in a given point in time, there are sufficient alternatives provided that they can be efficiently used by the application.
- Network delay estimates based on geographical coordinates can be very imprecise in many cases.
- In many cases it is hard to precisely establish root causes of the behaviour of RTT as the changes cannot always be attributed to routing. Therefore one should accept that this kind of variability has to be measured using methods based on active probing or client reporting.
- Evaluation of effective delays between clients and service instances based only on geographical coordinates can produce significant errors. Topological information regarding at least hosting networks should additionally be taken into account for critical services.
- Potential added value of ISP in this case is the ability to support probe measurements by offering a sufficient set of representative measurement points in its domain. Adopting a more aggressive approach, the ISP can also carry out such measurements on its own and even offer service resolution. In fact, the ability of ISP to provide the smallest possible set of representative (good) measurement points can be an advantage once the ISP gets involved in the provision of FUSION service resolution.

Summarising, the results obtained from the experiments confirm the well-known fact that application-level round-trip delays can differ significantly depending on the source-destination pair. While an RTT of only a few ms can be observed for source and destination in the same city and AS, an RTT of over 700 ms can be expected between Europe and certain far-Eastern locations. While in the majority of cases the observed delays were quite stable over long periods of time, in certain cases RTT experiences significant diurnal variations that, however, are quite predictable (most probably they result from policies deployed in the network to handle inter-domain traffic). These observations suggest that OTT measurement methods that have been exploited extensively by many OTT service providers in the inter-domain scenario will also be sufficient for the orchestration and resolution of many FUSION services.

Less explored area relates to the **transient behaviour of inter-domain delay** during large events relatively easily identifiable in BGP. ISPs, having access to timely BGP routing information, can potentially provide an added value to cope with such transient conditions. This can be valid under the

[5] Each BGP update is caused by some event taking place in network equipment (like a failure or configuration change, etc.). We refer to an event as "large" when it is a root cause for a lot of updates (and possibly for a lot of subsequent route changes followed by further updates, etc.). So informally, a large event can be related to a BGP update storm.

condition that there will be services demanding very fast reaction to events. Such reaction could be as simple as seizing the resolution to instances topologically covered by the event (this decision can be taken before RTT measurements can indicate any problems). In the next section, we take a closer look at the qualitative aspect of the transient behaviour of inter-domain delay.

3.2.3 Observable BGP events and inter-domain delay changes

In the previous section we gave a general picture of delay dynamics, showing a very moderate degree of variability of inter-domain delay in the case of many destinations. This has led to the conclusion that the use of classical methods for estimating inter-domain delays can be sufficient for a lot of FUSION services. By contrast, in this section we provide a more detailed picture of short-time scale variations of delay during identified failures of network links (referred to as "large events"). In our case we were able to identify root causes of BGP storms using operational tools or simply hints from the network management team. We motivate this experiment as explained below.

Demanding or critical services may require orchestration and/or resolution based on up-to-date knowledge of the network state. Therefore the last experiment for inter-domain routing was to assess **the impact of routing changes on the e2e delay between users and datacenters**. To achieve this we aligned in time BGP routing events with the histogram of e2e delays for a set of data centres. While in the previous section we analysed the long-term (stable) behavior, in this section we focus on the very short-term changes.

Our main observations regarding each BGP routing event are given in section 9.3. The main conclusions are given in the following list.

1. For any "major" event[6] observed **the number of destinations explicitly affected did not exceed 10% of all monitored destinations**. In reality, the number of affected destinations could have been higher, but they remained hidden from us due to the limited resolution of the probe-based measurement method.
2. For most of the observable cases, the impact takes the form of a loss/increase of connectivity (sometimes a drop of connectivity). Such a **change of RTT is transient**, takes place when routing converges within the network, **its duration typically fits into the 6 minutes interval,** and only on rare occasions degradation time extends to longer intervals of, say, 20 minutes.
3. When the routing converges, **RTT typically stabilises on the level from before the event** (even the AS path changes). Noticeably, during event-free periods, RTT is often pretty stable. These facts seem to confirm the conjecture that majority of alternative paths for inter-domain destinations are similar in length to the primary ones.
4. **Up to 10% of visibly affected destinations (datacenters) experience a persistent change of RTT** in consequence of routing updates when a statistical *major* event occurs. This translates into less than 1% of all monitored destinations being affected by a persistent change of RTT as a result of a *major* event.
5. Not more than 0.5% of all monitored destinations on average experience a major persistent changes or RTT (a change above 30% of the initial value of RTT) during a *major* event.
6. In reality, less profound BGP events also contribute to RTT variations (we provide examples of such changes in section 9.2.4) and the percentages of affected destinations can be higher than indicated above. However, the number of such events grows rapidly. **Therefore tracking them in real time using BGP information and making efficient use of this knowledge in the service resolution process can easily lead to scalability problems.**

[6] Roughly speaking, a major event generates an significant number of notifications in a short time interval.

3.3 AS-internal delay

To complement our knowledge on E2E delays in the network we have also analysed the delays internal in the Orange AS. The measurements relate thus relate to this specific domain, but we believe the results can be representative for many operational networks. The measurement scenario and related results are discussed in detail in section 9.4. In the following, the main conclusions drawn from the study are listed.

- **Latencies are rather low/moderate and relatively stable.** We recall the reader that we are discussing the intra-domain component of the E2E RTT delay, and note also that the destination locations that were used correspond to possible locations of future mini-data centres of an ISP: in future generation Points-of-Presence of ISP designed to host network functions and user services according to new service provisioning paradigms. Note also that there is no wireless component involved.
- In most cases, we observe longer periods of where the delay is at a stable lower bound. The most obvious explanation is that this lower bound indicates pure transmission and switching delay in network devices, i.e., without queueing in routers. The load on the links along the paths is rather low and (probabilistically) the majority of the packets do not experience queuing delay. Occasionally, we observed stable jumps by several ms of the lower-bound of the delay. These jumps of the lower bound lasted for quite long periods of time (e.g., hours). We conjecture that such changes can result from changes in network routing. Unfortunately, we have not been able to validate this conjecture because intra-domain routing information was not available.
- The source of higher-order variations in the delay is not known exactly - occasionally it will be due to link load jitter (queueing effect), but alternatively it may result from varying server load (both explanations can apply to RTT and file transfer measurements). We would have to know server load histograms to eliminate/confirm the server-load component. We point the reader to the case of a Google DC in the measurement set, where the jitter is highest among all the servers which suggests that server-load may be the dominant component of the higher-level variation of latencies).
- The differences of the pure forwarding delay from different locations towards a given point in the same autonomous system can vary (so far, a confirmed value of the ratio is three). Therefore, placing service instances in the vicinity of users and appropriate service instance resolution can be important for services demanding in terms of latencies.

The measurements we have used are quite sparse in time due to the limitations imposed by current operational procedures. We nevertheless believe that the results already obtained can be used to justify designing the resolution and orchestration for demanding services that operate on the level of particular autonomous systems thus promoting the ISP to become a part of the value chain in FUSION architecture.

3.4 Inter-AS bandwidth considerations

Routing service requests to remote ASes has both performance and cost implications. In FUSION the utility function for a specific service determines the constraints on the performance that a service request can receive. However several service instances may exist within the maximum network latency determined by the utility function. Instances may be located in EZs within the local AS as well as in neighbouring and remote AS domains. Resolving requests for high bandwidth services to instances located in EZs in remote domains causes increased traffic on inter-domain links, which, depending on the business relationship between ASes can significantly increase the costs to the ISPs servicing those requests. Inter-domain links can be generally classified as customer-provider, when a customer AS pays for transit connectivity from a provider AS, or peer-to-peer where two ASes exchange traffic mutually free of charge [GOA]. To reduce inter-domain costs, ISPs prefer to keep traffic within their

own domain or to route requests to ASes where they have a peering relationship. This reduction in inter-domain traffic and costs is one of the major factors in the deployment of local CDN nodes by ISPs for the delivery of popular static content. The replication of service instances in FUSION is primarily for ensuring that service instances are available within the performance constraints for that service but selection between instances by the resolution system can have a significant impact on the inter-domain bandwidth, and therefore the costs, over inter-domain links.

A conclusion from this analysis is that when considering services in locations that meet the performance constraints of the utility function of a service, a large number of locations will not be within a user's AS or in neighbouring ASes. This means that minimising transit costs is not simply a matter of selecting local or neighbouring EZs. Most candidate EZs are available in remote ASes over paid-for transit links and so minimising transit costs in the resolution process means taking into account the costs of individual transit links. Reducing transit costs is mostly in the interest of the ISP and, furthermore, detailed information on transit costs is unlikely to be shared with third parties, for business confidentiality reasons. Hence, resolution decisions that aim to reduce transit costs while maintaining performance targets according to the utility function of a service are best made by ISPs themselves, or by trusted third parties that have access to full cost information.

A detailed analysis of the availability of datacentres over peering and transit links has been suppressed in the public version of this report as this content is currently under review for publication in a scientific journal.

3.5 Role of ISP in EZ selection

Many of our observations from this section confirm the findings known from earlier works, e.g., [ELKD2012], [ACBD2004], [RIPEPRO], [iPlane]. The main messages relevant for FUSION that results from the studies presented in this section are presented in the following.

1. Using the OTT approach, timely reaction to short-term deteriorations of connectivity is possible only when e2e measurements are taken very frequently. This however can be difficult to achieve in realistic scenarios. Therefore, **monitoring BGP events in real time can be seen as an enabler that allows to increase the responsiveness of service resolution (and possibly service orchestration) platform.**
2. BGP route changes are observed for a large portion of the used IP address space and over a wide range of time scales. Consequently, keeping track of respective changes by smaller service providers can be prohibitive for them. Monitoring BGP events on large scale can effectively be done by ISPs, especially in the vicinity of their AS. This in turn may open opportunities for ISPs to reclaim position in the value chain by providing resolution service and supporting orchestration for demanding services.
3. As is shown by the authors in [VERMO2014], such a position of ISPs can be even strengthened in the future when IPs get involved in the edge cloud ecosystem under the growing popularity of demanding services distributed in the edge.
4. The knowledge of the length of inter-domain paths alone is not sufficient for service resolution if inter-domain delay is the criterion. For effective resolution with delay as a criterion, BGP information should be combined with direct measurements of delay. Observed variability of end-to-end- delays (confirmed by direct measurements) suggests that ISP-supported resolution has the most commercial potential for demanding services. Inter-domain delay changes often coincide with inter-domain routing events and in such cases BGP information can efficiently be used for service resolution.
5. A market value of final conclusion arising from this analysis is that ISP who really plans to offer service resolution for FUSION services should combine e2e measurements with monitoring information from inter-domain routing.

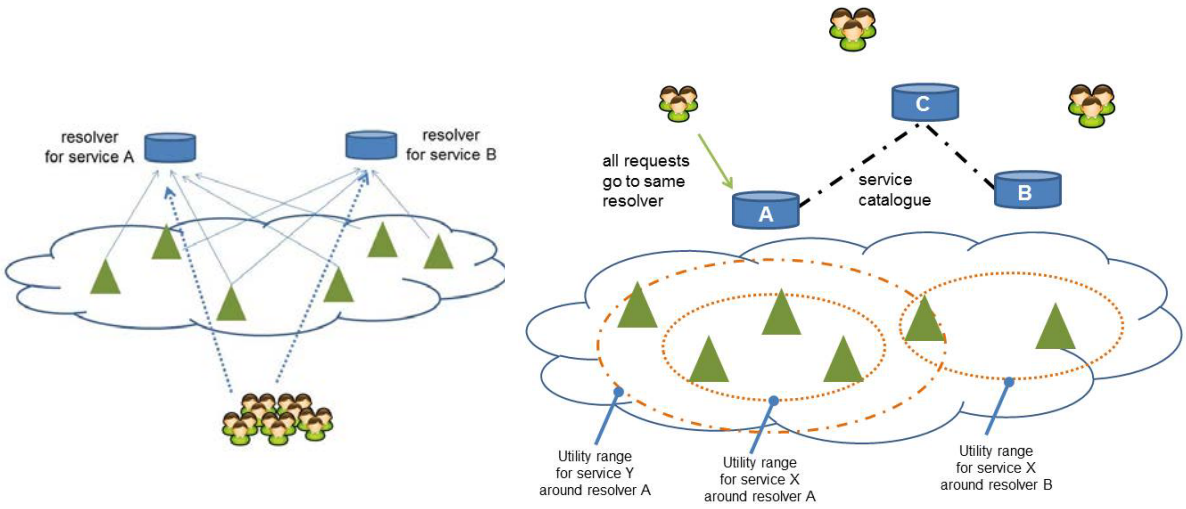
4. SCALABILITY ANALYSIS

In the operational model of FUSION, clients will send a resolution request for a given service ID to a resolver. Conceptually, this means that the resolution is decoupled per service since there is no overlap per service. An important question, and the main focus of this section, is the optimal degree of distribution in service resolution. We compare a per-service, logically centralized resolution model with the distributed utility-based pub-sub architecture introduced in section 1.2.

In a centralized solution, we have one resolver per service that handles requests from users worldwide. However, this means that each data centre worldwide hosting an instance of that service must report to that single resolver. Moreover, it is unlikely that a centralized resolver has a detailed and up-to-date view on network metrics between any user location and any data centre. In a distributed solution, multiple resolvers may handle requests for a given service. This may come at the price of synchronization overhead between resolvers.

4.1 Models for service resolver placement

We will study the two models shown in the figure below. Note that the right hand side of the figure is the FUSION resolution architecture already described in section 1.2.



(a) Centralized model: one resolver per service

(b) Utility-based publish-subscribe

Figure 8 Models for resolver placement studied in this section.

Centralized model (CM)

This model is shown in the left part of Figure 8. Each service has its own (logical) resolver, and *any user worldwide* requesting service A will be directed to an instance by the same service resolver. The IP address of the resolver can be retrieved through unmodified DNS. ISPs or other entities like CDN operators can deploy and operate replicas of service resolvers for the purpose of minimizing latency and network costs for high-volume services, but these proxies would still be conceptually a centralized per-service entity. The internals of the resolution process can remain proprietary to the service provider.

Due to business reasons, availability of detailed network monitoring information is unlikely in this model, since resolvers will be operated by a different business entity than the network operator (ISP). Monitoring information on the network between end-user locations and execution zones can be

retrieved using data from over-the-top measurement providers like Cedexis or CloudHarmony. Nevertheless, it is possible to have access to detailed network monitoring information from the ISP, e.g. using a similar approach as formulated in [FPS13], where the network provider returns to the resolver an ordered list of candidate execution zones in his network, without exposing his internal business logic.

In this model, signalling information flows in the following directions:

- all user requests for a given service are directed to the same (logical) instance, and resolution answers are flowing in the opposite direction
- each execution zone hosting an instance of a service reports on service availability to the dedicated resolver of that service. This means that execution zones must maintain separate connections with all resolvers of services hosted in their execution zone.
- there is no exchange of information between resolvers
- network monitoring information is consumed separately by each resolver

Utility-based publish-subscribe (U-PubSub)

This model was already explained in section 1.2. In this model, signalling information flows in the following directions:

- service requests from the user to his domain resolver, which possibly forwards the request to another resolver if the other resolver does not expose the actual locators of instances in specific execution zones.
- instance locators of resolved requests from resolvers back to users
- instance availability updates from execution zones to subscribed resolvers. Information of all services that a given execution zone subscripted to can be aggregated. Execution zones can send updates either periodically or event-based, e.g. when the number of available updates drops below a predefined threshold, when there is a step change of at least N session slots compared to the previous update, etc. In case a resolver wants to hide the locators of some execution zones, such messages can also be generated by resolvers.
- service advertisements between service resolvers.
- network monitoring information is consumed by the domain resolver

Comparison between both models

The table below summarizes the similarities and differences between both models.

	Centralized, per-service resolution	**Utility-based pub-sub**
Finding IP address of appropriate resolver	unmodified DNS (akin to CDNs)	unmodified DNS (akin to CDNs)
Resolution logic	based on utility, network metrics and session slot information + possible service-specific information (e.g. monitoring probes) from deployed instances	Based on utility, network metrics and session slot information

Information exchange between resolvers	none	catalogue + forwarded requests
Availability of network information	From OTT providers and/or exposed by ISPs (e.g. via ALTO)	From OTT providers and/or exposed by ISPs (e.g. via ALTO) + internal details if resolver is operated by ISP
Session slot information	Separate flows from execution zone to all resolvers of the deployed services, no aggregation possible	To all subscribed resolvers, aggregation per resolver possible

4.2 Model for placement and capacity allocation of service instances

As the focus of our study is on the resolution, we start from a given deployment. The deployment model states the location of service instances, and the capacity allocated to each of those instances. Both deployment and capacity allocation should be adjusted to the expected demand.

We have used empirical evidence from literature and publicly available datasets to model the demand as well as the geographical location of execution zones. These models are described in this section.

4.2.1 Geo-spatial distribution of demand

The demand is modelled after the demographic distribution of the worldwide population. We started from the 'cities1000' dataset provided by GeoNames.org. This file contains all cities worldwide with a population of more than 1000 inhabitants. Nine countries are not present in this dataset, because we could not find reliable statistics on the internet penetration. The concerned countries are very small, involving e.g. Pitcairn or the French Southern Territories.

The dataset contains 112 106 cities, with 3.03812×10^9 users. According to the most recent statistics on the world population, this is an underestimation with a factor of 2.34. This has two main reasons. First, the original dataset does not contain any city worldwide with less than 1000 inhabitants. Second, and according to us the most important reason, is the age of the input data. For example, the city of Ghent contains 231 000 users in the dataset, whereas the most recent figures indicate a population of 250 000 users.

At this stage, **we only use the 43 173 European cities in the dataset**. Each city generates a total demand (across all services considered, see below) following a **Poisson process** with a parameter lambda that is **proportional to the city population**: lambda is set to 1/100 000 of the city population. Note that we could easily experiment with other scaling factors. We believe this would not affect the overall conclusions, as the demand ratio between city populations is maintained.

Figure 9 visualizes the resulting demand. Lambda values range between 0.02 and 75.57. The cumulated demand to be handled is a Poisson process with lambda 4 811.83.

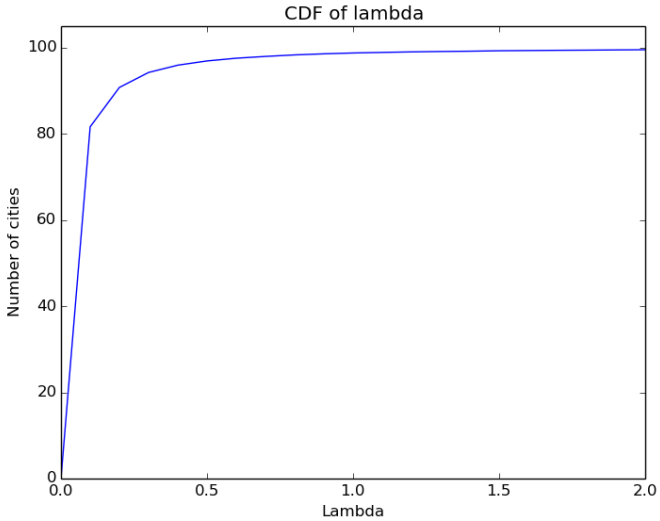


Figure 9 CDF of the fraction of cities generating service requests following a Poisson distribution with parameter lambda less than or equal a given value.

4.2.2 Locations of execution zones

Highly distributed infrastructure

In this scenario, we are considering all 1005 datacentres located on the European continent in the set found on [DCMAP]. These locations are shown in Figure 10.

This model is suitable for services with extremely low-latency, high-bandwidth requirements that need to be placed in multiple locations close to the user. Examples are tactile Internet services with required response times below 10 ms.

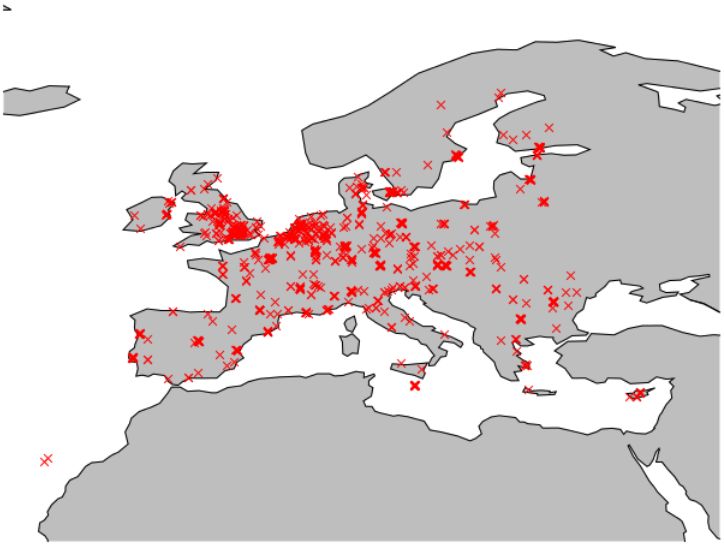


Figure 10 Locations of 1005 European datacentres

Medium distributed infrastructure

In this scenario, we assume that instances are deployed on 50 different locations throughout Europe. We applied k-means clustering [KMEAN] on the 1 005 geographical locations of the highly distributed

scenario to obtain 50 cluster centroids. For each of the centroids, we then select the DC in our data set that is geographically closest to its location. The final selection is shown in Figure 11.

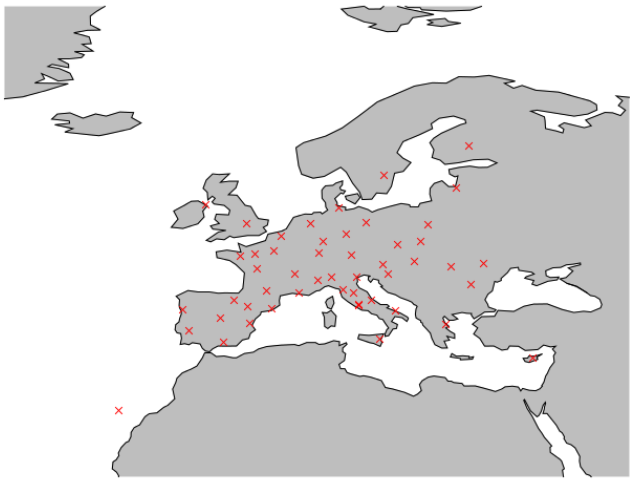


Figure 11 Locations of 50 DCs after applying k-means clustering

This reduction in execution zones was introduced for various reasons:

- Orchestrators may want to limit the number of execution zones under control
- For services requiring a response within 80-100 ms, including audio-visual applications such as video conferencing and cloud gaming a deployment in a number of the existing DCs is sufficient to meet performance requirements

Min/max Latency between cities and datacenter locations

Latency is determined based on the geographical distance between a city and the DC, according to the model that was developed in section 3 of this deliverable.

The best connected cities see all 5 DCs in the same city (0 ms). When we consider all 1005 DCs, the worst connected cities see DCs between 5 ms and 39 ms. When we consider only the subset of 50 DCs, the worst connected cities see DCs at 7 ms for the closest DC and up to 39 ms to the 5th closest.

4.3 Reporting overhead

In this section, we study the reporting overhead of the CM and U-PubSub resolver deployment scenarios described in section 4.1.

To estimate the number of services being used, we consulted various sources:

- Number of apps in Google Play Store: 1.5 x 10^6 (www.appbrain.com/stats/number-of-android-apps)
- Number of websites: 876 x 10^6 (tekeye.biz/2014/how-many-websites-are-there)
- Number of registered domain names: 265 x 10^6 (tekeye.biz/2014/how-many-websites-are-there)
- Pages indexed by Google: 4.5 x 10^9 (worldwidewebsize.com)

Based on these numbers, and accounting for the fact that only services needing a medium to high degree of distribution can benefit from the FUSION architecture, **we will estimate the number of services between 2 x 10^6 and 1 x 10^9.**

Obviously, the requirements will differ across each service. We will model the maximum tolerable utility range of a service as a **truncated log-normal distribution** .

4.3.1 Centralized model

In the centralized model, the signalling traffic due to **reporting** is independent per service, as we have (logically) separate resolvers per service. For a given service, the traffic flowing into the service-specific resolver can be calculated as follows:

$$N_{EZ} \cdot \frac{S_{report}}{T}$$

where N_{EZ} denotes the number of execution zones where an instance of the service is deployed, S_{report} denotes the size of an individual report and T is the reporting frequency. Note that this formula assumes that each execution zone follows the same periodic reporting policy.

We estimate the size of a report, S_{report} from an execution zone to a resolver at **20 bytes**. This is the sum of 16 bytes for a zone identifier (e.g. a 128-bit UUID), and 4 bytes for the number of session slots (32-bit integer). Note that in the centralized model, it is not necessary to include a service identifier in the report: this is implicit by the resolver receiving the report.

The parameter N_{EZ} depends on the acceptable utility range for the service being considered as well as on the placement policy. Services needing low latency will need to be deployed in many execution zones and the session slots needed to serve the demand from a given location can be spread over multiple DCs in the utility range to improve resilience to the failure of a single DC.

In a first placement policy, we focus on a resilience and assume that service instances are deployed in all 50 DC locations of Figure 11. The calculation of the reporting traffic for a single service is straightforward. The reporting traffic from a given EZ to one service resolver amounts up to 1.6 kbps for T = 0.5s, dropping to only 0.013 kbps for T = 60s. The total number of reporting traffic is then easily calculated by multiplying these numbers with a factor of 50 and the total number of services.

The results are captured in the following table:

no. services	**T = 0.5s**	**T = 1s**	**T = 30s**	**T = 60s**
2 x 10^6	3.2 Gbps	1.6 Gbps	0.053 Gbps	0.026 Gbps
1 x 10^9	1 600 Gbps	800 Gbps	26.67 Gbps	13.33 Gbps

Obviously, this is only a very rough approximation. Given the large number of DCs available in Europe, even within low latency limits, most services will only be deployed in one or two DCs in Europe. We therefore refine the calculations for a more advanced instance placement that take into account utility requirements.

For a given service with a maximum tolerable utility range, we calculate the **minimum number** of DC locations that need to host that service, so that at least one DC is accessible within that service utility range from all 43 173 European cities in our dataset. Here, we define utility as latency. We considered K=50 and K=1005 DC locations, as explained in section 4.2.2.

Table 1 Number of DCs needed so that each city in Europe has at least one DC in the given latency (utility constraint)

K=50		**K=1005**	
latency	*no. of DCs*	*latency*	*no. of DCs*
[0-27]	not feasible	[0-27]	Not feasible

28	6	[28-44]	4
[29-30]	5	45	3
[31-41]	4	[46-58]	2
[42-44]	3	[59 …]	1
[45-60]	2		
[61 …	1		

Figure 12 shows the total traffic flowing into the central service-specific resolver for a service with a maximum tolerable latency.

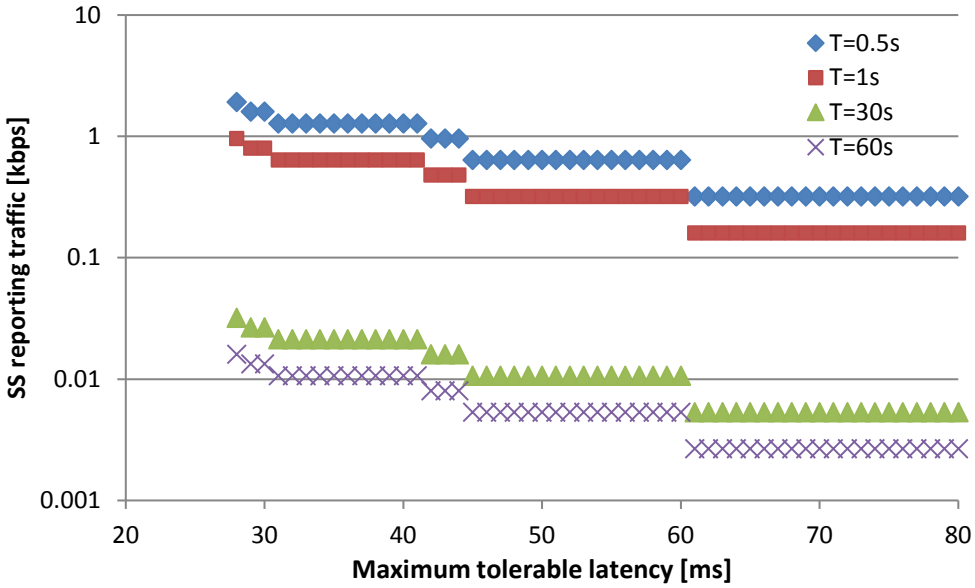


Figure 12 Reporting traffic from all replicas of a service with a given maximum tolerable latency to a single resolver, and parameterized with the reporting interval T

To calculate the total number of signalling traffic flowing from datacenters to service resolvers, we model the distribution of service utility range following a **truncated log-normal distribution (μ = 5, σ = 1) in the range [28,100 ms].** The lower bound of this range is determined by the minimum latency between a city and its nearest DC in our dataset. The result is reported in the table below.

Table 2 Reporting traffic between execution zones and per-service centralized resolvers

no. services	T = 0.5s	T = 1s	T = 30s	T = 60s
2×10^6	1.2 Gbps	0.62 Gbps	0.021 Gbps	0.010 Gbps
1×10^9	620.4 Gbps	310.2 Gbps	10.3 Gbps	5.17 Gbps

With the given assumptions, the total amount of reporting traffic is proportional to the number of services and the reporting interval. We considered the worst case, where each service has its own resolver, so execution zones cannot aggregate reports of multiple services to the same resolver. On

the other hand, we did not account for a possible optimization where the reporting period T is adjusted to the rate at which the session slots change in a given execution zone. For example, if the number of available session slots in an execution zone changes not much over time, this execution zone could decide not to send a reporting update to the resolver of that service.

4.3.2 Utility-based pub-sub model

In this model, resolvers subscribe to a set of DCs to obtain session slot updates of a set of the hosted services (not all services hosted in that execution zone might be of interest to the resolver).

Such a session slot report is a list of tuples (service ID, number of session slots). We estimate the size of such a tuple to *E* = **20 bytes**: 16 bytes for the service ID (e.g. a 128-bit UUID), and 4 bytes for the number of session slots (32-bit unsigned integer).

In this model, the total reporting traffic can be obtained by summing over all resolvers and over all execution zones:

$$\sum_{r} \sum_{DC \in \Omega(r)} \frac{Sub(r, DC) \cdot E}{T_{DC}}$$

The following parameters are used in this formula:

- *r*: index over all resolvers
- *DC*: index over all DCs
- $\Omega(r)$: set of all EZs where resolver *r* has subscribed to for session slot updates
- *Sub(r, DC)*: the number of services running in execution zone *DC* for which resolver *r* wants to receive session slot updates.
- *E*: size of a tuple in the session slot report (estimated to 20 bytes)
- T_{DC}: the reporting period used by that execution zone. In this section, we will assume a fixed and identical reporting period for all DCs

Not all services will be deployed in a given DC. We will use the same minimum-cost service placement used in the previous section. This means that we have selected from K=50 or K=1050 possible locations the minimum set needed so that each city in Europe has at least one instance within that utility range. We also keep identical the total number of services considered: 2 x 10^6 and 1 x 10^9. We also keep the statistical distribution of utility constraints: the maximum utility of a service follows a log-normal distribution.

We vary the number of resolvers between 2, 5, 10 or 28 (the latter number reflects the EU-28). To determine the resolver locations, we run a K-means clustering algorithm on the set of K=50 or K=1050 data centre locations, and map the result to the nearest actual DC. We thus assume that resolvers are also located in an existing data centre. The users from a given city all direct their request to the closest resolver (in terms of latency).

Each resolver will then have to subscribe to as much DCs as needed so that each city in its resolution domain can reach at least 1 DC within that utility range. Table 3 indicates, for different numbers of resolvers, the total amount of reporting connections between resolvers and zones, given the minimum-cost placement introduced earlier.

The global trend is as expected: the number of subscriptions needed decreases for services with less strict latency constraints. For a larger number of resolvers, there are also more subscriptions, but the trend is less than linear. For example, for a service with max latency 28, there are 5 (resolver,zone) connections for the 2-resolver scenario, whereas there are 36 connections for the 28-resolver

scenario. Although the number of resolvers has increased with a factor of 14, the number of subscriptions has only increased with a factor 7.

Table 3 Total number of connections between zones and resolvers for services with maximum utility. Resolver and zone locations are chosen from the K=1050 dataset.

2 resolvers		**5 resolvers**		**10 resolvers**		**28 resolvers**	
latency	*no. of conn.*	*latency*	*no. of conn.*	*latency*	*no. of conn.*	*latency*	*no. of conn.*
[0-27]	not feasible	[0-27]	not feasible	[0-27]	not feasible	[0-27]	not feasible
[28-40]	5	[28-44]	10	[28-34]	15	[28-41]	36
[41-43]	4	[45-58]	6	[35-41]	14	[42-44]	51
[44-57]	3	[58 …	5	[42-44]	16	45	34
[58-…	2			[46-51]	10	[46-53]	30
				[52-58]	11	[54-58]	31
				[59-…	10	59	28

Table 4 shows similar results for the K=50 dataset. Clearly, the number of connections is higher. We attribute this to the fact that there are less options to deploy services and resolvers compared to the K=1050 locations, meaning that the placement is less optimal.

Table 4 Total number of connections between zones and resolvers for services with maximum utility. Resolver and zone locations are chosen from the K=50 dataset.

2 resolvers		**5 resolvers**		**10 resolvers**		**28 resolvers**	
latency	*no. of conn.*	*latency*	*no. of conn.*	*latency*	*no. of conn.*	*latency*	*no. of conn.*
[0-27]	not feasible	[0-27]	not feasible	[0-27]	not feasible	[0-27]	not feasible
28	8	28	16	28	25	28	64
[29-30]	7	[29-30]	13	[29-30]	20	[29-30]	52
[31-41]	5	[31-41]	10	[31-34]	15	[31-41]	39
[42-44]	4	[42-44]	8	[35-41]	14	[42-44]	37
[45-60]	3	[45-60]	6	[42-44]	13	[45-60]	30
[61…	2	[61-…	528	[45…	10	[61…	28

Table 5 and Table 6 provide an overview of the total reporting traffic that flows between the execution zones and the resolvers, for various reporting frequencies and number of services. By comparing these tables to the results for the centralized case (Table 2), we see that the amount of signalling traffic is similar only for the 2-resolver scenario. As soon as there are more resolvers, the amount of signalling traffic exceeds the signalling traffic of the centralized case.

Table 5 Total reporting traffic between execution zones and resolvers. Results obtained from the K=1050 dataset with possible locations for resolvers and execution zones.

No. resolvers	no. services	T = 0.5s	T = 1s	T = 30s	T = 60s
2	2×10^6	1.075 Gbps	537.7 Gbps	17.9 Mbps	8.96 Mbps
	1×10^9	537.7 Gbps	268.8 Gbps	8.96 Gbps	4.48 Gbps
5	2×10^6	2.30 Gbps	1.15 Gbps	38.3 Mbps	19 Mbps
	1×10^9	1151 Gbps	575.6 Gbps	19.19 Gbps	9.59 Gbps
10	2×10^6	3.64 Gbps	1.82 Gbps	60.6 Mbps	30.3 Mbps
	1×10^9	1818 Gbps	909.1 Gbps	30.3 Gbps	15.15 Gbps
28	2×10^6	9.91 Gbps	4.95 Gbps	165.1 Mbps	82.6 Mbps
	1×10^9	4954 Gbps	2477 Gbps	82.57 Gbps	41.28 Gbps

Table 6 Total reporting traffic between execution zones and resolvers. Results obtained from the K=50 dataset with possible locations for resolvers and execution zones.

No. resolvers	no. services	T = 0.5s	T = 1s	T = 30s	T = 60s
2	2×10^6	1.22 Gbps	0.61 Gbps	20.3 Mbps	10.1 Mbps
	1×10^9	608 Gbps	304 Gbps	10.14 Gbps	5.07 Gbps
5	2×10^6	2.46 Gbps	1.23 Gbps	41.0 Mbps	20.5 Mbps
	1×10^9	1230 Gbps	615.4 Gbps	20.51 Gbps	10.26 Gbps
10	2×10^6	3.84 Gbps	1.92 Gbps	64.0 Mbps	32.0 Mbps
	1×10^9	1920 Gbps	959.9 Gbps	32 Gbps	16 Mbps
28	2×10^6	10.81 Gbps	5.41 Gbps	180 Mbps	90 Mbps
	1×10^9	5405 Gbps	2703 Gbps	90 Gbps	45 Gbps

Obviously, this is again only a first order approximation. Several refinements can be made to this model. Overall, we tried to minimize the number of DCs in use. All services with the same utility range are deployed on the same DCs. In reality, each service provider (or orchestrator) could/will use a different set of DCs. This means that resolvers will have to subscribe to more execution zones than considered. However, we think the differences would be minimal, since the contribution of the header of the session report in the total report byte size is minimal compared to the number of sessions.

Another important factor that should be studied further is the effect of the long-tail. Each service that tolerates high utility, is only deployed in one (or a handful) execution zones. This means that every

resolver worldwide must subscribe to the zone manager of that execution zone, which increases the stretch of the traffic.

4.4 Impact of session slot reporting interval on resolution

In the previous section, we studied the impact of the session slot reporting interval *T* on the resulting signalling traffic. However, the amount of signalling traffic that is considered acceptable depends on the achievable performance of the resolution. Lower values of *T* will lead to higher signalling traffic, but will also result in the resolvers have a more up-to-date view on the number of available session slots in each of the EZs. In this section, we focus on the dynamics of the service resolution in terms of the number of failed requests.

The study is performed using a discrete event simulator. This simulator allows us to assess the impact of various parameters, like demand distribution, session length, number of session slots per instance, etc.

4.4.1 Discrete event simulator

4.4.1.1 Architecture

A discrete event simulator models the operation of a system as a discrete sequence of events in time. Events are generated by components in the modelled system, and each event marks a change of state in the system, or generates one or more new events. Each event occurs at a specific moment in time, and the simulator keeps all events in chronological order in a queue. During the execution of an event, the internal time clock of the simulator is halted. After the event is executed, the simulator takes the next event from the queue and advances it internal time clock accordingly.

Figure 13 shows the different types of components in our simulator, as well as the type of events that flow between each of these.

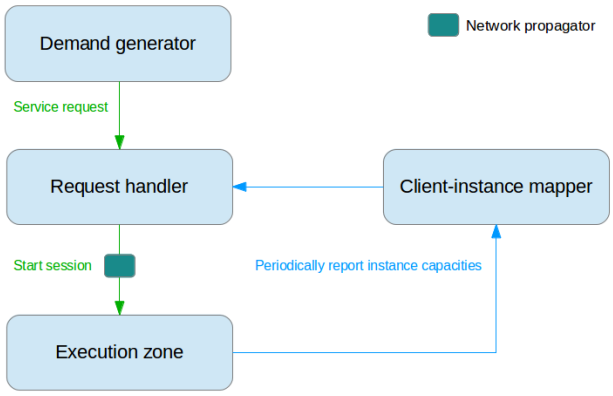


Figure 13 Flow of events between the different components in the simulator

Service demand is generated from different locations (e.g. cities), which we refer to as demand nodes. In our simulator, multiple **demand generator** components can be created. Each demand node (client node) generates *service requests* according to a given statistical distribution (demand pattern). This distribution is adjustable per component, which is a useful feature to model spatial and temporal differences in service popularity, e.g. non-English services or day/night variations due to time zone differences. A service request contains the ID of the requested service, the ID of the requesting client node, and the length of the session. The session length is determined by a node-specific statistical model that the experimenter must provide.

The simulator delivers the service request events to a **request handler**. The choice of request handler is again a configuration parameter of the system. Supporting the two models outlined in section 4.1,

either a demand generator component sends all its service requests to the same request handler; either it determines the appropriate handler by the service ID of the request. The request handlers select the most appropriate available execution zone for a given request, based on a mapper table that is updated periodically by the **client-instance mappers**. One client-instance mapper can update the tables of one or more request handlers, but obviously one request handler is only updated by a single mapper.

The instance mapper(s) incorporates the logic of the resolution process. For each pair of client node ID and service ID, it calculates a weighted list of suitable execution zones. The request handler will carry out a weighted random sampling to select an execution zone from this list, and then generate a *start session* event, which is consumed by the selected execution zone. This event is scheduled to arrive at an **execution zone** component after a given delay to simulate the network delay between client and execution zone.

The instance mappers are updated periodically with the session slots of the execution zones. This information is used to construct forwarding tables for the request handlers. The request handlers themselves are not aware of the number of available session slots of an execution zone, they perform mere lookups in the forwarding table and carry out weighted random sampling, as explained above.

When a *start session* event arrives at time *t*, the **execution zone** component will decrement its number of session slots for the requested service. A *session ended* event is scheduled at *t + T*, with T the session length specified in the request. This event will increment the number of available session slots again. In case there are no session slots available when the event arrives, the session is rejected and a corresponding entry is added to the simulator logs.

The execution zone reports at regular intervals to the client-zone mapper(s) how many session slots it has available. Depending on the chosen resolver deployment model the session slots are reported to a single resolver, or several reports are sent out to different resolvers.

4.4.1.2 Implementation

The simulator is implemented using **SimPy**, a process-based discrete-event simulation framework based on standard Python. It can be flexibly configured with models for demand generation, network topology and service placement.

Before running a simulation, the following **parameters and input data** need to be specified:

- reporting frequency of the execution zones to the client-zone mapper(s) on the number of available session slots per service
- update frequency of the request handler forwarding tables, calculated by the client-zone mappers. In the future, we could also consider to trigger recalculations not only on a periodic basis, but also if specific events occur (e.g. the number of session slots drops below a predefined threshold)
- latency matrix, containing the delay between every demand node ID and each execution zone ID. By default this table will only be consulted to simulate the delay between those two. However, the implementation of the network propagator allows us to introduce delays on every message that is sent with minimal effort. So this matrix can be extended to introduce delays between demand nodes and request handlers, execution zones and instance mappers, and so on.
- a file describing for each demand node which request handler(s) it should send service requests to.
- a file describing the client-zone mapper(s) that an execution zone needs to reports its available session slots to
- a file describing which request handler(s) are updated by which client-zone mapper
- a file describing the service placement: which service IDs are available on which zones, and how much session slots are available for a given service ID on a given zone.

- a file describing for each demand node the statistical distribution of the service requests to be generated
- a scaling factor, to scale down the service request rate from all nodes by the same factor. This allows to run large-scale simulation much faster because the total number of events is reduced, while keeping the relative differences between different nodes.

4.4.2 Session slot allocation

We use the demand model described in 4.2.1. Demand is generated according to a Poisson process proportional to the city population. To study the reporting overhead, it was only important to know in how many EZ an instance of a given service is deployed. Obviously, to study the impact of the reporting period on the resolution performance itself, we need to determine how many session slots should be provisioned in each of these zones.

We use a simple algorithm as follows:

- For each demand node, we select the N=5 closest EZ based on Haversine distance - the shortest distance between two points around the planet's surface [BRUM13] from the set of K=50 or K=1050 datacentre locations.
- Assuming an average session length T, the average number of session slots needed to serve the demand from a city generating demand according to a Poisson distribution with parameter λ is: $\lambda \cdot T$
- Based on the required session slots of the demand node (call S_i), each of those N=5 zones is allocated with $\lceil S_i \cdot N \rceil$ session slots, in addition to the ones possibly allocated for other cities. Note that the zone nor the resolver are aware of how many of its total session slots should be used for each city.
- Repeat the process for next demand nodes.

By using this placement algorithm, we will find good locations for zones and also guarantee enough session slots at each EZ to serve user demands.

4.4.3 Resolution algorithm

When a resolver receives a service request, it will consider the N data centres with the best utility (lowest latency) towards the requesting client. This subset is further reduced by excluding from this list execution zones that reported zero available session slots in their last session slot update. Then, a datacentre is chosen using one of two options:

- **uniform:** a datacentre is randomly selected. This option is basically ignoring session slot updates, apart from excluding datacentres with zero session slots available to be selected
- **weighted:** according to the number of available session slots. The resolver is more likely to select datacentres with a higher number of available session slots.

The uniform algorithm is a good benchmark to study the situation where the resolver has knowledge of the logic that was used during the placement algorithm. In the weighted case, some unexpected side-effects may occur, as the weights are calculated on the total number of (available) session slots advertised by each zone.

To better understand this, consider the situation where there are only 3 datacentres (DCs). For a specific service, DC1 and DC2 are the only ones within the utility range of city A, while DC2 and DC3 are the only datacentres with session slots and within the utility range of city B. Now suppose that both cities generate the same demand, requiring 250 session slots. This means that the placement algorithm will place 125 session slots in DC1 (for city A), 250 session slots in DC2 (125 for city A, 125 for city B), and 125 session slots in DC3.

Ideally, the resolver would resolve half of the requests from city A to DC1, and half of the requests to city B. However, in the weighted case, it will see 125 session slots advertised by DC1, and 250 session slots by DC2[7]. Thus, DC1 will be selected with probability 1/3 (125/375), while DC2 will be selected with probability 2/3. A similar situation occurs for the requests from city B. This means DC2 will be oversubscribed from both city A and city B. It will report a very low number of session slots, meaning that the resolver will update its forwarding tables and send almost all requests to DC1 and DC3. In turn, this results in DC2 being heavily undersubscribed… To damp these oscillations, the resolver applies exponentially weights to the session slot reports. In particular, instead of using the most recent session slot report of an execution zone, it uses a value SS_{zone} that is calculated as follows:

$$SS_{zone} = \alpha \cdot SS_{most\ recent\ report} + (1 - \alpha) \cdot SS_{zone}$$

In contrast, the random case will uniformly sample between DC1 and DC2 for city A, and between DC2 and DC3 for city B. This resolution policy is perfectly in line with the initial placement.

4.4.4 Results

The simulation results have been suppressed in the public version of this report as this content is currently under review for publication in a scientific journal.

[7] Although we make our argument assuming the starting position of the simulator, an identical conclusion can be drawn when considering regime after a number of session slot updates.

5. SERVICE RESOLVER DESIGN

5.1 Architecture

In Figure 14, the reference architecture of FUSION resolver is depicted. It has been designed having in mind also the requirements for the prototype implementation. In particular, the *service request handler* and *client-to-service-instance mapping* functional blocks have been clearly distinguished from each other to allow for independent implementation by different partners of the resolution and optimization functions.

Throughout the rest of this section we explain briefly the overall operation of the architecture. Its basic functional blocks are described in more detail in sections 5.2, 5.3 and 5.4. The main interfaces are described in ANNEX 2.

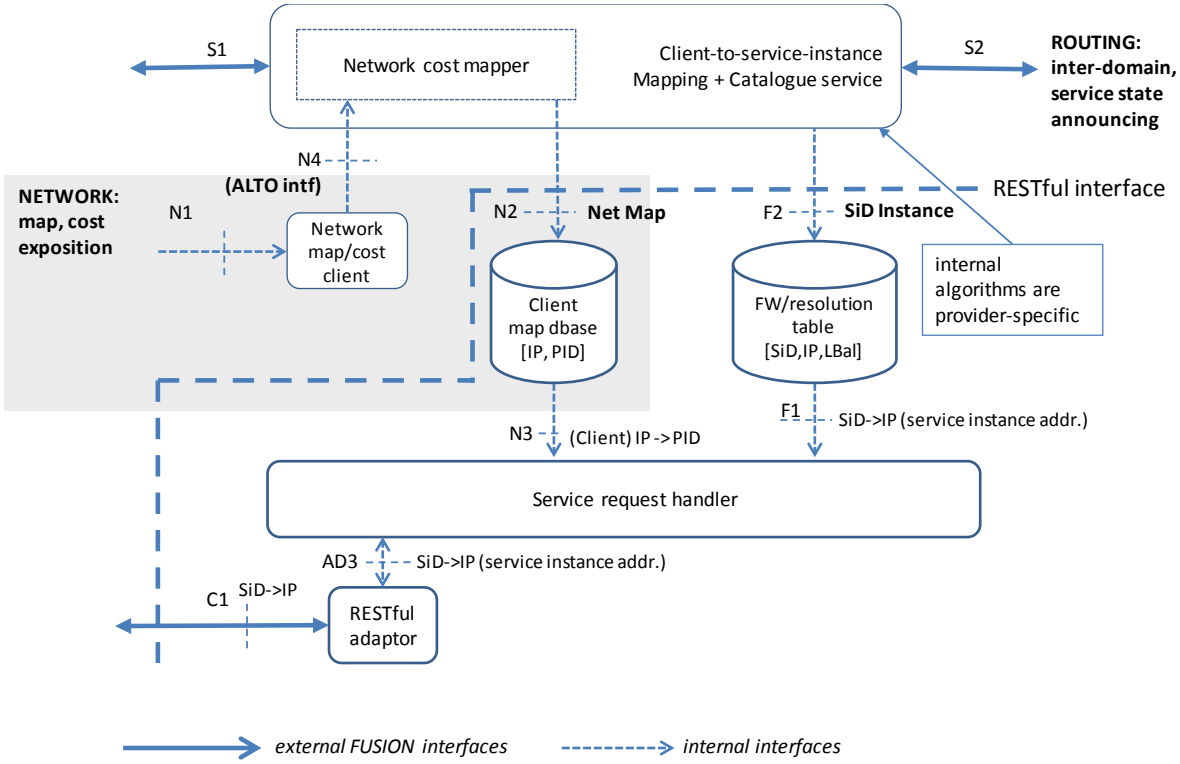


Figure 14 Reference architecture of the resolver

The overall idea of the design is to separate long-term operations (e.g. recalculating forwarding tables) from short-term operations (e.g. resolving requests on per-request basis). These blocks are shown in the figure as *Client-to-service Instance Mapping* function and *Service Request Handler* function respectively. They communicate with each other through a database shown in Figure 14 as *FW/resolution table* (Forwarding/resolution table). The instance mapping sets the entries in the FW/resolution table, and this information is read by the Service Request Handler. The format of the entries and the interpretation of load balancing rules are further discussed in section 5.3.

The ***Client-to-service-mapping*** functional block is responsible for handling service routing information, i.e., communicating with:

a) execution zones (i.e. their zone gateways) and/or upstream resolvers via interface S2 to receive information on the state of service instances in particular zone, and
b) downstream resolvers via interface S1 to propagate (possibly processed) information on service instances needed for resolution purposes, e.g. the number of available session slots.

The protocol specification of interfaces S1 and S2 are similar and their descriptions can be found in deliverables D3.2 and D3.3. The Client-to-service-mapping has also access to the network state information via interface N4 which takes the form of IETF-ALTO interface[8] according to RFC 7285.

We do not describe in detail the design of *Network map/cost client* function, because from the perspective of client-to-service mapper this function is simply the role of ALTO server and existing work can be used.

The *Service Request Handler* is responsible for handling client requests received on the C1 interface. To this end it accesses the FW/resolution table using the F1 interface. Interfaces N2 and N3 are both related to the Service Request Handler but are not further specified. In a fully-fledged implementation of the architecture their role is to provide the S*ervice Request Handler* with a network map that can be configured by the *Client-Service-instance-mapping* in compliance to its policies. This way IP addresses of requesting users can be mapped onto PIDs (ALTO Provider-Defined Identifier) that are configured in the FW/resolution table. In the current implementation, developed for the needs of the demonstrator, the *service request handler* obtains network maps by directly accessing the *Network map/cost client* via interface N4.

5.2 Client-to-service-instance mapping service

The current implementation of this block uses Python and is internally organised into three interworking components as specified in the following:

(1) A script to connect to ALTO server to get latency metrics (RESTFUL interface).
(2) An optimization component which gets the latency input from (1), then optimizes the cost-utility, and writes the output to a file.
(3) Another script that reads the output file in (2), connects to the resolver via a RESTFUL interface to update the forwarding table.

5.3 Forwarding/resolution table

5.3.1 Information model and operation

The *Forwarding/resolution table* is a structure used by the resolver as information base for service resolution. The information is organized around services (or service types, identified by service identifiers, SID), which are keys in the database, and groups of clients (called Endpoint Groups, also keys in the database) identified by their PID (Provider-Defined Identifier as defined by IETF ALTO). It is manageable by its “Mapping interface” described in ANNEX 2.

The following diagram presents the logical organization of the information in the forwarding table.

[8] Limited functionality implementation developed for the needs of FUSION demonstrator.

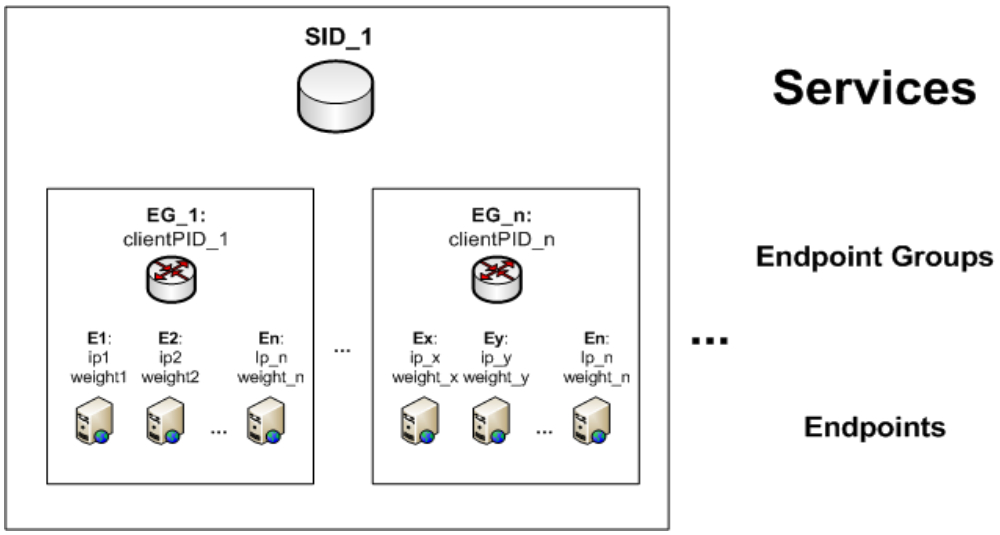


Figure 15 Structure of the forwarding table.

For a given service (e.g., SID_1 in the figure), the Endpoints of this service (i.e. as E_1, E_2, …, E_n) correspond to service instances. It is assumed that each service instance E_i has a unique IP address ip_i. A subset of all Endpoints (or instances) of a given service can be grouped into an endpoint group (EG_i). Each EG_i is then assigned one client PID ($clientPID_i$ in the figure), following the semantics of PID in the IETF ALTO framework.

Within a given service endpoint group EG_j, each service endpoint E_i is assigned a weight $weight_i$. Weights are then used by the resolver to load balance requests from clients among service instances belonging to a given service endpoint group. This way for a given grouping of clients defined by its $clientPID_i$ and a given service type SID_j all related service instances and respective load balancing parameters can be found.

Weight type is unsigned integer. It means that there is a possibility to assign negative values to the weight, but only endpoints with a weight greater than 0 will be considered for processing. The value 0 and negative values may be used to temporarily disable the endpoint (exclude from processing) without the necessity to delete it from the data base.

There is full flexibility to configure endpoints inside endpoint groups. In particular, a given service instance *n* (ip_n in the figure) can be used by (assigned to) one or more EG-s. Clearly, a given service instance can have different weights in different EG-s.

Resolving client request for a given service to the right EG is achieved through finding the set of records related to the service and then matching IP address of the requesting client to clientPID parameter based on the longest match principle.

5.3.2 Implementation

5.3.2.1 Database

To store the resolver's data, the PostgreSQL database has been used.

PostgreSQL offers data types to store IPv4, Ipv6 (inet, cidr types), and MAC addresses (macaddr). It is better to use these types instead of plain text types to store network addresses, because these types offer input error checking and specialized operators and functions (see https://www.postgresql.org/docs/9.4/static/functions-net.html). The types mentioned are necessary to properly manage resolver's data: network subnets (as part of network PID), and endpoint address (location IP).

The resolver implementation uses JPA (Java Persistence API) as the interface between itself and the database. As the JPA provider, Hibernate is used (the default JPA provider in Wildfly application server).

Note on the implementation constraint: inet and cidr types are specific to PostgreSQL database only, and do not have a direct equivalent in Java. Java's String field (of a class) has to be mapped manually to inet or cidr by overriding SQL statements[9] (INSERT and UPDATE). Opposite mapping is done automatically.

5.3.2.2 Runtime environment

The following components have been used to build Resolver's runtime environment:

- OS: Debian GNU Linux 8.0 amd64 (64-bit architecture)
- Application server: Wildfly 8.2.0 Final[10]
- Database: PostgreSQL 9.4[10]

5.4 Service request handler

Interfaces to this function are described in ANNEX 2.

5.4.1 Operation

In the following, we present the flow of activities performed by the FUSION request handler when a client request is received.

Note: Currently, only RESTFul API is available. DNS handler can be implemented if needed.

The process of resolving service name into location (IP) consists of several steps:

1. At the start of the resolution process there is a matching of the client's IP address to the proper PID(s). This may be the real client IP or overrode by "client" query parameter.

2. Then, the PIDs found in step 1 are used to match endpoint group(s) (EG) in service (SIDn).

3. If more than one EG, that matches the client IP, is found in service (SID), the PID containing the subnet with the longest mask (that matches the client IP) is chosen.

> Example:
>
> There are 3 PIDS: EUR with subnet 10.10.10.0/24, POL with 10.10.10.0/25, and WAW with 10.10.10.0/26.
>
> There is a service (SID), i.e. www.orange.pl with two endpoint groups defined: EUR, POL.
>
> Client IP is: 10.10.10.10.
>
> The matched EG in service www.orange.pl will be POL, as it has longer mask in matching subnet.
>
> *Note: The longest match for PIDs only points to WAW, but it is not defined in our SID.*

4. If no EG is matched, no location will be returned.

5. After EG is chosen, list of endpoints that belongs to the chosen EG is processed to find location.

6. The list of endpoints is divided into two groups by type of endpoint's IP address (IPv4, IPv6) and after that processed separately (if necessary). Only the endpoints with weight greater than 0 are taken into

[9] Java class containing field, mapped to inet or cidr, has to be annotated with @SQLInsert, @SQLUpdate (Hibernate specific), which define overriden SQL statement syntax.

[10] Low-level communication with database is ensured by application server (and not Resolver), so appropriate data source configuration and JDBC driver deployment has to be done by application server's administrator.

account. The sum of weights and hits of endpoints taken are computed. Sum of weights is used to normalize them.

7. Based on endpoint's weight from step 6, and total number of hits, the final location of the service are computed (both types, IPv4 and IPv6, if necessary).

8. The value of hit counters for chosen endpoints are incremented and they are saved in DB.

5.4.2 Implementation

5.4.2.1 Programming Environment

Service request handler application is implemented as a Maven project using Eclipse 4.5.0 (Mars) JEE Integrated Development Environment (IDE). The resolver code will made available publicly according to the regulations adopted by FUSION consortium.

The following list contains the main components used in the project (including Eclipse's plugins):

- Java – openjdk-8-jdk (version 8u45),
- Git (2.1.4) – version control system software,
- Gitlab CE (8.6.4) – versioning repository manager,
- Maven Integration for Eclipse WTP (1.2.0) – to ensure maven support,
- Egit Team Provider (4.3.1) – to provide versioning system and interface between IDE and Gitlab,
- Jboss Tools (4.3.1) - a set of plugins that complements, enhances and goes beyond the support that exists for Jboss/Wildfly and related technologies,
- Jfrog Artifactory (4.7.1) – Maven artifact repository manager.

5.4.2.2 Runtime environment

The following components have been used to build the Resolver's runtime environment:

- OS: Debian GNU Linux 8.0 amd64 (64-bit architecture)
- Java: openjdk-8-jre version 8u45
- Application server: Wildfly 8.2.0 Final[10]
- Database: PostgreSQL 9.4[10]

5.5 Network map/cost client

Interface to this function is described in ANNEX 3.

5.5.1 Operation

This function gathers information on network state on interface N1 (in any source – from network measurement systems, probes, etc.) to calculate network delays (in our case) and present them in the form of ALTO network costs maps on interface N4. Moreover, this module offers a configuration interface (not shown in the reference architecture, but described in ANNEX 3) that allows manual setting of the values of respective metrics for testing purposes.

5.5.2 Implementation

This module has programming and runtime environments similar to those for Service request handler, and enhanced with the features already known from Forwarding/resolution table module that are needed to use a PostgreSQL database (used to store network cost/map information).

6. DISTRIBUTED RESOLUTION

In this section, we present a new method for selection between replicated servers distributed over a wide area, allowing application and network providers to trade-off costs with quality-of-service for their users. First, we create a novel utility framework that factors in quality of service metrics. Then we design a polynomial optimization algorithm to allocate user service requests to servers based on the utility while satisfying transit cost constraint. We then describe an efficient - low overhead distributed model with the need to only know a small subset of the data required by a global optimization formulation. Extensive simulations show that our method is scalable and leads to higher user utility compared with mapping user requests to the closest service replica.

As the Internet becomes the enabler for more types of services with a wider spectrum of requirements, pressure is being put onto the Internet ecosystem to facilitate service placement and to select the best replica for each user request at each instant in time. This replication always involves multi-stakeholder trade-offs involving costs (deployment and traffic related) and user quality of service (QoS).

There are many drivers for service replication, including server resilience, network diversity, and proximity of servers to users. Deploying services closer to the users allows the application providers to improve on QoS metrics like latency and/or throughput for all users. Some frameworks, like fog computing [BMZ+12], even attempt to put service instances at the extreme edge of the network in locations such as access points.

Although in theory this replication could be optimal, in practice there are several obstacles: deployment costs vary between geographical areas and may be prohibitive in some locations, demand forecasting is inaccurate, flash crowds are unpredictable. Efficient allocation of user requests to service replicas will have to rely on a service selection at query time.

Service quality has two major sets of component metrics, relating to computation and networking parameters. Servers will have to be properly provisioned for the arrival rate and holding time of user requests otherwise users will be not served or blocked, increasing application latency. Network distance will have primarily an effect on end-to-end delay but, in many scenarios, causes an increase in packet loss and/or a decrease in good-put. The service selection system will have to take into account both computation and networking factors to optimize its selection.

Resolution involves converting a service name to a specific network locator for the selected replica. Our work assumes that the user's ISP is in the best position to make this selection. The ISP has accurate information regarding the user's position in the network, the current network status and, furthermore, it allows the ISP to apply traffic network policies in the selection process to reduce traffic costs. A centralized approach would, in theory, allow global optimization but it would often be unscalable and unrealistic. A central entity would not have access to information on the detailed user position, the network topology or current network status and would be incapable of implementing ISPs' specific traffic policies as it would have to arbitrate between conflicting policies of different ISPs which would be problematic from a business point of view. For those reasons, our server selection model can be implemented in a similar way of PaDIS [PSF+11] which allows ISPs to better assign users to servers by using their knowledge about network conditions and user locations.

In brief, the contributions of this work are as follows:

- Firstly, we introduce the *utility function* relating to one or more QoS metrics that allows application providers to define based on their application's requirements.
- Secondly, we design a *polynomial centralized optimization algorithm* that allows ISPs to redirect their users to the best replica, allowing to trade-off their traffic costs with users' QoS. In addition, the model allows optimizing for multi-services at the same time.

- Finally, we propose a *simple - efficient distributed model* that allows ISPs to run a local version of the selection algorithm without the need for global knowledge of all service replicas and network conditions.

The detail of the architecture, mathematical formulation of the distributed resolution algorithm and the experimental results have been suppressed in the public version of this report as this content is currently under review for publication in a scientific journal.

7. USE CASE: WEB-RTC

Orange has been working on developing an architectural support for WebRTC services to leverage its knowledge of the network and its state in order to improve the QoE of the users. This will be achieved by selecting appropriate instances of TURN server and by intelligent orchestration of those instances assuming their distribution across the network. In the following, we provide an overview of the concept of the system being designed in Orange and intended to be launched in a prototype version. Of course only positive feedback from testing the prototype and further strategic considerations of different technical and business aspects can lead to a decision about deploying the solution on a commercial basis.

In the following subsections, we provide a brief description of the WebRTC concept with the emphasis on the role of TURN servers in this architecture, characterise the possible role of FUSION architecture on the design of our platform and provide an overview of the resulting architecture of TURN Servers Management and Orchestration system.

7.1 WebRTC communication and TURN servers

The sections below highlight the WebRTC concept and the role of TURN servers in WebRTC communication.

7.1.1 The principles of WebRTC communication

Modern web browsers are not only used for rendering of web pages, but are also standalone application platforms that are able to natively interpret and execute HTML, JavaScript and CSS code. They are constantly modernized in order to assure better performance and functionality. One of the biggest milestones in last years is the exposition of networking functions and video/voice peripherals over a web browser API to web applications. As a result, web applications are not only able to communicate with the servers that host the content presented in web pages, they can also *initiate* their own additional data steams based on TCP and UDP (see Figure 16). This offers an opportunity to create web-technology based applications that before were reserved only for desktop environments.

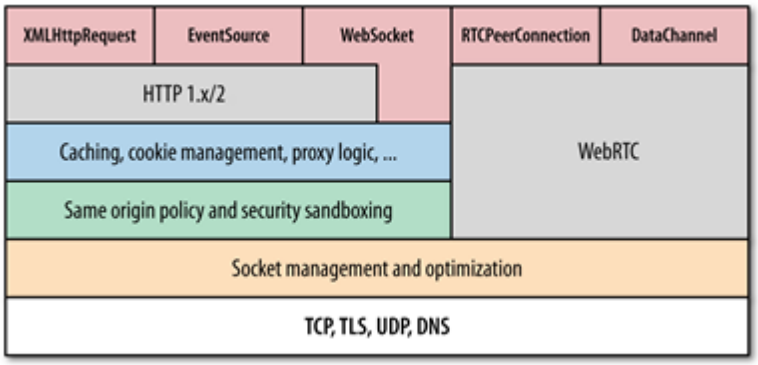


Figure 16 Networking mechanisms, services and protocols available in modern web browsers (source [ORRTC]).

The *WebSocket* API allows creating TCP/TLS based connections that can be used for guaranteed data delivery or for signalling purposes, e.g. SIP signalling. The *RTCPeerConnection* and *DataChannel* can be used for unguaranteed data delivery designed for real-time traffic control and real-time data transmission. Such connections are used within the RTC communication that is provided in browsers thanks to the embedded WebRTC stack (see Figure 17Figure 17). This stack is responsible for real-time video and voice playback and capture capabilities directly by the browser, without additional plugins like Adobe Flash Player.

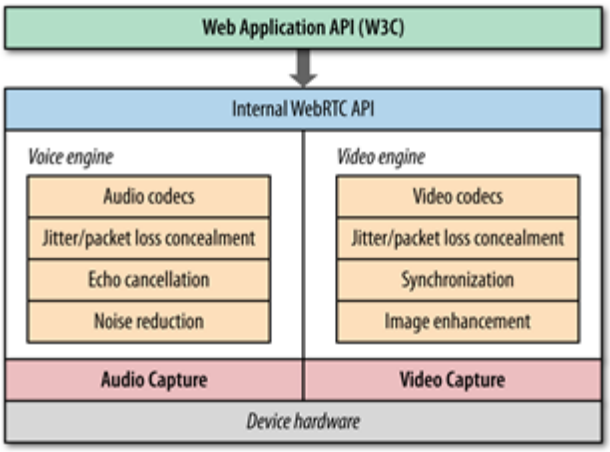


Figure 17 WebRTC API stack (source [ORRTC]).

The implementation of the *RTCPeerConnection* and *DataChannel* APIs assures native support for RTC transmission and control protocols like SRTP and SCTP. The ICE, STUN and TURN protocols – used as helper protocols for successful establishment of data communication between the peers over various types of networks – are also natively present in the WebRTC stack (see Figure 18). Availability of all this protocols and methods enables real-time video and VoIP communication directly from the web browser.

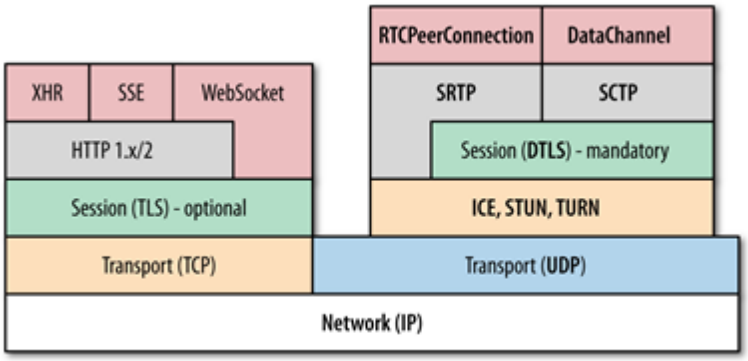


Figure 18 WebRTC protocols stack for Video/VoIP communication (source [ORRTC]).

The overall architecture of WebRTC solution, as seen from the implementation point of view, is presented in Figure 19. It is divided into the browser part (that is provided by browser vendors) and the application part. The implementation of the WebRTC stack in browsers is left open to the vendors but they have to embed the functionality mentioned above. Finally, within the browser an implementation of the W3C-standardized WebRTC JavaScript API has to be present. This script enables the development of WebRTC web applications that should able to work on all web browsers compliant with the specification.

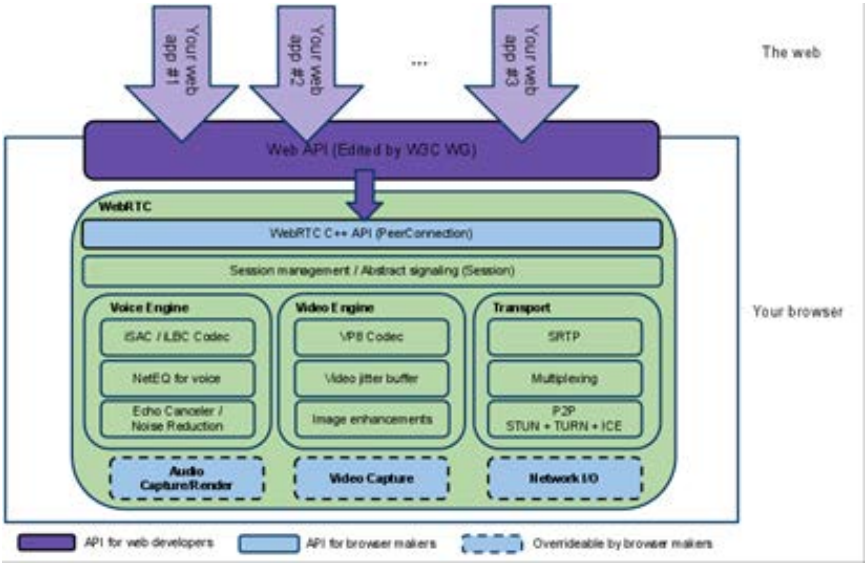


Figure 19 The WebRTC architecture (source [WRTCO]).

7.1.2 The role of TURN servers in WebRTC communication

The origins of STUN (Session Traversal Utilities for NAT) and TURN (Traversal Using Relays around for NAT) protocols and servers that implement them come from the problems of bidirectional data communication over the network when the endpoints are hidden behind the NAT (Network Address Translation) – especially in a case of UDP connections, see Figure 20. The natural consequence of adopting NAT is unreachability of the IP address of peers – when there is a need to communicate with them from the internet or, in general, from the network that is in front of the NAT protected network.

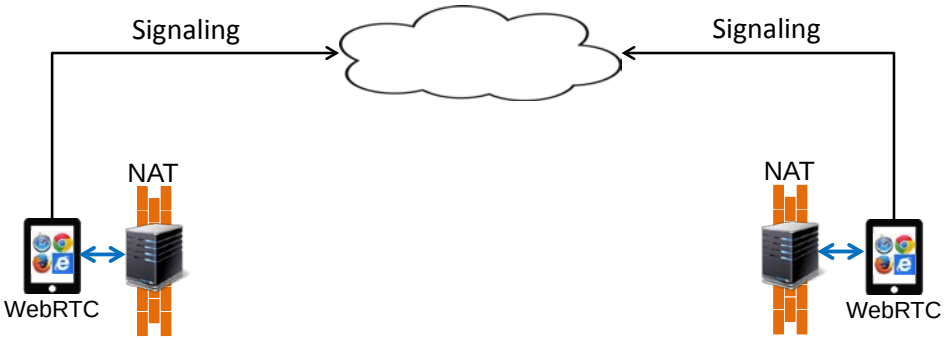


Figure 20 Visualization of the WebRTC data traffic problem with NATs.

The WebRTC based audio/video communication traditionally involves both signalling traffic and data traffic. Signalling traffic is used to initiate, close and modify parameters of the session and it is always handled with the help of a mediation point which in case of all-web solutions is an HTTP server. Such servers are always visible for the future call endpoint either, because they are assumed to reside in a network that is accessible for all the peers participating in the session. After the session negotiation phase is finished a data session that doesn't require any mediation point between the call endpoints is initialised. Such data session is always bidirectional (note that in case of multi-party call there is always a need for a mediation point). When one of the peers is behind a NAT it's impossible to setup such bidirectional communication. As we have already mentioned, STUN and TURN servers are used to cope with this problem but their ability to cope with NAT related problems depends on the type of the particular NAT combination [NAT].

The STUN server helps in most cases of bidirectional communication over NAT. STUN is used to learn about the IP address of peer as seen by peers located in front of the NAT. STUN protocol thus helps to find, for a tuple <IP address, port number> of a peer located behind the NAT its corresponding tuple

<IP address, port number> for incoming communication. In fact STUN server helps in the signalling phase of the connection establishment but doesn't participate afterwards in the data transfer phase (see Figure 21).

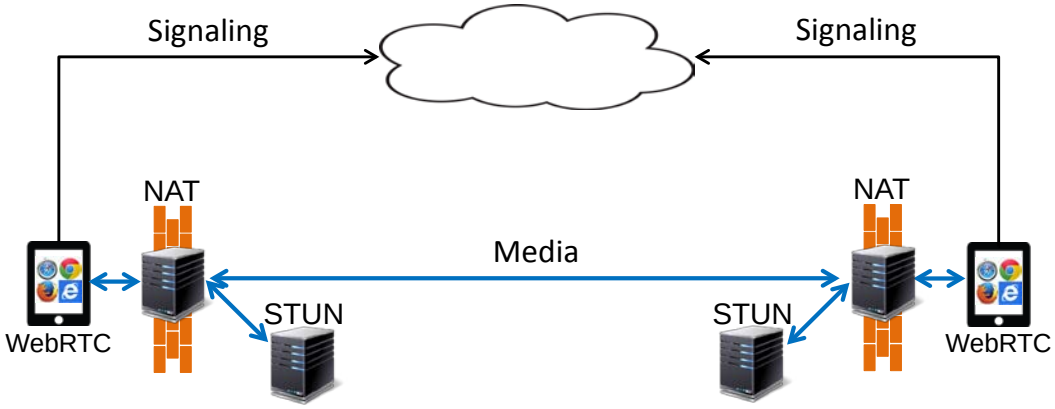


Figure 21 STUN supported media communication between peers.

The TURN server, in contrast to the STUN server, participates in the data transfer phase of the bidirectional communication during the whole session, as shown in Figure 22. The TURN server has to be visible by both peers and acts as a data proxy during the connection. Although its basic role is to cope with the NAT problem, adopting a TURN server can be also helpful to improve the quality of data traffic of clients and can be used to control selected network traffic classes in the operator's network, as described in the later in this section.

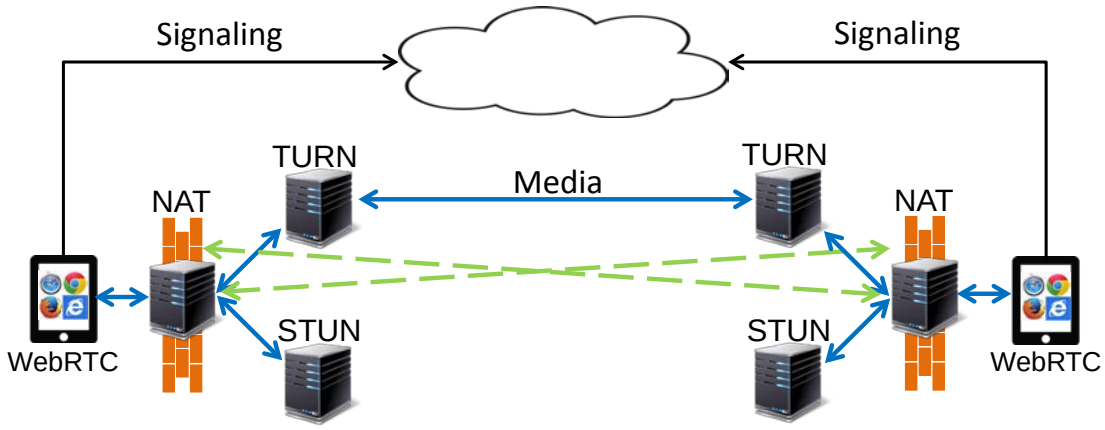


Figure 22 TURN supported media communication between peers.

The Interactive Connectivity Establishment (ICE) technique [ICE] (and ICE framework in WebRTC communication) is the last piece of the WebRTC peer-to-peer communication puzzle that helps to find the best connectivity method for media stream transmission. It is an extension to the offer/answer model that is present in peer-to-peer media communication. It adds special fields to SDP messages with alternatives (candidates) of IP-port tuples and leverages STUN and TURN protocols in order to find the best method for peer-to-peer communication. Using a set of procedures ICE tries to query the actual conditions of each possible peer-to-peer connection. In case of problems with the connection setup that result from the presence of NAT (or policies on routers, or firewall rules), ICE tries to find the best possible connection method. The search starts by trying a direct connection, and if needed it downgrades towards STUN support communication and finally, if needed, to the TURN server option. The ICE agent is typically implemented in WebRTC clients and delivers to the corresponding WebRTC

client the tuple <IPaddress-port number> of the best possible candidate for the requested peer-to-peer connection.

7.2 The impact of FUSION on the proposed architecture

The key concepts from the FUSION project that will be incorporated into the developed service architecture are:

- Service Resolver
- Zone Manager

The Service Resolver, based on the given set of service policies, network status, network service compute resource and network service QoS indicators, will propose the TURN server that guarantees a high QoS. Of course, such resolver can make decisions not only for WebRTC services, but for any service that demands optimal delivery of data through the network. We plan to reuse the resolver implementation being designed for the needs of the FUSION demonstrator, with appropriate adoptions if necessary.

The Zone Manager exposes underlying resources to the Service Resolver mechanism, like TURN servers in the given (geographical) area. We note that specific requirements of our WebRTC platform seem to call for a proprietary implementation of this component. Therefore the reusable part will be the architectural concept of the zone manager rather than the actual implementation used in the FUSION demonstrator.

7.3 Proposed architecture of TURN Servers Management and Orchestration system

The proposed architecture of the whole solution is a consequence of the decision to adapt various architectures and patterns for development of distributed applications, and to combine them together into a system for hosting and managing Virtual Network Functions[11] – with WebRTC service and TURN servers as VNF examples. These services should be implemented using distributed resources in multiple data centres. The architecture of such a system should be compliant with different service deployment patters and service architecture patterns. In the proposed solution the following concepts and technologies have been considered:

- WebRTC service communication and signalling patterns developed mainly by W3C organization and IETF standardization body and that show the correct placement of elements of WebRTC services and communication patterns between WebRTC clients, WebRTC Signalling Servers and STUN/TURN servers.
- WebRTC/TURN initial architecture proposed by ABM that helped to determine: (a) the key elements of the developed system and (b) how the interaction between the Service Provider and TURN Servers Network should look like.
- Architecture of the implemented system for the measurement of performance of TURN servers located in different DCs. It defines communication patterns between the WebRTC Service provider and TURN Servers Management System and how QoS measurement and QoS delivery of elements can be implemented.
- NFV reference architecture (with additional guidelines coming from OPNFV implementation) that shows the proper decomposition of the system for the management of VNFs into logical blocks.
- SOA systems reference architecture designed to implement distributed complex IT systems shows how interactions between its elements can be implemented. It also shows which

[11] As of the ETSI NFV architectural framework.

elements of IT infrastructure, that nowadays are available from the shelf, can be used to achieve this goal.

- FP7 FUSION architecture introduces concrete patterns for the decomposition of Service Resolution, Network Monitoring and Service Quality Monitoring modules into logical blocks. It also helps to define how to integrate them in order to deploy such a platform in a distributed cloud infrastructure.

As has been observed, the case of Distributed System for Virtual TURN Servers can be treated as a special case of a system for hosting Virtualized Network Services/Functions (or other services that for some reason can be hosted by Network Operator).

In the following sections our architecture will be presented along with the description of the roles of its logical elements and relations between them. Also a simplified model of interactions between them will be shown. This can be helpful to understand their role in realising a WebRTC session supported by TURN server(s) hosted in a distributed cloud system.

7.3.1 Logical domains

The elements of the proposed architecture can be decomposed into logical domains. Elements of the system are organised into groups that play different roles in the whole solution. They can also be scattered among different DCs (data centers) in case of deployment of the system within a distributed DC cluster. This organisation is shown in Figure 23.

Decomposition into logical domains introduces logical blocks of elements with their own responsibilities. They are mutually independent and interact with other domains over interfaces that are adopted to distributed environments. We note this decomposition is rather proprietary. Therefore all relationships between our solution and FUSION architecture have to be analysed carefully and taking into account that only selected components of FUSION are explicitly needed in our case. In the following, the main elements of this decomposition are explained.

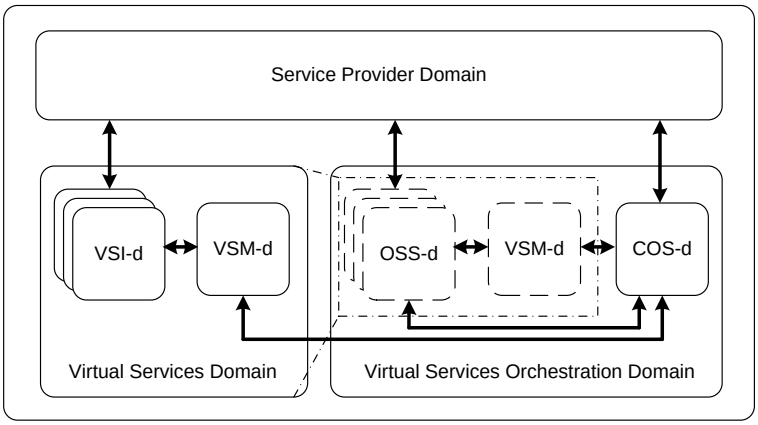


Figure 23 Domains available in the flow of communication.

1. Service Provider Domain (SP-D) – it contains the elements that instantiate the WebRTC Service that is provided by the WebRTC vendor and it comprises at least such elements as WebRTC Clients and WebRTC Signalling server. The last one is used for rendezvous and session parameters negotiation.
2. Virtual Services Orchestration Domain (VSO-D)
    a. Core Orchestration System subdomain (COS-d) – it manages the instances of Virtual services like TURN servers localized on distributed and virtualized cloud infrastructure. It provides mainly the functionality of distributed system management, virtualized services orchestration and management mechanisms, and virtualized services selection mechanisms. This subdomain should exist in one of the data centres selected for the management of the whole group of DCs and this subdomain is dedicated to

the management of all the components of the distributed system. For security and scalability reasons its elements can be replicated and spread across many data centres.

b. Orchestration Supporting Services subdomain (OSS-d) – it holds the elements that can be virtualized or not and that offer supporting services for Core Orchestration System. For example, the elements responsible for Service or Network QoS estimation and evaluation can be placed here. Some of these services can be exposed to the 3rd Party Service Providers like WebRTC Service Provider. The elements of this subdomain can exist in multiple data centres and can offer their services to any other element. These services can be managed by Core Orchestration System subdomain, but it is not mandatory and logically they can be treated as extensions of Core Orchestration System.

3. Virtual Services Domain (VS-D)
    a. Virtual Service Management subdomain (VSM-d) – it performs low level operations that are demanded by the Core Orchestration System and are devoted to the control and monitoring of instances of Virtual Services – e.g. Virtual Network Functions like TURN Servers or supporting services of COS. In particular, life-cycle management operations of these instances as well as all the aspects of their configuration are handled in this domain. The elements of this domain are multiplied and they exist in every data centre instance in order to manage Virtual Services located in different DCs.
    b. Virtual Service Instantiation subdomain (VSI-d) – it hosts (and instantiates) the instances of Virtual Services (e.g. Virtual Network Functions). The TURN Servers instances are hosted on cloud environment that is composed of distributed instances of OpenStack data centres. This domain exists in every data centre instance that offers its resources for our WebRTC/TURN server solution.

7.3.2 Functional architecture

Figure 24 shows the proposed functional architecture of the Distributed TURN Servers Management System. This architecture considers only minimal set of logical blocks that should be involved to assign TURN Servers to 3rd parties. This architecture reflects also the decomposition of the whole solution into logical domains from Figure 23. It also presents the distribution of the computing resources, Virtual Network Functions and their direct Managers into Zones (Zones correspond to different instances of data centres managed by different Virtual Infrastructure Managers).

The green colour highlights the elements of the whole solution (and their interactions) that are needed to realise services requested by end users. Blue colour highlights the elements responsible for the delivery and assurance of the elements that will be used to realize the service (and interactions between them. Note that more detailed flow of interactions between these elements will be shown in the next sections.

Key elements of the Distributed TURN Servers Management System are grouped within grey blocks.

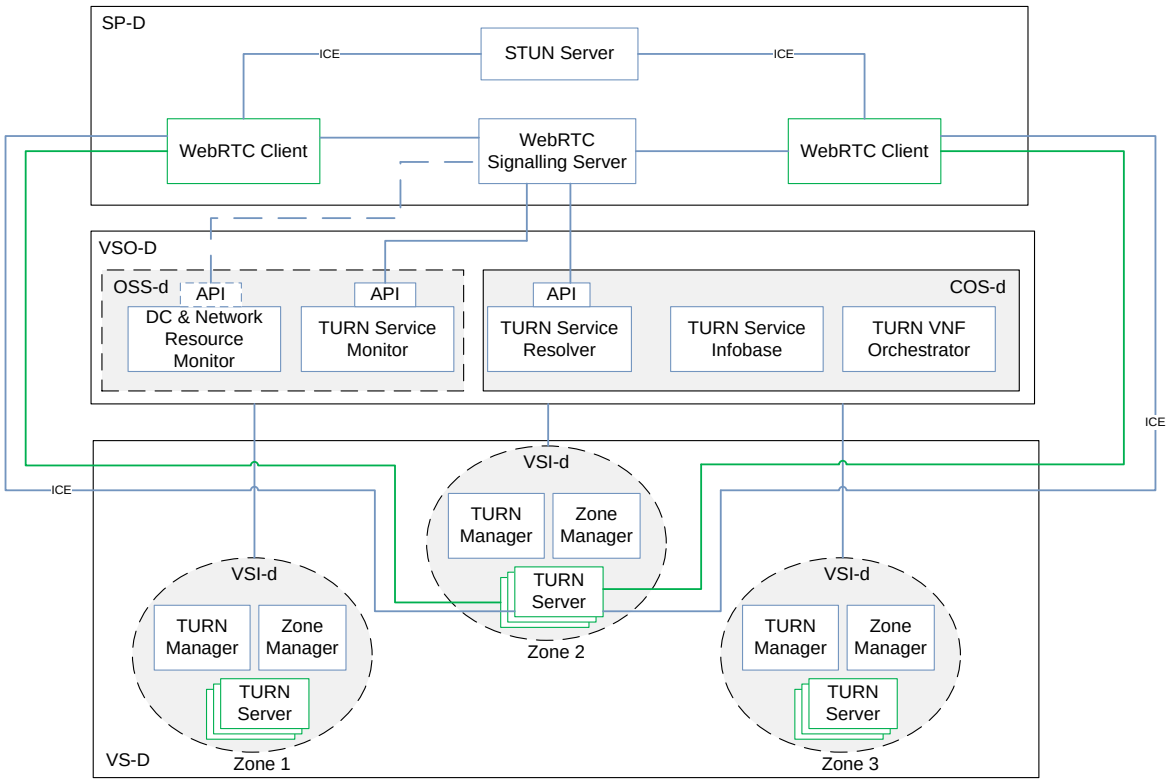


Figure 24 Functional architecture of Distributed TURN Servers Management System.

A simplified flow of actions is the following. The WebRTC client requests from the WebRTC signalling server the establishment of WebRTC connection. WebRTC signalling server uses its API to send getTURNList command to the TURN Service Resolver. It gathers all necessary information about the properties of the Service, profiles and about the available instances of TURN servers. It also interprets the parameters of the request and analyses historical monitoring information about the TURN servers' quality and performance.

Then the TURN Service Resolver makes a selection of a particular TURN Server (in a given zone) for service delivery and sends the answer to the WebRTC Signalling Server. The communication between the caller and the callee is established through selected TURN server. The TURN Manager controls and manages the operation of TURN servers and it is directly controlled by the Orchestrator. The performance QoS metrics are collected from TURN servers and indirectly from WebRTC clients, and are processed by Service Resolvers and WebRTC Signalling Server. Zones are geographical and/or logical locations (Data Centres) where the instances of TURN Servers are placed.

7.3.3 Mapping onto NFV architecture

The proposed architecture can be mapped into the NFV architecture. A top-level picture of such a mapping is shown in Figure 25.

TURN enabler is responsible for the selection of the appropriate TURN server as well as the overall TURN server management. Its logic is accomplished by such VNF Orchestrator, VNF Manager and Zone Manager taken together.

The selection of the TURN server will be based on predefined criteria to guarantee adequate QoS to end users. OSS/BSS comprises the elements responsible for the exposure of services to service providers and their management – especially to EMS or legacy management systems. TURN QoS VNF (Resource and Service Monitor), Service Resolver and TURN Servers can be treated as VNFs or VMs playing a role of supporting services. TURN VNF Orchestrator and TURN VNF Manager belong to the NFV Management and Orchestration domain.

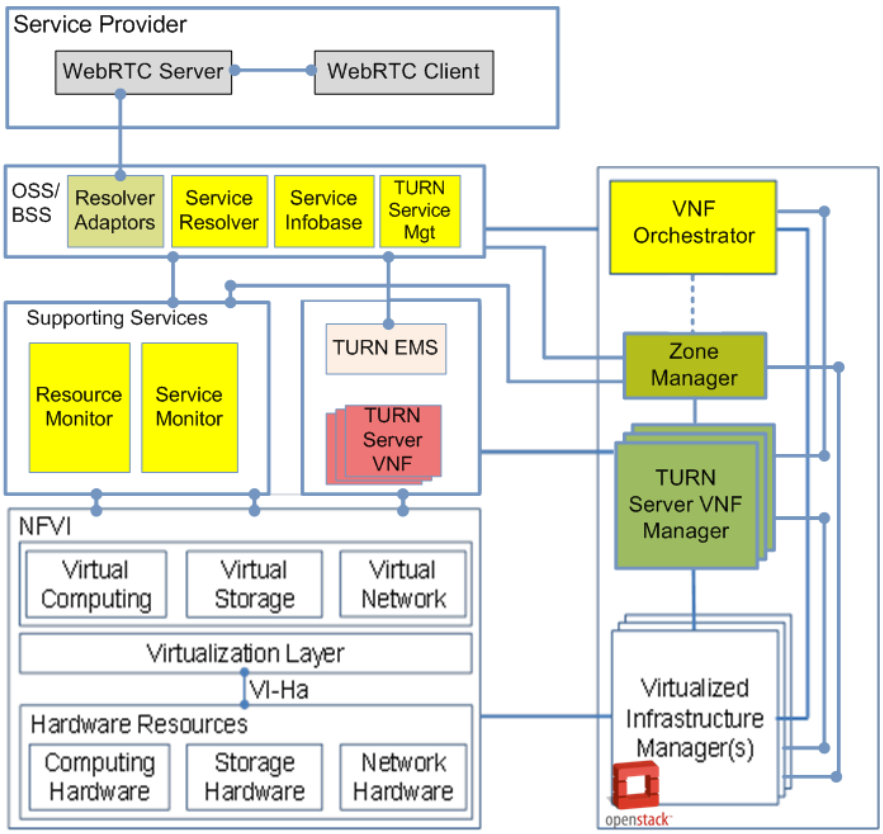


Figure 25 The TURN Server Management and Orchestration system mapped to NFV reference.

8. REFERENCES

[ALTO] RFC7285 Application-Layer Traffic Optimization Protocol

[ACBD2004] S. Agarwal, Ch. Chuah, S. Bhattacharyya, Ch. Diot. The impact of BGP dynamics on intra-domain traffic. In Proceedings of the joint international conference on Measurement and modeling of computer systems (SIGMETRICS '04/Performance '04). ACM, New York, NY, USA, 319-330, 2004.

[ASlinks] http://www.caida.org/data/active/ipv4_routed_topology_aslinks_dataset.xml

[BM06] S. Boyd and A. Mutapcic. “Subgradient Methods”, in Lecture notes of EE364b, Stanford University, 2006.

[BMZ+12] F. Bonomi, R. Milito, J. Zhu and S. Addepalli. “Fog Computing and its Role in the Internet of Things” in MCC, 2012.

[BRUM13] Glen Van Brummelen. “Heavenly Mathematics: The Forgotten Art of Spherical Trigonometry” in Princeton University Press, 2013.

[CPLEX] www-01.ibm.com/software/commerce/optimization/cplex-optimizer

[CSW+08] D. Carrera, M. Steinder, I. Whalley, J. Torres and E. Ayguade. “Utility-based Placement of Dynamic Web Applications with Fairness Goals” in NOMS, 2008.

[DATASET] https://github.com/richardclegg/multiuservideostream

[DCMAP] http://www.datacentermap.com

[ELKD2012] A. Elmokashfi, A. Kvalbein, C. Dovrolis. BGP churn evolution: a perspective from the core. IEEE/ACM Trans. Netw. 20, 2, 571-584, 2012.

[FET14] Fettweis, Gerhard P. "The tactile internet: applications and challenges."Vehicular Technology Magazine, IEEE 9.1 (2014): 64-70.

[FKH15] Fan, X., Katz-Bassett, E., & Heidemann, J. (2015). Assessing affinity between users and CDN sites. In Traffic Monitoring and Analysis (pp. 95-110). Springer International Publishing.

[FPS13] Frank, B., Poese, I., Lin, Y., Smaragdakis, G., Feldmann, A., Maggs, B., ... & Weber, R. (2013). Pushing cdn-isp collaboration to the limit. ACM SIGCOMM Computer Communication Review, 43(3), 34-44.

[GOA] L. Gao, On Inferring Autonomous System Relationships in the Internet, IEEE/ACM Transactions on Networking, December 2001.

[GCC+13] S. Gangam, J. Chandrashekar, I. Cunha and J. Kurose. “Estimating TCP Latency Approximately with Passive Measurements” in PAM, 2013.

[HEARINGAID] M.A. Stone and B.C. Moore. “Tolerable Hearing Aid Delays. Est. of Limits Imposed by the Auditory Path Alone using Simulated Hearing Losses” in Ear and Hearing, 1999.

[iPlane] http://iplane.cs.washington.edu/data/data.html

[JN93] Jakob Nielsen. “Usability Engineering: Response Times: The Three Important Limits” 1993.

[KPP15] Komosny, D., Pang, S., Pruzinsky, J., Ilko, P., & Polasek, J. (2015). PlanetLab Europe as Geographically-Distributed Testbed for Software Development and Evaluation. Advances in Electrical and Electronic Engineering, 13(2), 137.

[KT11] M. A. Khan and U. Toseef. “User Utility Function as Quality of Experience (QoE)” in ICN, 2011.

[LAC13] R. Landa, J. T. Araújo, R. G. Clegg, E. Mykoniati, D. Griffin and M. Rio, "The large-scale geography of Internet round trip times," IFIP Networking Conference, 2013, Brooklyn, NY, 2013, pp. 1-9.

[ORRTC] High Performance Browser Networking, Oreily, Chapter 14: Browser Networking, Chapter 18: WebRTC

[PSF+11] I. Poese, G. Smaragdakis, B. Frank, S. Uhlig, B. Ager and A. Feldmann. “Improving Content Delivery with PaDIS” in IEEE Internet Computing, 2011.

[RECO800] Recommendation G.114 (05/03) (https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-G.114-200305-I!!PDF-E&type=items)

[RIPEPRO] https://labs.ripe.net/datarepository/data-sets

[VERMO2014] L.Vermoesen, et al., “On the Economical Benefit of Service Orchestration and Routing for Distributed Cloud Infrastructures: Quantifying the Value of Automation”, Alcatel-Lucent Bell Labs Business Modeling White Paper, http://www.fusion-project.eu/publications/fusion_businessmodeling_quantifyingthevalueofautomation_whitepaper_v1.1.pdf, 2014.n

[WJF+10] P. Wendell and J. W. Jiang and M. J. Freedman and J. Rexford. “DONAR: Decentralized Server Selection for Cloud Services” in SIGCOMM 2010.

[XL13] Hong Xu and Baochun Li. “Joint Request Mapping and Response Routing for Geo-distributed Cloud Services” in INFOCOM, 2013.

[ZHZ+10] Z. Zhang, Y. Charlie Hu, M.Zhang, R.Mahajan, A. Greenberg and B. Christian. “Optimizing Cost and Performance Online Service Provider Networks” in NSDI 2010.

9. ANNEX 1 – MEASUREMENTS

9.1 Path variability in inter-AS domain routing

9.1.1 Data collection methodology

The dataset used in this study has been gathered using route reflectors (RR) installed in AS5617. It contains logs of all route update messages disseminated within this AS according to iBGP rules.

Network setup for AS5617 is depicted in Figure 26**Error! Reference source not found.**. As can be seen, this AS peers upstream with two transit network providers and downstream with customers in Poland. Actually, the logs have been taken from the central RR in the figure which maintains sessions with all border routers of the AS.

The data set contains logs from the central route reflector in AS5617 and covers 4 weeks of Sept. 2015 (starting from Sept. 1st) and the period from Jan. 8th to Feb. 8th 2016.

https://apps.db.ripe.net/search/query.html?searchtext=AS5617#resultsAnchor

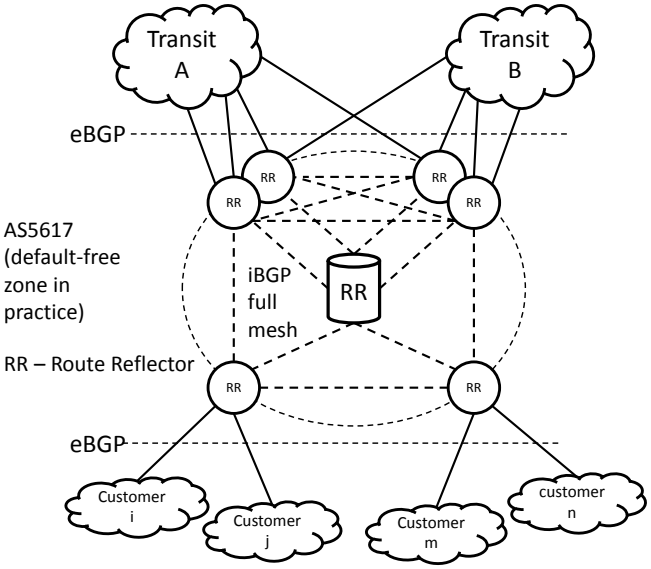


Figure 26 Schematic view of AS5617

The data set contains approx. 80 000 000 "raw" records (20 000 000 records per week on average). A sample record (line in a text file) is given below:

```
2015-09-07 10:41:30.151929,poz_r3,New Announce,103.252.84.0/24,AS Path:
5511 2914 17660 59219 (IGP) Local-Pref: 90   MED: 1 Communities: 5511:521
5511:666 5511:710 5511:5511 Next Hop: 194.204.128.131 Aggregator: AS59219
103.252.84.1 ,
```

As a matter of fact, the logs contained in the data set represent a time series describing and we will refer to it as "BGP time series"; the full log (containing all entries, non-filtered) will be referred to as "raw BGP time series".

9.1.2 Types of analyses

The results cover consecutive weeks of September 2015, covering:

- Time series modelling of the arrival process of BGP events affecting iBGP in AS5617.
- Statistics of the iBGP time series for the observed AS.

In the following, the results of the study are presented in a graphical form to cover the following statistics:

1. Total number of route updates as a function of the length of network suffix
2. Total number of route updates as a function of route stability time threshold
3. Number of active prefixes as a function of route stability time threshold
4. Total number of route updates as a function of route stability time threshold
5. Fraction of the global IP space occupied by active prefixes (the full size of IP address space is 2^{32})
6. Number of active prefixes as a function of the size of network suffix
7. Number of active destination AS-es as a function of the length of network suffix
8. Average path length change per route update as a function of the length of network suffix
9. Std deviation of path length change per update as a function of the length of network suffix

Note that the indices and titles of the statistics in the above list are the same as those used in the corresponding subsections of section 9.1.4. We will first construct a time series model that is used throughout the study for modelling iBGP update arrival processes. This is discussed in the next section.

9.1.3 Time Series Model

We convert the raw BGP update time series (with continuous time) into a discrete time equivalent with time slot duration equal to *s* seconds. In the discrete version of the time series, all updates appearing in a given time slot are attributed the same time of arrival. We select *s* to only be as large so that the resulting number of samples in the time series remains manageable for the analysis.

The basic model for update analysis is based on the observation of time series related to individual prefixes (destinations). The various model parameters are presented in Figure 27.

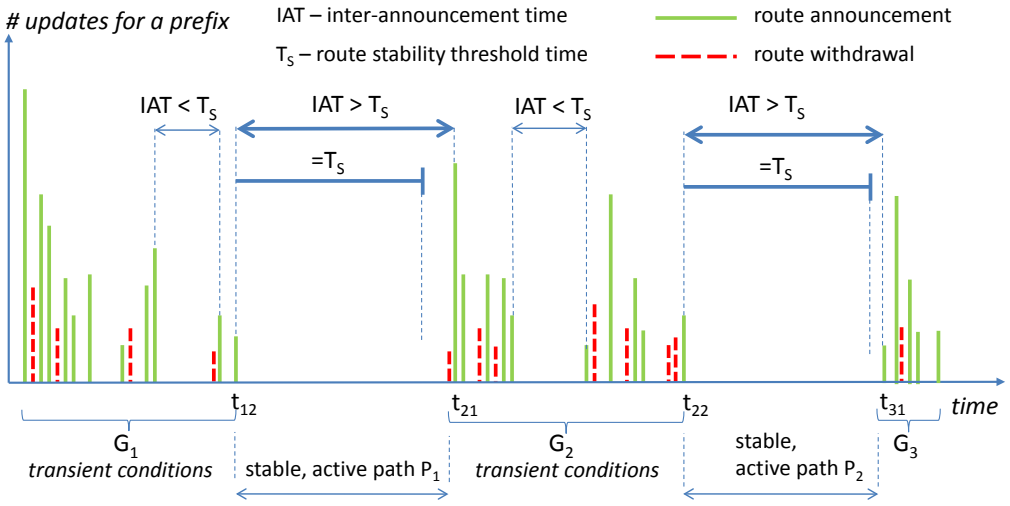


Figure 27 Time series for prefix-level updates for threshold T_G.

According to Figure 27, iBGP updates are distributed in time unevenly, and they arrive in groups (bunches) of different size.

Groups can be defined in various ways, e.g. based on the time between certain events. In our analysis, we are mostly interested in semi-stable conditions of the routing paths for individual destinations to get insight into how often it will be necessary/sufficient to update forwarding entries of FUSION resolvers. Referring to Figure 27, we define a group of updates for a given IP prefix to be a series of interleaved updates (announcements, withdrawals) that (a) starts with either an announcement or with a coupled pair of withdraw/announcement messages, (b) terminates with an announcement message, and (3) the group is separated from adjacent groups by time (termed inter-announcement time, IAT, see the figure) that exceeds a given duration T_S (in the remainder of this report we will refer to time parameter T_S as ***route stability time threshold***). In the example in Figure 27 there are three such groups of updates for a prefix.

Note: There are no announcements between two consecutive groups. A group often begins with a withdraw update message that typically is followed almost instantly by a series of coupled announcements. In general, group of updates is defined conditionally, i.e., for a fixed value of parameter T_S which allows to filter out frequent changes (occurring more frequently than defined by T_S).

A group of updates for a given prefix (and a given value of parameter T_S) is characterised by the fact that the path ultimately chosen to be active after the last announcement has been received will not change at least up to the time when the next group of updates starts. For example, t_{12} in the figure is the time instant when group G_1 terminates and the path that is active at this moment will remain valid (i.e., used by the AS for the given prefix) at least up to time t_{21}.

In an operational network, routes for a given destination can be very unstable during the life time of a group of updates (e.g., during the time from t_{21} to t_{22}). Such transient conditions typically do not last too long – up to 140 seconds for withdrawals and twice as less for announcements (according to [RIPELABS, https://labs.ripe.net/Members/vastur/the-shape-of-a-bgp-update][12]). Routing during transient phase can be inconsistent which can result in looping. Obviously, from the point of view of FUSION the most interesting case is when the time period when routing is stable (IAT in the figure) is much larger than instability time (t_{21} up to t_{22} in the figure).

There are two main parameters used throughout our analysis of iBGP statistics:

- **Route stability time threshold, T_S**. We note that by changing the value of parameter T_S we can "scan" the distribution of the minimal duration of periods with stable routes for a given prefix. This method is used in the majority of the analyses presented in the subsections that follow. In this study, we assume the following **values for parameter T_S (given in seconds): 5, 15, 30, 60, 120, 240, 480, 960, 1920, 3840, 7680, 43200**. Note that this series is approximately geometric with step 2 from 15 up to 7680 and then has a one-step increase up to 43200; this has some impact on the shape of plots in the figures.
- **Size of network (address) suffix**. This is simply the length of the variable part of the address that can be used within a given network, i.e., for network prefix given in the format *a.b.c.d/p*, the size of network suffix is *32-p*. In the figures, the network suffix length is always given; in the text we often refer to prefix length so attention should be paid to the terms used.

Throughout the analysis we also make use of the following terms:

- **Active prefix**: For a given observation period and a given value of parameter T_S, a prefix is said to be active if there has been at least two consecutive sequences of updates for this prefix encountered in the observation period that are separated by inter-arrival time (IAT) greater than T_S. Referring to Figure 27, assuming that G_1 is the first group observed in the observation period, the prefix becomes active at time t_{12} when subsequent group satisfying the condition IAT>T_S is detected (G_2 in this case), otherwise the prefix will remain inactive for a given value of parameter T_S.

Route update: we clearly differentiate between iBGP updates which refer to all iBGP messages disseminated within the AS and ***route updates***. *Route update* only refers to an *active* prefix (in the sense explained above) and takes place at group termination time t_{i2} (e.g., t_{12} and t_{22} in Figure 27). Note that the requirement for a prefix to become *active* when respective *route update* takes place means that at least two consecutive groups of updates are detected during the observation period for this

[12] We note that, depending on the frequency, a phenomenon known as *route flapping* may or may not be captured appropriately by our model. Nevertheless, for being a marginal and pathological behaviour, route flapping is not treated in a special way in our analysis.

prefix, otherwise no route updates will be counted for such a prefix. In the following, we often omit "route" from the term and use the term "update" instead.

9.1.4 Results

In the following subsection we present the results of the study and a detailed discussion. A summary of this discussion has been shifted to section 3.2.1.

9.1.4.1 Total number of route updates as a function of the length of the network suffix

We remind that throughout this analysis, route update events are always determined according to the definition of stability threshold T_S as explained in section 9.1.3.

A diagram of this metric obtained for the whole period of Sept. 2015 is given below. Note that the X axis has a logarithmic scale (in the linear form, it is similar to the previous one although features lower variability for each size of network address suffix).

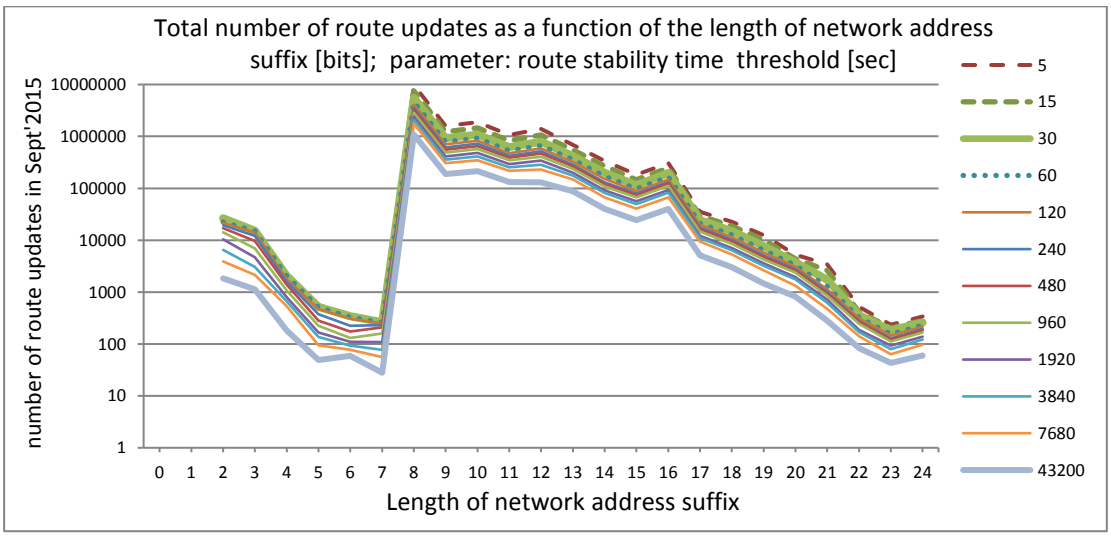


Figure 28 Total number of route updates as a function of the length of network suffix [bits] for different values of stability time threshold [sec] – averaged for whole Sept 2015.

As can be seen, the highest frequency of changes (regardless of the value of stability time threshold T_S during the observation period) are experienced by small networks with prefix (or mask) length equal to /24 (i.e., 8 bit size of network address suffix as shown on X axis in the figure). A general trend is that larger networks (shorter prefixes /p) experience much lower update frequencies, i.e., have much more stable routing paths that smaller networks. Moderate deviations from this rule can be observed for network suffix length 10, 12, 16 and for suffix length 24 (p=8). There are also some deviations for the values of network suffix size smaller than 7 (p>25).

Note for example, that for subnetworks with address prefix length equal to 16 there were about 40000 route updates with route stability period not shorter that 43200 sec. (12 hours) and about 132000 updates for the period not shorter than 240 sec. Respective average rate of updates were one per 65 sec. and one per 20 sec. For the most active prefixes (prefix length 24, and suffix length 8 in the figure, i.e., small networks) these numbers translate into one update per 2.6 sec and one update per 0.75 sec. These number set upper limits on the speed of optimisation algorithms for different reaction time-scales; in reality algorithmic requirements will be much less stringent as only changes of routes for subnetworks with interesting data centres will have to be included into optimisation.

An interesting question is whether networks as small as up to 256 addresses can be useful for locations of FUSION execution zones. It seems that this level of granularity may not be interesting for the majority of potential applications of FUSION. However, the analysis in section 9.1.4.3 demonstrates

that BGP updates visible in the considered AS affect a significant fraction of the global Internet address space. Therefore we take 8-bit network suffixes into consideration. More insight into the practical importance of this address range can be found by analysing the path length variation. The results on this topic are presented in sections 9.1.4.6 and 9.1.4.7.

9.1.4.2 Total number of route updates as a function of route stability threshold

A diagram for this metric obtained for whole Sept. 2015 is given below.

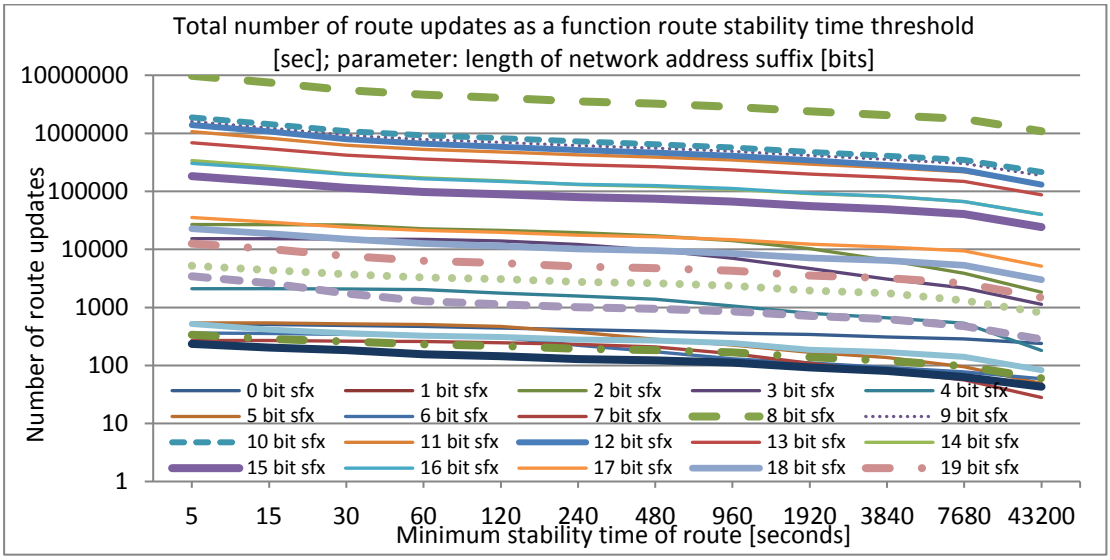


Figure 29 Total number of route updates as a function route stability time threshold; parameter: length of network suffix.

In this figure, *Serie X bit sfx* corresponds to the length of network address suffix equal to X (i.e., prefix /(32-X). As seen from the figure, prefixes /24 (networks with suffix length 8) generate the highest number of route updates for all inspected values of stability threshold T_S. This drawing is in fact a projection of results from section 9.1.4.1.

9.1.4.3 Total number of route updates as a function of route stability time threshold

A diagram for this metric obtained for the whole Sept. 2015 is given below.

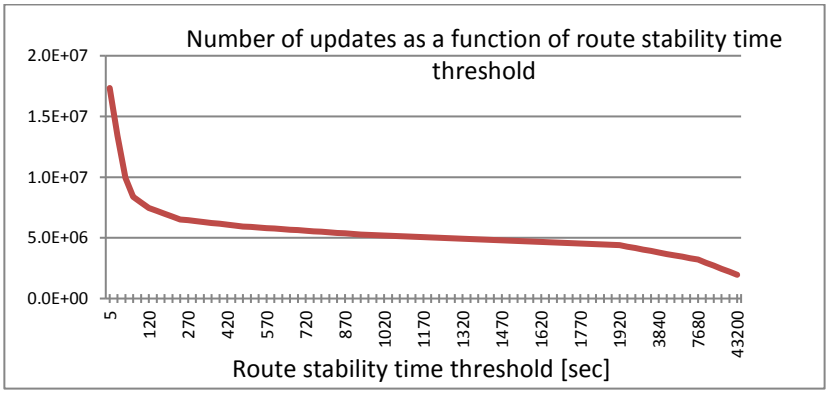


Figure 30 Number of route updates as a function of route stability time threshold.

The distribution has been collected for the whole observation period and over all active prefixes. Note that the chart for the number of updates is piece-wise linear: the actual calculations have been obtained for the values of stability threshold T_S defined in section 9.1.3 and missing values have been obtained through linear interpolation. Note also that x-axis is not linear.

As seen from the figure, the majority of updates take place on relatively short time-scale. As we commented previously, for scalable optimisation algorithms a limited frequency of changes at the network layer is preferable. For example, one can see that setting the stability threshold T_S equal to 240 sec. filters out (i.e., hides from the optimisation) about 70% of total route updates so only up to 30% of updates would have to be processed by the routines updating the service resolution tables of the resolver. We note such a threshold determines the time scale for reaction of resolution to the changes of routing paths.

A proposition similar to that from section 9.1.4.1 is that the analysis of the degree of path length instability can be helpful to asses is including BGP information into optimisations of service resolution can be beneficial. Initial results on this topic are presented in sections 9.1.4.6 and 9.1.4.7.

9.1.4.4 Fraction of the global IP space occupied by active prefixes

A diagram for this metric obtained for the whole Sept. 2015 is given below.

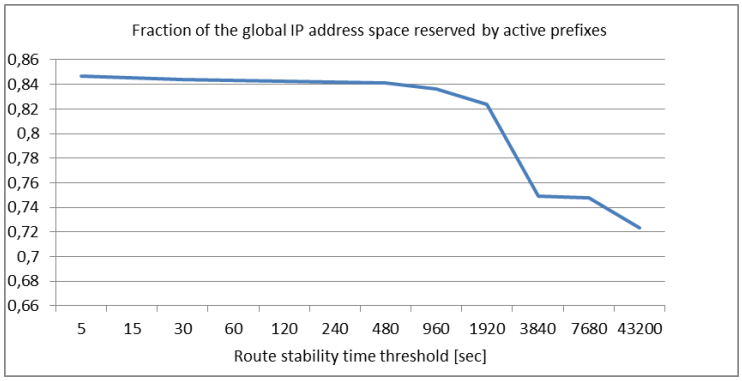


Figure 31 Fraction of the global IP space occupied by active prefixes.

This was obtained by observing all prefixes that are active (experience at least one activity period) during the measurement period and summing up the size of IP address space (e.g., $2^{(32-p)}$) for these prefixes (recall that we say a prefix is active if it experiences at least one cycle of stability time threshold).

A very slow decay of the graph suggests that majority of prefixes experience activity periods with a broad range of values (during the observation period of one month). This means that majority of prefixes has potential for orchestration/resolution optimisations. As stated previously, actual benefits will strongly depend on actual variability of paths (path delays/losses).

The sharp drop of the curve for the largest values of stability threshold T_S may result from finite observation period (one month).

We conclude noting that route update churn covers a significant portion of users.

9.1.4.5 Number of active destination ASes as a function of the length of the network suffix

A diagram for this metric obtained for the whole Sept. 2015 given below.

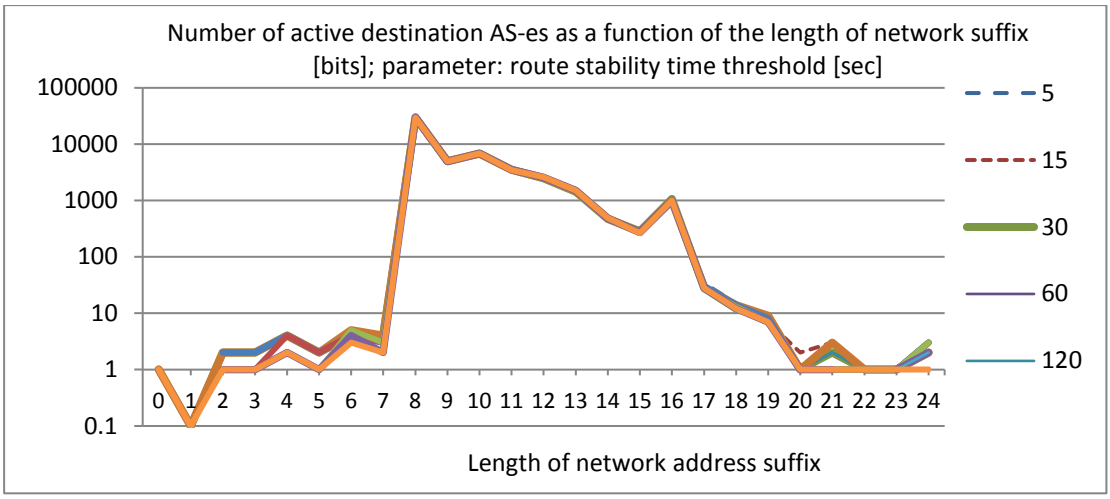


Figure 32 Number of active destination AS-es as a function of the length of network suffix; parameter: route stability time threshold.

This diagram has been prepared as a check for the conclusions formulated in sections 9.1.4.1 and 9.1.4.3.

9.1.4.6 Average path length change per route as a function of the length of the network suffix

A diagram for this metric obtained for the whole Sept. 2015 is given below.

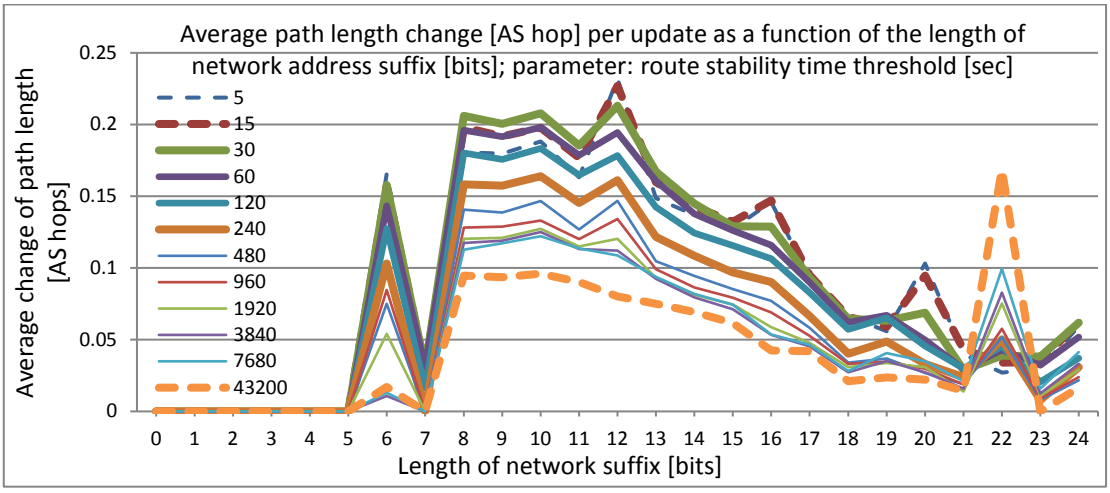


Figure 33 Average path length change [AS hop] per update as a function of the length of network suffix; parameter: route stability time threshold.

Path length change is defined as the absolute value of the difference between the length of path (given as the number of AS hops along the path) before and after update of a route (thus, for two pairs [1,3] and [13,11], each representing two path lengths, the difference is the same and equals 2).

The figure shows that smaller networks (longer prefixes) tend to experience larger changes of path length resulting from routing updates than larger network. Nonetheless, even large networks (with adress size > 18, i.e., prefix < 14) can experience non-zero changes. These observations are not new.

However, the key result for our analysis from the figure is that **the average path length change is very small for all prefix lengths**. Indeed, it can be seen that the average value of this parameter is contained in the rage (0.05, 0.22) depending on prefix length. This indicates clearly that alternative routes for a

given source-destination pair (in different ASes) typically do not vary in length. **In consequence, it will be hard to infer changes of inter-domain delay knowing only the length of current inter-domain path.** In the next subsection standard deviation of path length change is analysed to get deeper understanding of the variability of inter-domain path length. It will facilitate drawing the final conclusions for this section.

9.1.4.7 Standard deviation of path length change per route update as a function of the length of network suffix

A diagram for this metric obtained for the whole September 2015 is given below.

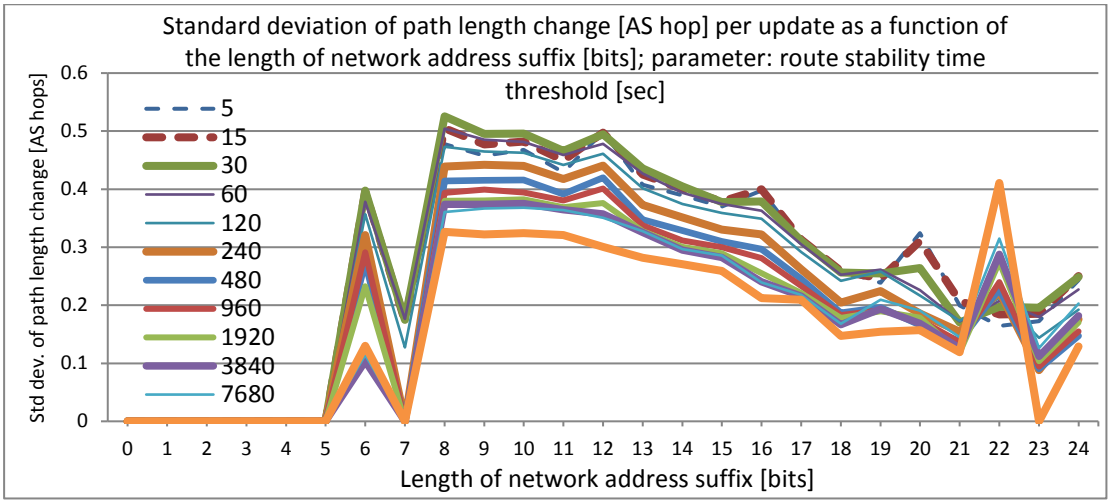


Figure 34 Standard deviation of path length change [AS hop] per update as a function of the length of network address suffix; parameter: route stability time threshold.

This figure is complementary to Figure 33 and shows the standard deviation of path length change due to update as a funtion of network address length.

It can be seen that the shape of plots in the figure resembles that of plots in Figure 33, and the value of standard deviation is pretty stable for the whole range of remaining parameters (length of network suffix, route stability time). This confirms our previous suggestion that **the knowledge of the length of inter-domain paths alone is not sufficient for service resolution based on inter-domain delay**. We thus conclude that if service resolution has to be based on the delay criterion then the knowledge of inter-domain path information should be enhanced with direct measurements of delay.

Actually, only sufficient degree of variability of end-to-end delays (confirmed by direct measurments), combined with the existence of correlation between delay changes and BGP events, can justify the inclusion of BGP information into service resolution. These topics are discussed in sections 9.2 and 9.3 of this annex.

9.2 Variability of inter-domain delay

The goal of the analysis in this section is to identify the main types or “patterns” that can be observed in the behavior of e2e latency towards different datacenters and understand possible causes for this behavior. Our aim is to see the potential of ISPs offering a FUSION resolution service. In the following, different aspects of e2e delay behavior are discussed in separate sections. Owing to the measurement methodologies used, the e2e delay is meant as respective RTT.

The measurements were taken over the period from 8th January to 30th March 2016 using the tools from Cloudharmony as was explained in section 3.1. In this analysis, we use traces obtained from two

probes connected to two different ISPs[13]. If necessary, we tag related figures by inserting the acronym of respective ISP (ISP1 or ISP2) in the header of the figure.

9.2.1 Overall characterization of delay variability

Figure 35 provides overall characterization of RTT using a subset of DCs spread across Europe and one located in the US. It demonstrates that there can be relatively long periods of quite stable RTT for many DCs owned by different providers. However, periods with higher delay variability also occur, as shown on the left-hand side of the figure.

The left-hand part of the plots from Figure 35, zoomed-in and with some DCs having been filtered out (e.g. citycloud/sweden-x and digitalocean/ams2), is shown in Figure 36. It shows that even during the periods with higher or more variable delay typically there are DCs that offer sustainable performance. Of course, the details of the behavior can depend on the home AS of the client (which will be commented on later) as well as the DC provider or DC instance, nevertheless, a general conclusion that can be drawn from the analysis is as follows:

- Most of the DCs analyzed offer sustainable RTT over quite long periods of time, but for many of them there are periods when the performance increases or is highly variable at different time scales (sometimes on the order of hours, sometimes several minutes[14]).
- Typically, even if some DCs experience increased RTT in a given point in time, there are sufficient alternatives provided that they can be efficiently used by the application.

It can be seen that even within the geographical scope of Europe the differences in delay from a given location to particular DCs can be quite significant – it can be seen in Figure 35 by comparing, e.g., aruba/DC5-DE and digitalocean/ams2 to citycloud/sweden-north. Moreover, insight into the delays for other DCs located in the USA (cloudprovider/las-vegas in Figure 35, and some other not presented here) prove that some "distant" American DCs perform similarly to the relatively "close" European ones. Therefore another general conclusion follows:

- RTT based on geographical coordinates can be very imprecise in many cases.

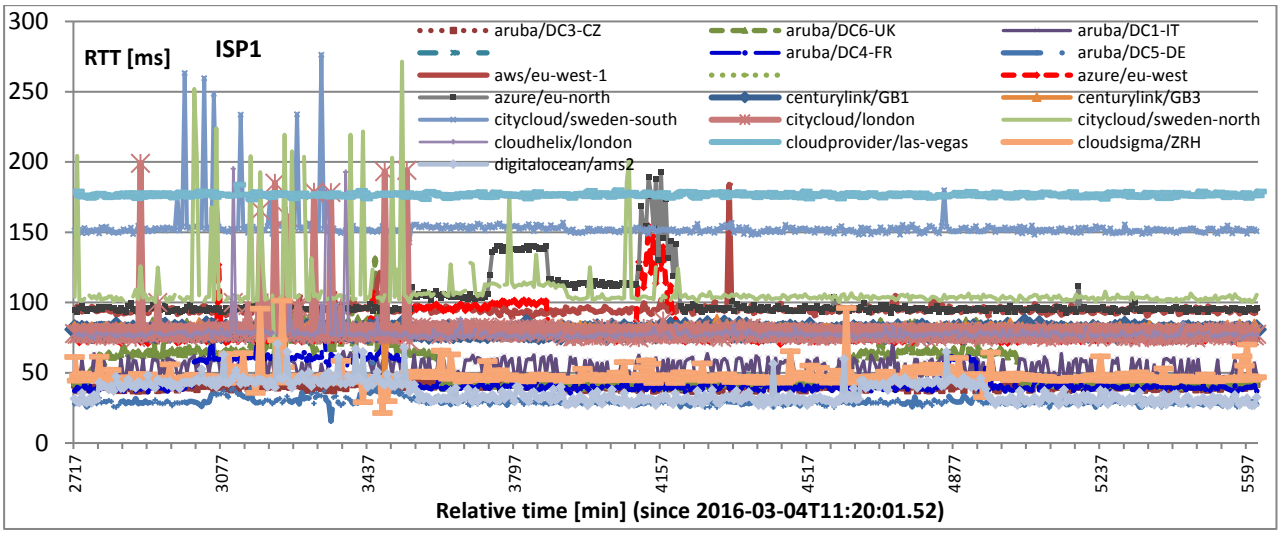


Figure 35 Periods of stable and variable e2e delay (subset of DCs in Europe).

[13] A short experiment was also done for LTE access as explained later in this section.

[14] We note that the finest granularity of the results presented here is 6 minutes. However, we note also that for Cloudharmony probes, the variance of RTT within a sample of 12 consecutive measurements typically is very small even if the variance of average RTT for consecutive samples is high in a given period of time.

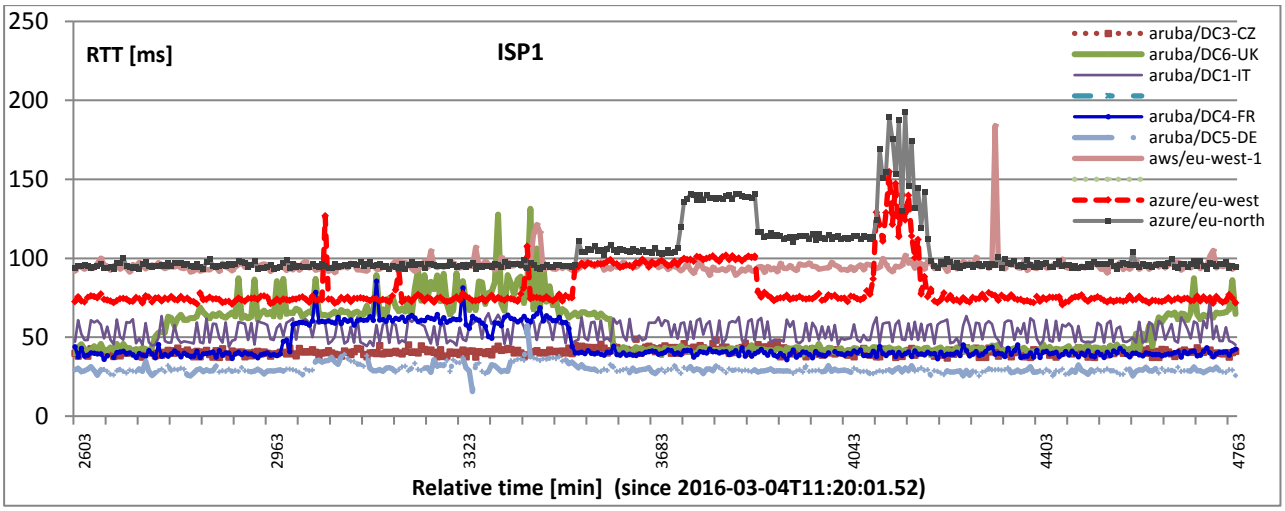


Figure 36 Periods of variable e2e delay – example structure (subset of DCs in Europe).

In addition to the above, in Figure 36, one can also notice three different types of RTT variability, represented by azure/eu-north (also azure/eu-west), aruba/DC6-UK (also aruba/DC4-FR) and aruba/DC1-IT. We discuss them in more detail in section 9.2.2. The impact of the client-hosting AS on the RTT observed will be discussed in section 9.2.3.

9.2.2 Policy-based variations

There are several forms of delay variations resulting from policy-based traffic handling. Different entities can imply such policies and often they can be identified only indirectly (and without 100% guarantee) through correlating traces for multiple datacenters. As will be shown in section 9.2.3 using traces from different AS can help identify probable entities, but no guarantee can be given.

In Figure 37 periodic diurnal variations can be seen for vultr/frankfurt in which case a shift of RTT from around 50 ms to around 80 ms can be observed. The occurrence of highly variable component of the histogram is closely correlated in time with the stable increase of the "lower bound" of RTT. The existence of a consistent "lower bound" of the RTT curve suggests the existence of a deterministic delay component along the path (including protocol stack in the endpoints). Our hypothesis is that the values at the level of 50 ms reflect mainly (almost constant) packet forwarding delays in the routers along the path in the network (including the network internal to the DC). The deterministic increase of the delay to the level of around 80 ms is most probably caused by allowing additional traffic on certain resources (networking or computational) to some predefined level and throttling excessive traffic in some way. The latter can be inferred by analyzing the progressive (step-wise) increase of RTT samples (hardly seen in the figure). Visible spikes can result from congestion caused by a non-perfect throttling mechanism or some other processes.

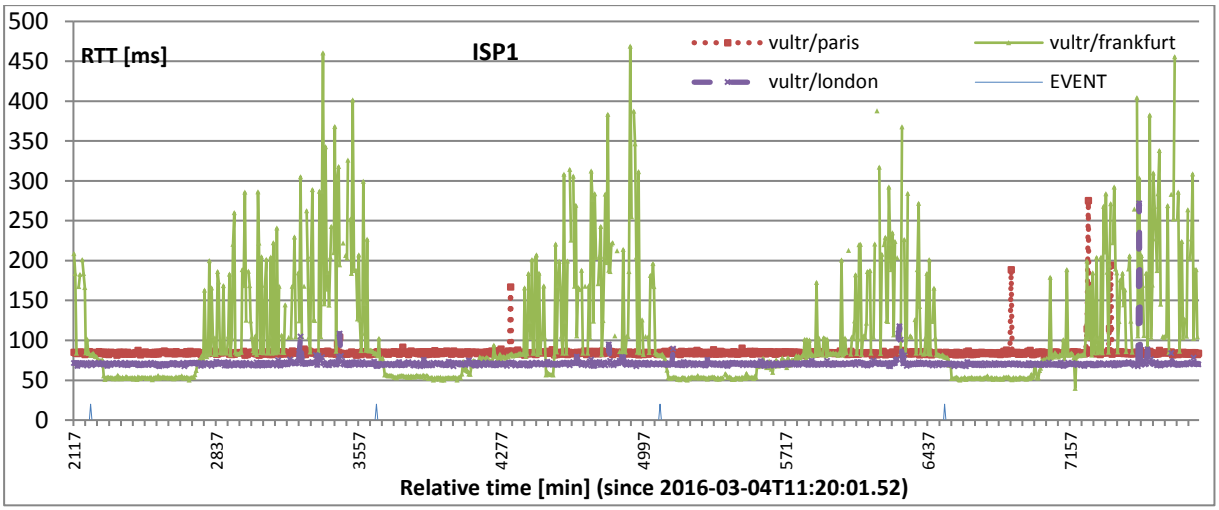


Figure 37 Periodic (diurnal) RTT changes.

Similar behavior to the one described in Figure 37 can be found in Figure 38. Here, DCs from different cloud providers (spread geographically and topologically) feature similar "spiky" periods, although the policies adopted at each of these point can differ (octave/pl_warsaw seem to have no particular policy defined for high load periods while lunacloud/eu-west and linode/london do have).

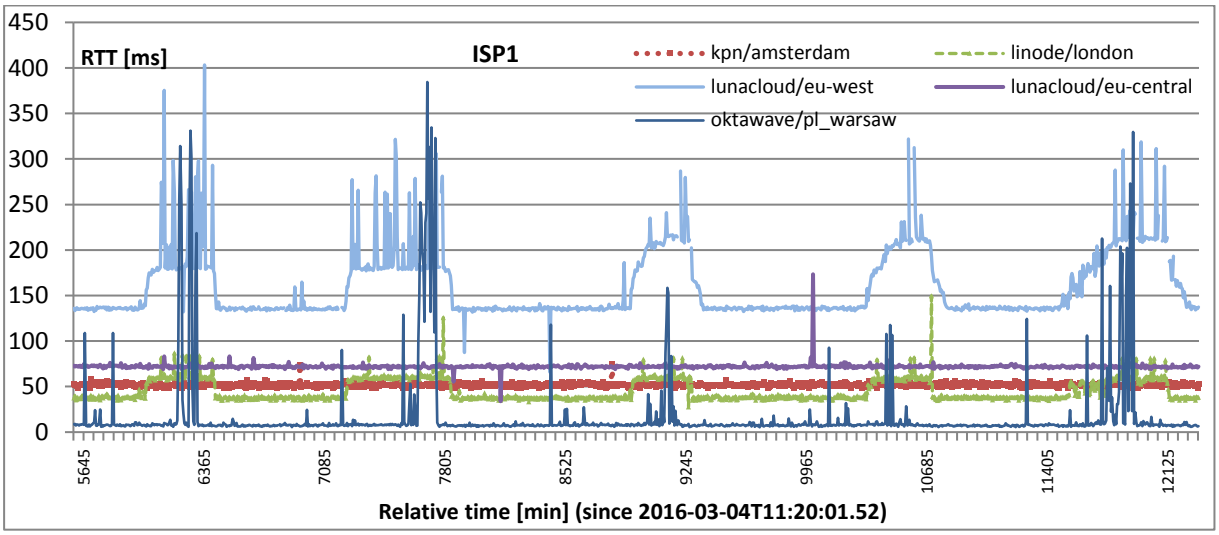


Figure 38 Correlation among different datacentres.

A number of different types of RTT behavior is shown in figures from Figure 39**Error! Reference source not found.** to Figure 43. In Figure 39, the data center digitalocean/ams3 (red plot) features activation of a policy (at time epoch 7961) that results in a double-level RTT trace (the difference between the levels is about 10 ms). This effect is similar to the one when multiple paths are used. Nevertheless, it is hard to determine the true cause and the part of the chain (network/data center/server(s)) responsible for this kind of RTT behavior.

In Figure 40, the double-level RTT pattern (similar to the one seen in Figure 39) continues over the whole observation period with one visible step increase of RTT. This increase can result from a change of routing (the jump is very steep which fits well with a routing change), but this is only a speculation as we have not been able to find any routing event correlated with this change[15].

[15] Monitoring BGP, we only observe events that can impact forward traffic. If routing changes relate to backward traffic, we do not observe respective events.

Another example of activating a new policy is shown in Figure 41**Error! Reference source not found.**. In this case, low/high RTT diurnal pattern arises for linode/london DC (similar to the one visible for this DC in Figure 39). Interestingly, the spikes in the high-load periods seem to create a "layered" pattern.

Such "layered" pattern is shown in more detail for DC softlayer/FRA in Figure 42. Periodicity of this pattern confirms our conjecture about the existence of respective policy along the full chain of functions involved (network, data center, protocol stack at server). However, the reason why the values of RTT during the high-load periods take (almost) discrete values (with a step of approx. 22 ms in the considered case) is unknown. We have checked for the existence of this pattern in ISP2 and found similar cases, too – see an example in Figure 43. A tiny difference between both of these cases is that the discretisation of RTT spikes is less profound in Figure 43.

The main conclusions are as follows:

1. It is hard to precisely explain the causes of the behavior discussed in this section.
2. The changes discussed do not seem to be correlated with notifications from the inter-domain routing (which we analyzed in addition to RTT). Therefore one should accept that this kind of variability has to be measured using methods based on active probing or client reporting.
3. Potential added value of ISP in this case is the ability to support probe measurements by offering a sufficient set of representative measurement points in its domain. Adopting a more aggressive approach, the ISP can also carry out such measurements on its own and even offer service resolution. In fact, the ability of ISP to provide the smallest possible set of representative (good) measurement points can be an advantage once the ISP gets involved in the provision of FUSION service resolution.

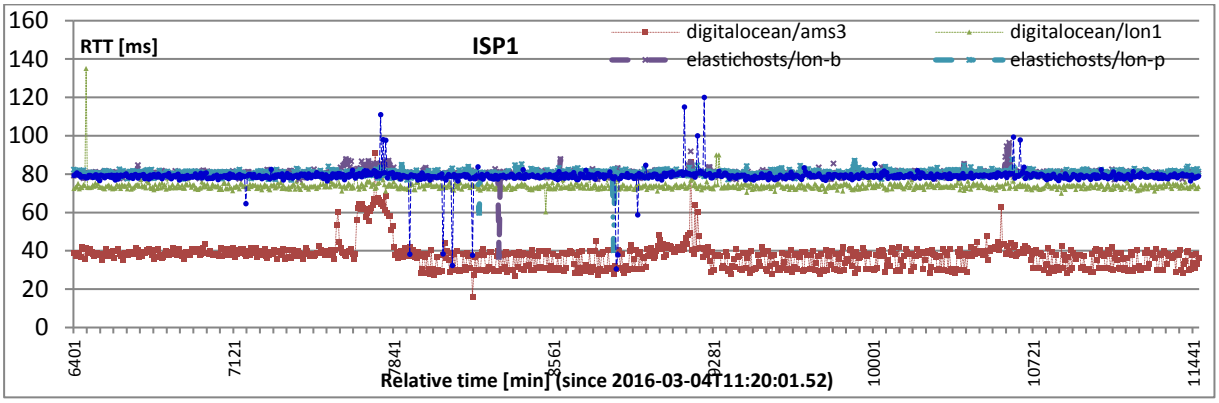


Figure 39 Activating a policy – RTT "discretisation".

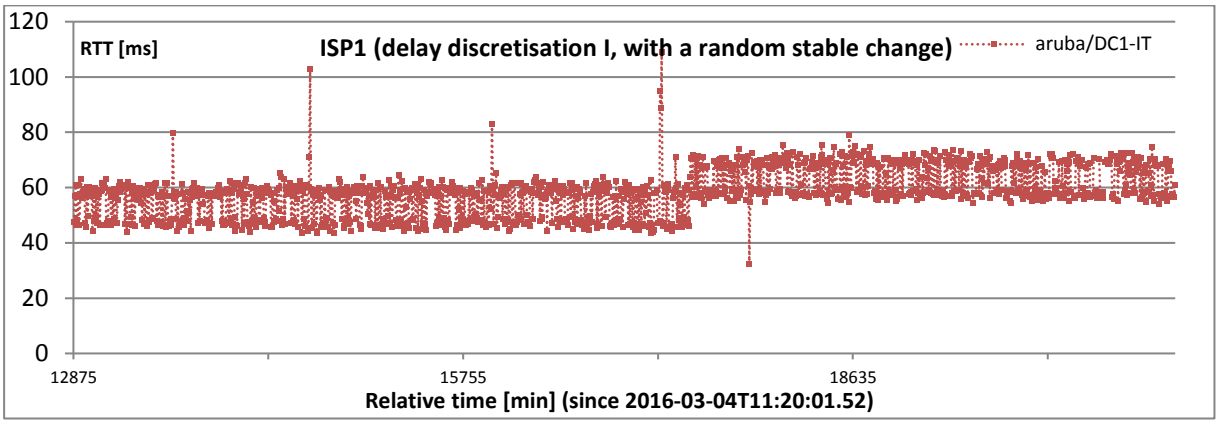


Figure 40 RTT "discretisation" (with a longer-term stable change).

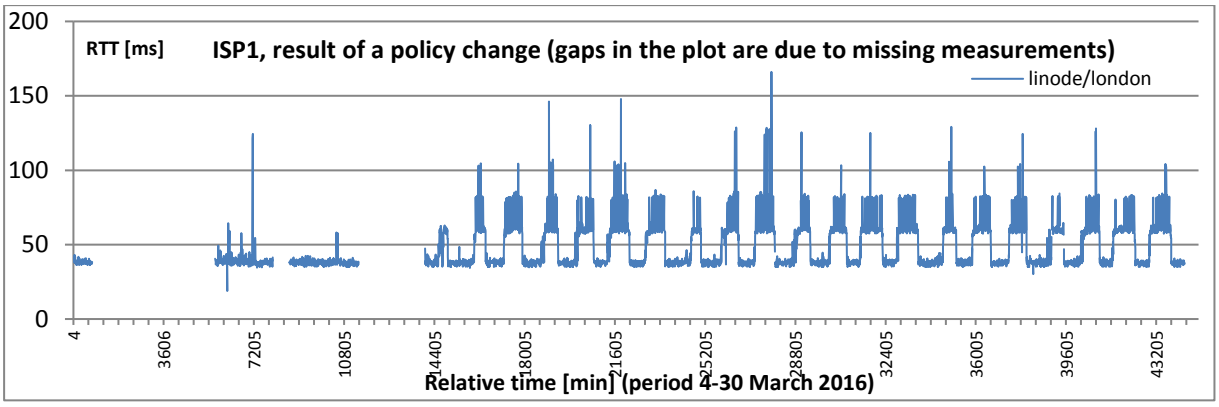


Figure 41 Activating a policy – diurnal RTT variations.

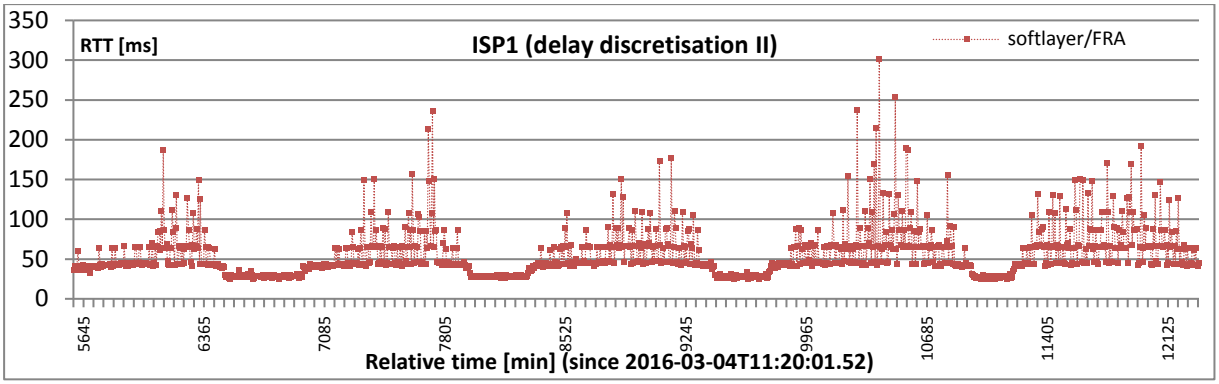


Figure 42 Discretised multi-level diurnal RTT changes.

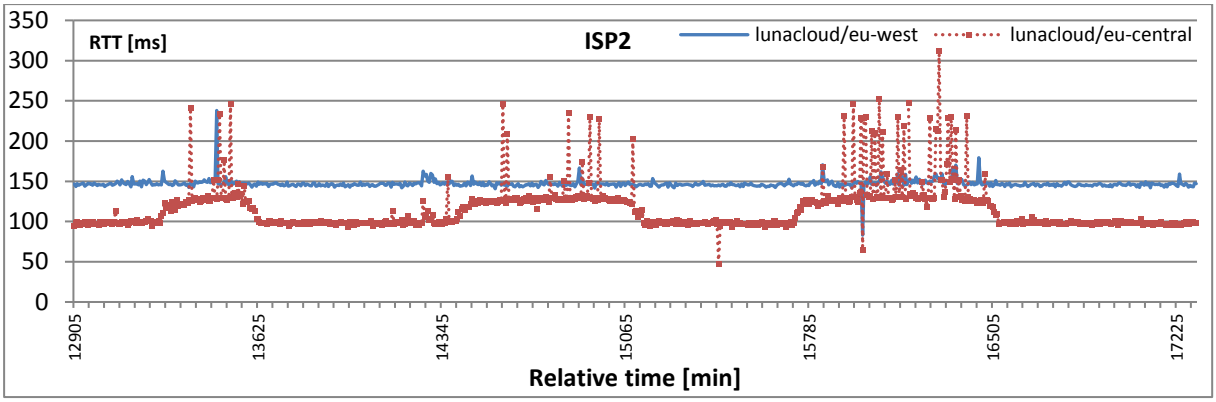


Figure 43 Diurnal RTT changes.

9.2.3 Impact of home AS of the client

In general, RTTs experienced by users of different ISPs seem to share similar types of behavior. To see this we present selected examples in figures Figure 44and Figure 45 for ISP2 to be compared to remaining figure in this section that relate to IP1.

Nevertheless, it becomes obvious that the characteristics of RTT for a given data center obtained from similar geographical location can differ significantly depending on the actual ISP of the client. To see this, in figures Figure 46 to Figure 50 we present corresponding examples for two selected DCs and respective RTT measured from ISP1 and ISP2. On the other hand, a very flat RTT characteristic observed from ISP1 towards lunacloud/eu-central can be seen in Figure 38 while significant variability of delay for this datacenter from ISP2 can be seen in Figure 43. Additionally, in Figure 50 we plot the evolution of RTT during a fixed time period for a subset of data centers and for a fixed access and LTE. The

comparison confirms that using different access providers can lead to significantly different RTT experience of users.

The main conclusion follows:

- Evaluation of effective delays between clients and service instances based only on geographical coordinates can produce significant errors. Topological information regarding at least hosting networks should additionally be taken into account for critical services.

Figures

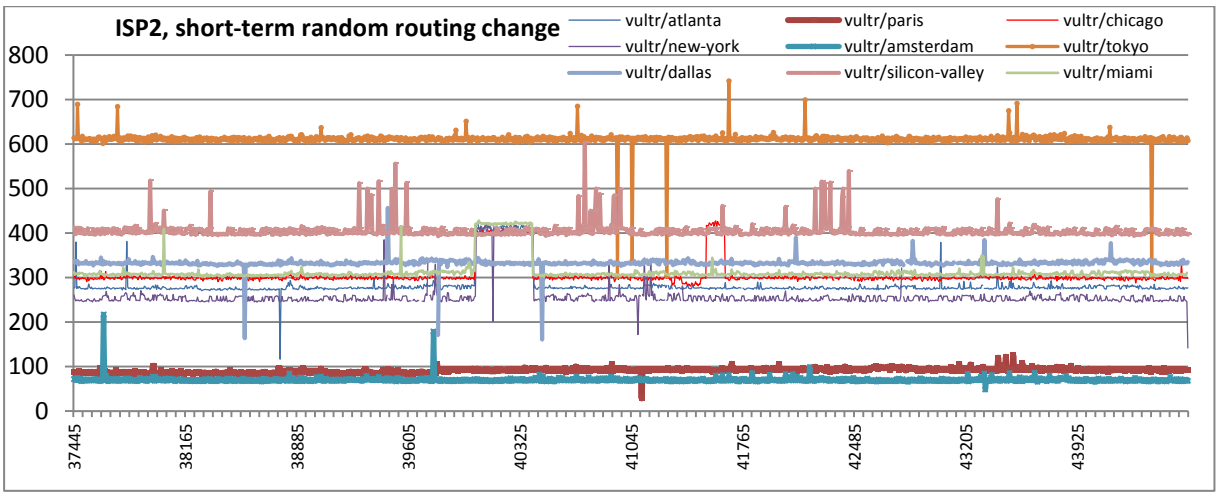


Figure 44 RTT from ISP2 – general picture (example 1).

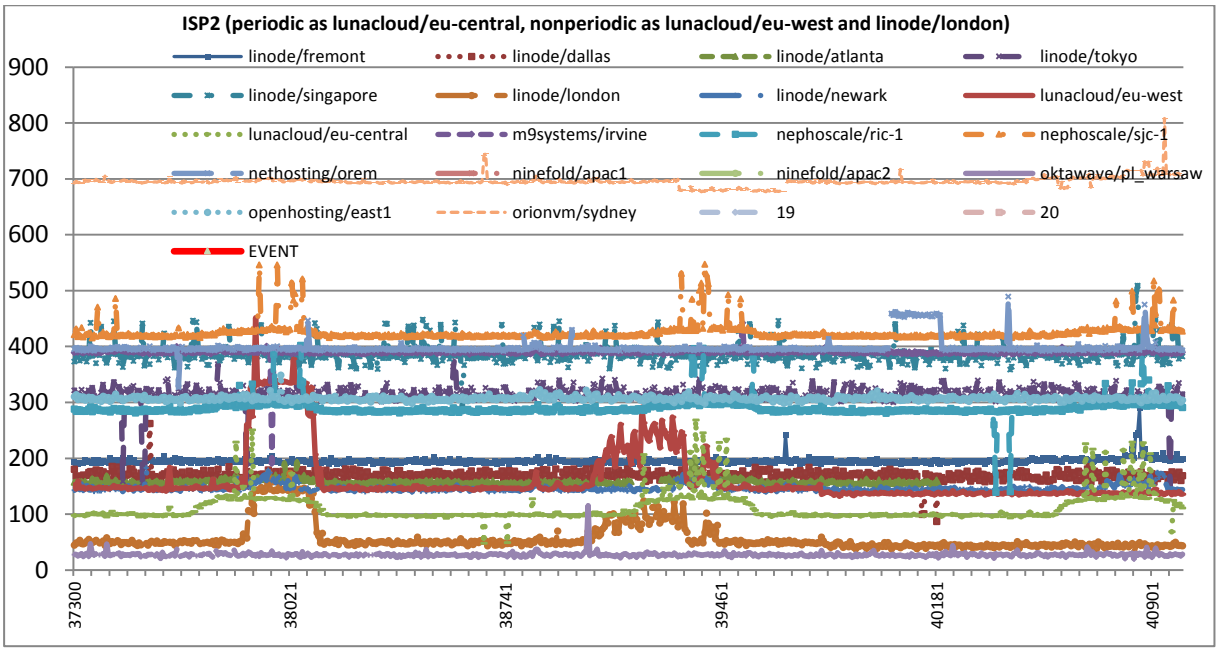


Figure 45 RTT from ISP2 – general picture (example 2).

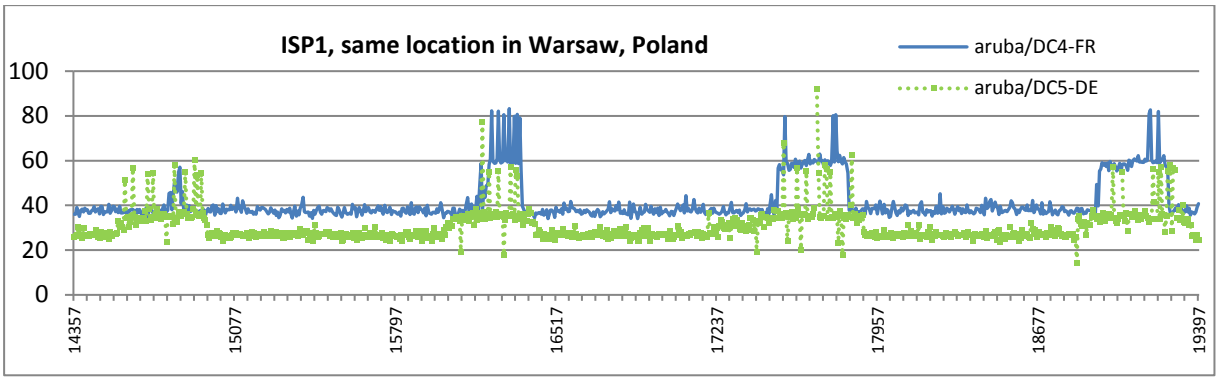


Figure 46 Variable RTT seen from ISP1 (example 1)

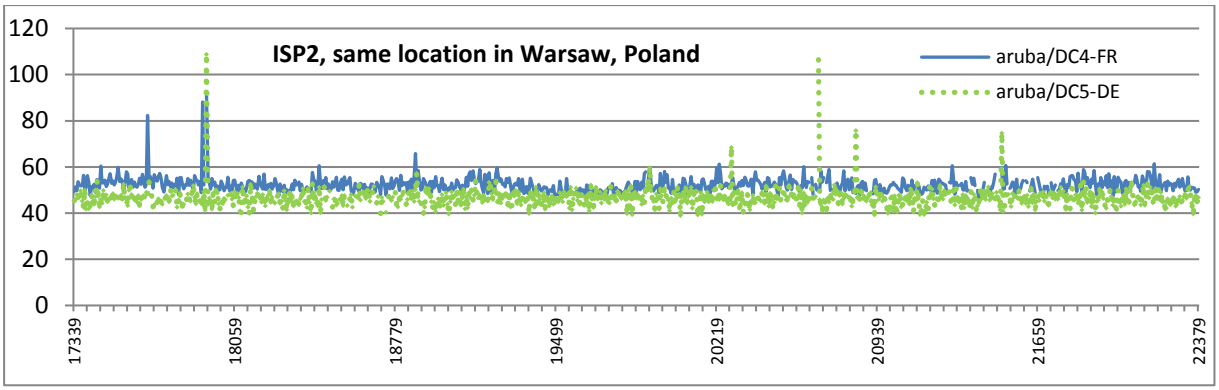


Figure 47 Stable RTT seen from ISP2 (example 1)

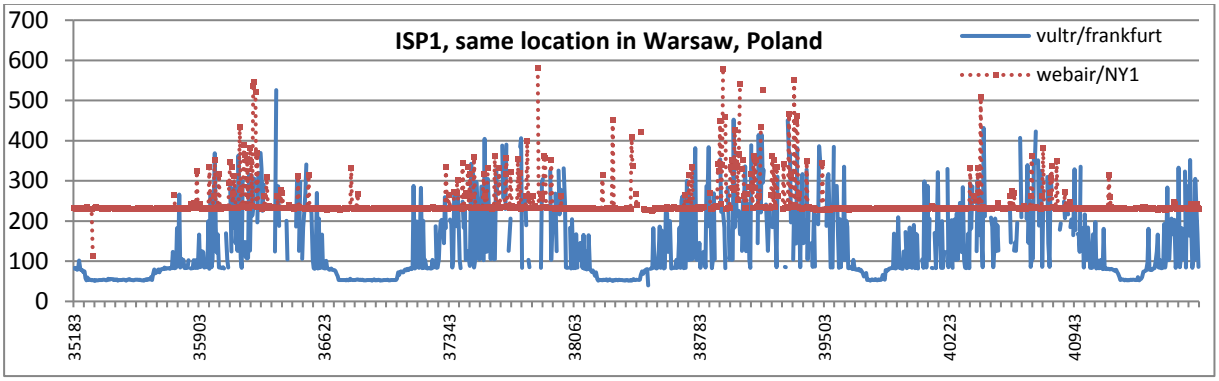


Figure 48 Variable RTT seen from ISP1 (example 2)

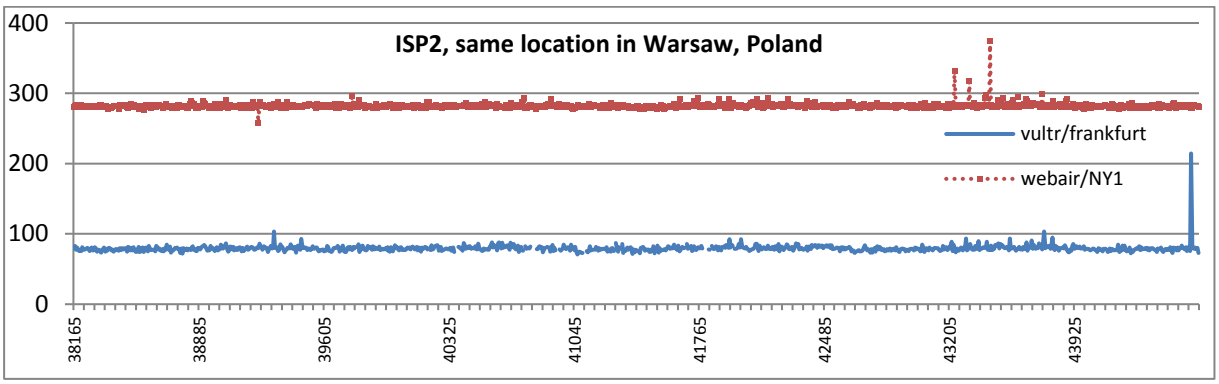
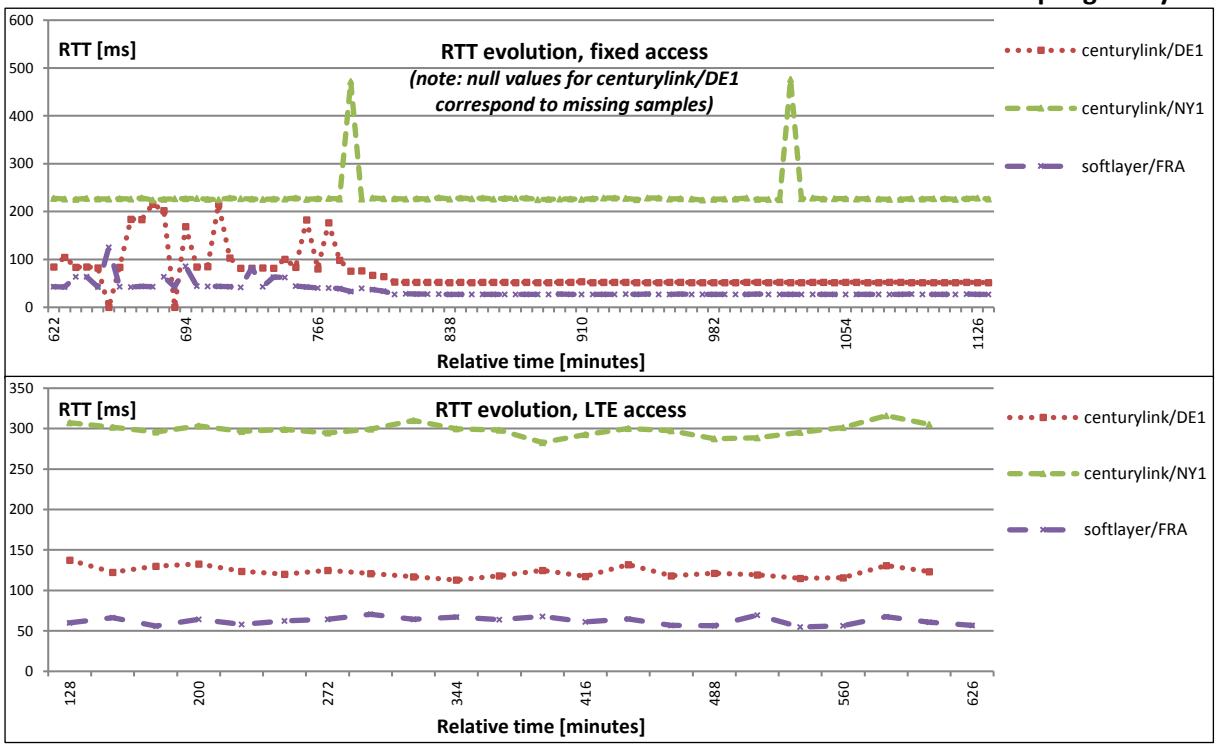


Figure 49 Stable RTT seen from ISP2 (example 2)

Figure 50 RTT for a fixed and LTE access compared (for LTE, the sampling period was set to 24 minutes so the resolution of the traces is lower than for the fixed access where sampling every 6 minutes was used).



9.2.4 Impact of routing changes

Analysis of identified cases of a change in network routing shows that the changes of RTT resulting from changes of inter-domain routing are quite fast. Of course, in a very short time scale (several-to-dozens of seconds) routing can experience instabilities and so does RTT, but if measurements are taken with frequency of several minutes (6 minutes in our case) the slopes of RTT curves are very steep, exposing very few intermediate points during the transition. In the following we show three typical examples of such changes from among many more identified during our study. In principle, they differ regarding the duration of the change – from short-time changes as in Figure 51, through mid-term and long-term changes as in figures Figure 52 and Figure 53, respectively. Similar (changes are recorded in Figure 36 for azure/eu-west and azure/eu-north.

The examples shown above share in common that the epochs of the changes due to routing events are random (at least from the perspective of the AS where they were registered). This means that counteracting to such changes for the purposes of service resolution in FUSION can only be reactive. Clearly, one possibility is to rely on active measurements using probes or client reports. The advantage of this approach is that it operates using directly the quantities of interest. With this option, however, the responsiveness of the resolution system will be limited by the maximum frequency of measurements that can be used in practice. Alternatively, monitoring BGP routing can lead to faster detection of possible causes, but additional logic is then needed to map detected events into the decisions about resulting delays[16]. Thus, potentially faster reaction based on analyzing BGP routing information requires additional complexity on the algorithmic side of the resolver.

[16] Note that a change in routing does not bear explicit information about resulting change in transmission time. The latter has to be inferred, typically based on additional topological information and/or historical knowledge.

A more detailed analysis of the dependence of RTT on the changes of inter-domain routing is provided in section 9.3.

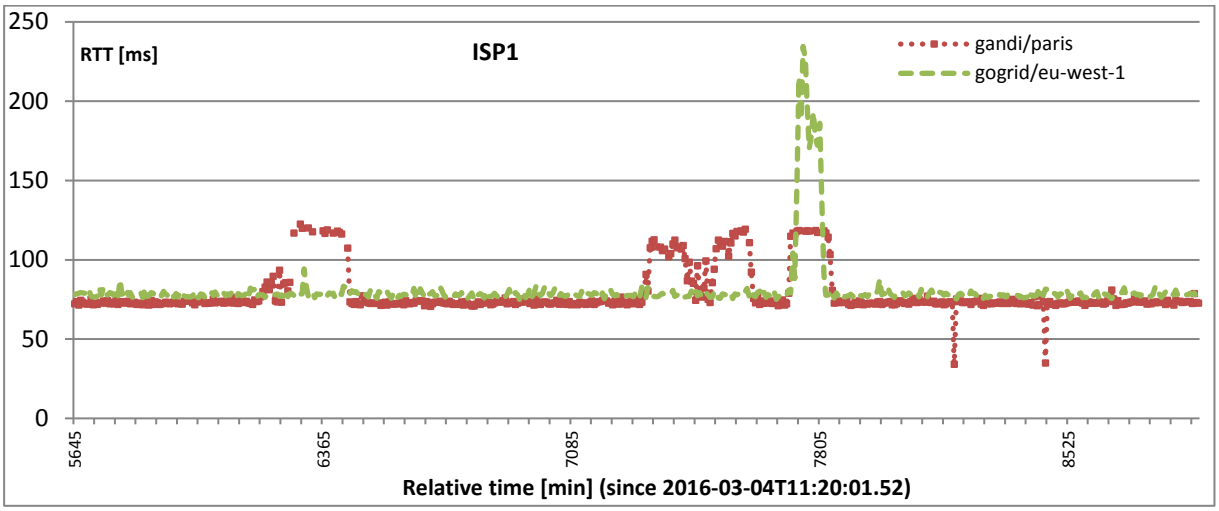


Figure 51 Random, short-term (order of several hours) routing changes, partially correlated for different DC providers.

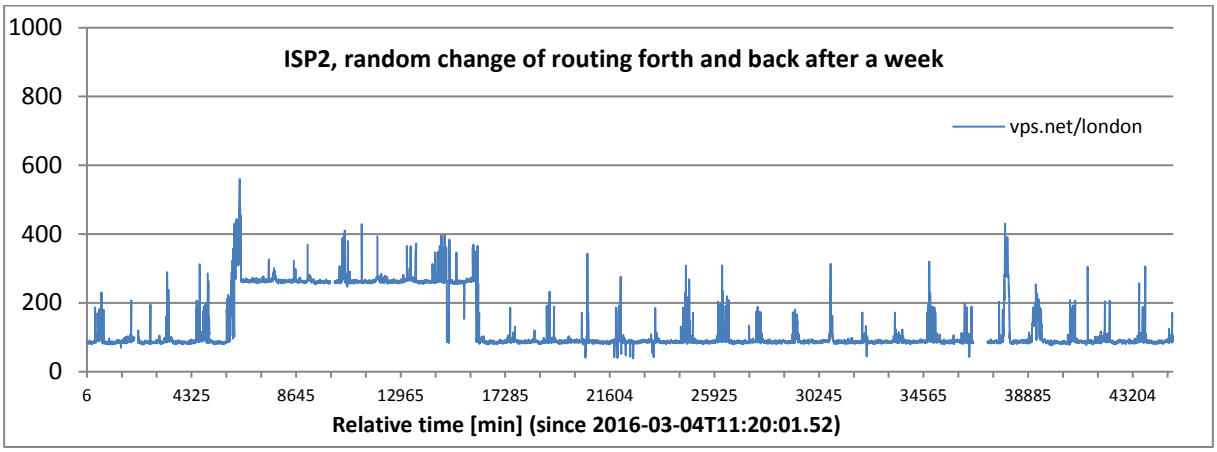


Figure 52 Short to mid-term routing change (order of a week)

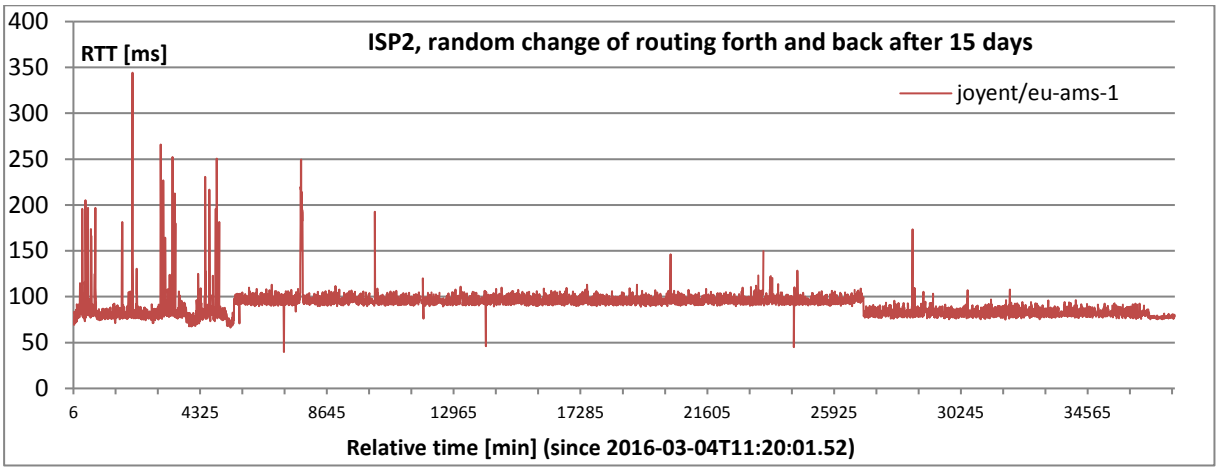


Figure 53 Mid-term (order of several days) routing change

9.3 Impact of inter-domain routing variations on delay

9.3.1 Discussion and conclusions

The last experiment for inter-domain routing was to assess the impact of changes of inter-domain routing on end-to-end delay. To achieve this we aligned in time several BGP routing events with the histogram of RTT for a set of data centres. BGP events for analysis were chosen based on the number of BGP notifications generated during the event.

We analysed selected events from among those that occurred in the period from Jan. 8th to March 30th 2016 and generated large number of notifications. We focused on relatively large events as they are easy to detect. Otherwise very detailed and time-consuming analysis would have been needed to identify interesting events and correlate them with appropriate destinations covered by our measurement probe. Notably, we were able to identify the root cause (a direct transit link to a Tier-1 network or distant AS) for most of those events.

Every figure in this section contains RTT plots for all destinations that were visibly impacted by the event (or events) covered in the figure. In some cases also other destinations (i.e., not affected visibly) are included for sake of comparison (e.g., linode/london in Figure 55). We caution, however, that with the sampling period equal to 6 minutes we can lose relevant part of the information for some destinations that actually may have been affected.

The approximate time of occurrence of each event was determined as the time when the number of BGP notifications registered during some short time interval exceeded an appropriate threshold. In each figure, event occurrence time (CET) is explicitly listed in the header of the figure and is marked on X axis for sake of readability, while the values describing time on X axis are relative and relate to the event. For any figure, plots annotated in the legend with integers do not matter.

Our main observations regarding each event are given in the description of respective figure. Each figure contains all the information of interest. The main conclusions based on the analysis of the figures are given in the following list.

1. For any major event observed (easily detectable as we explained above) the number of destinations explicitly affected did not exceed 10% of all monitored destinations. For most of the observable cases, the impact takes the form of a loss of connectivity (sometimes a drop of connectivity). Such a change of RTT is transient and takes place when routing converges within the network. In reality, the number of affected destinations can be higher, but they remain hidden due to the limited resolution of the measurement method of RTT.
2. When routing converges, RTT typically stabilises on the level from before the event (despite the AS path changes). Noticeably, in event-free periods, RTT is often pretty stable (in those periods RTT results mainly from forwarding delays in routers and in the transmission network). Considering these facts we conclude that many alternative paths for the inter-domain traffic are similar in length to primary ones.
3. Up to 10% of visibly affected destinations experience persistent changes of RTT in consequence of routing updates due to statistical major event. This translates into not more than 1% of all monitored destinations.
4. Even less destinations (say, not more than 0.5% of all destinations) experience major persistent changes or RTT (say, above 30% of the initial value) during a major event.
5. The figures from points 3 and 4 should only be treated as rough references. In reality, less profound BGP events also can contribute to RTT variations (we provide examples of such changes in section 9.2.4**Error! Reference source not found.**). Of course, tracking such events in real time and making efficient use of this knowledge in the service resolution process can be difficult for scalability reasons.

9.3.2 Figures

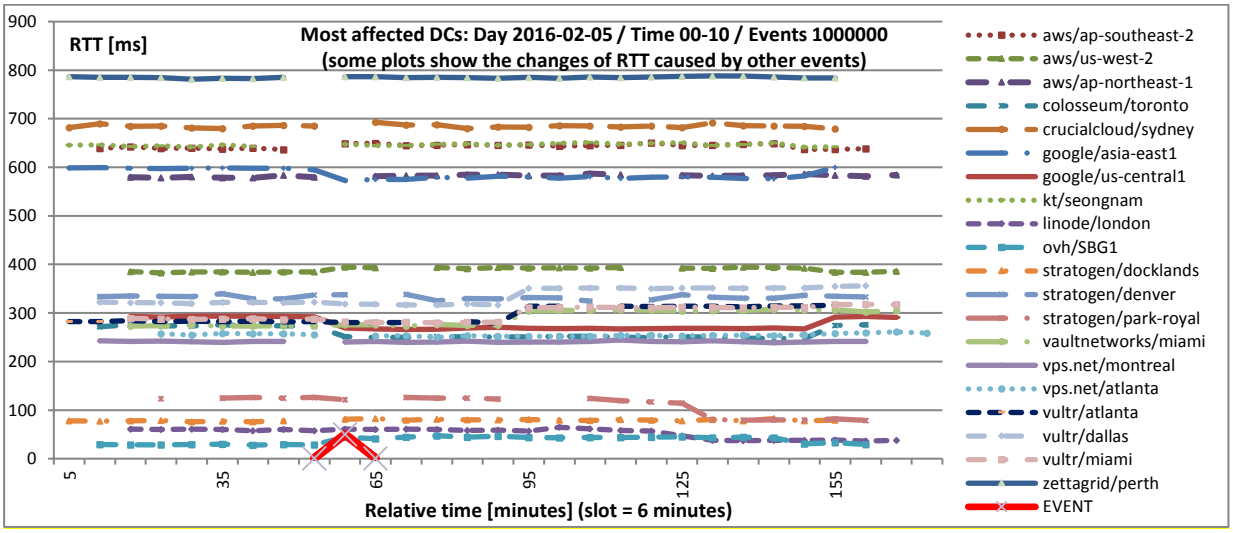


Figure 54 Event with 1200000 of notifications. Several destinations suffers from a short loss of connectivity. Few destinations experiences a small change of RTT. Visible short loss of connectivity can matter only for demanding services. The impact of the event is most profound for ovh/SBG1 (the least-valued plot) – observed increase form ~30 ms to ~45 ms may be non-negligible for several services.

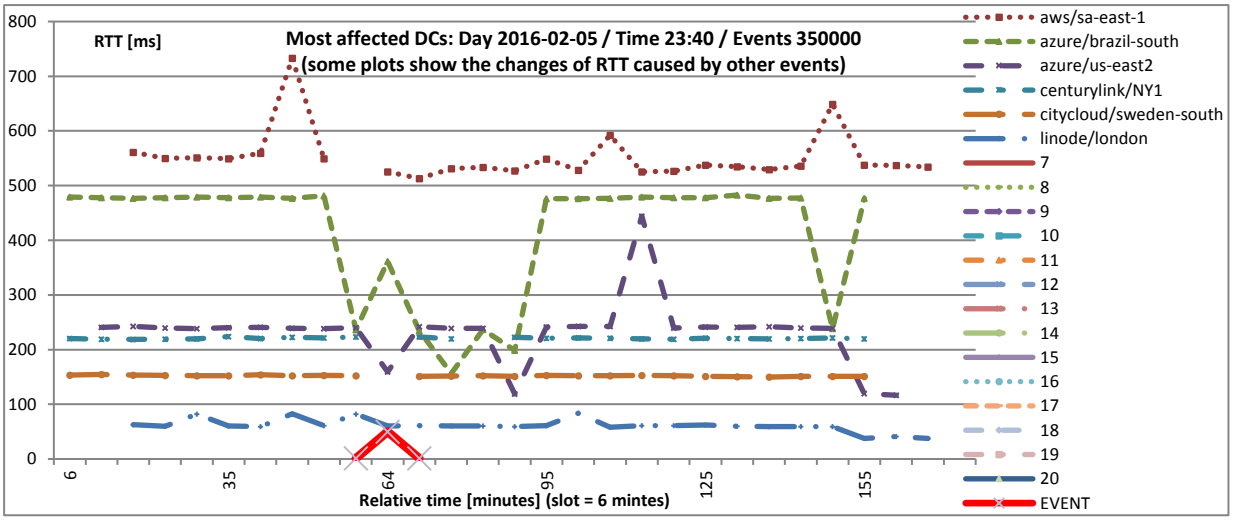


Figure 55 Event with 350000 of notifications. A small number of destinations suffers a short (one time slot) loss of connectivity. Two distant destinations experience decrease of RTT during a period of approx. 30 minutes after the event (azure/brazil-south RTT drops by approx. half). Disregarding the fact that the most affected destination (azure/brazil-south) is very distant, the duration of the impact for this DC may call for designing service resolution strategies that adopt special treatment of such destinations.

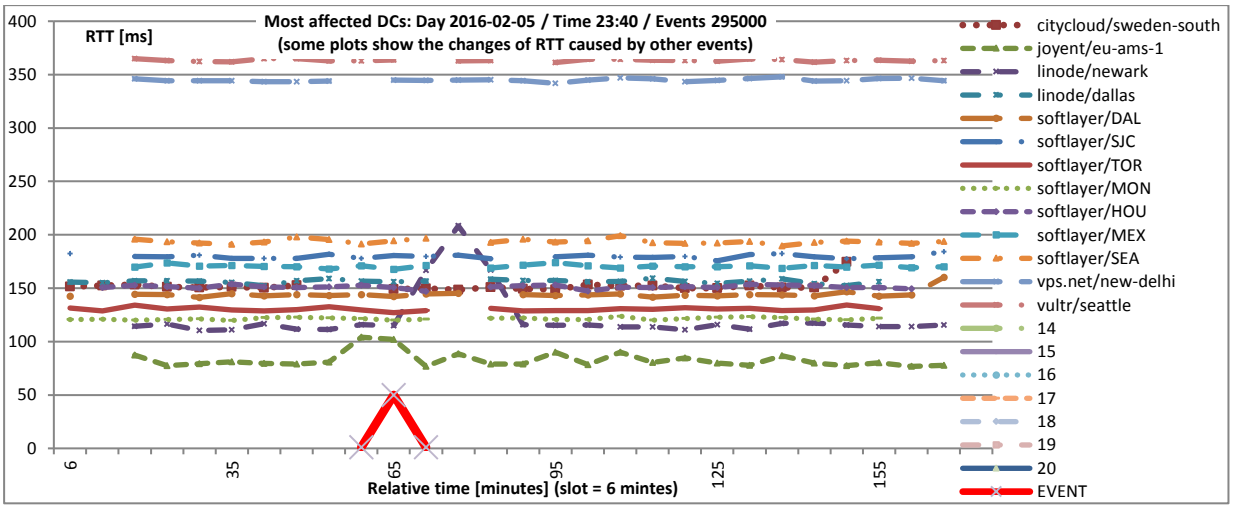


Figure 56 Event with 295000 of notifications. Temporary loss of connectivity for several destinations. No persistent change of RTT observed for any destination. This kind of variability can matter only for demanding services.

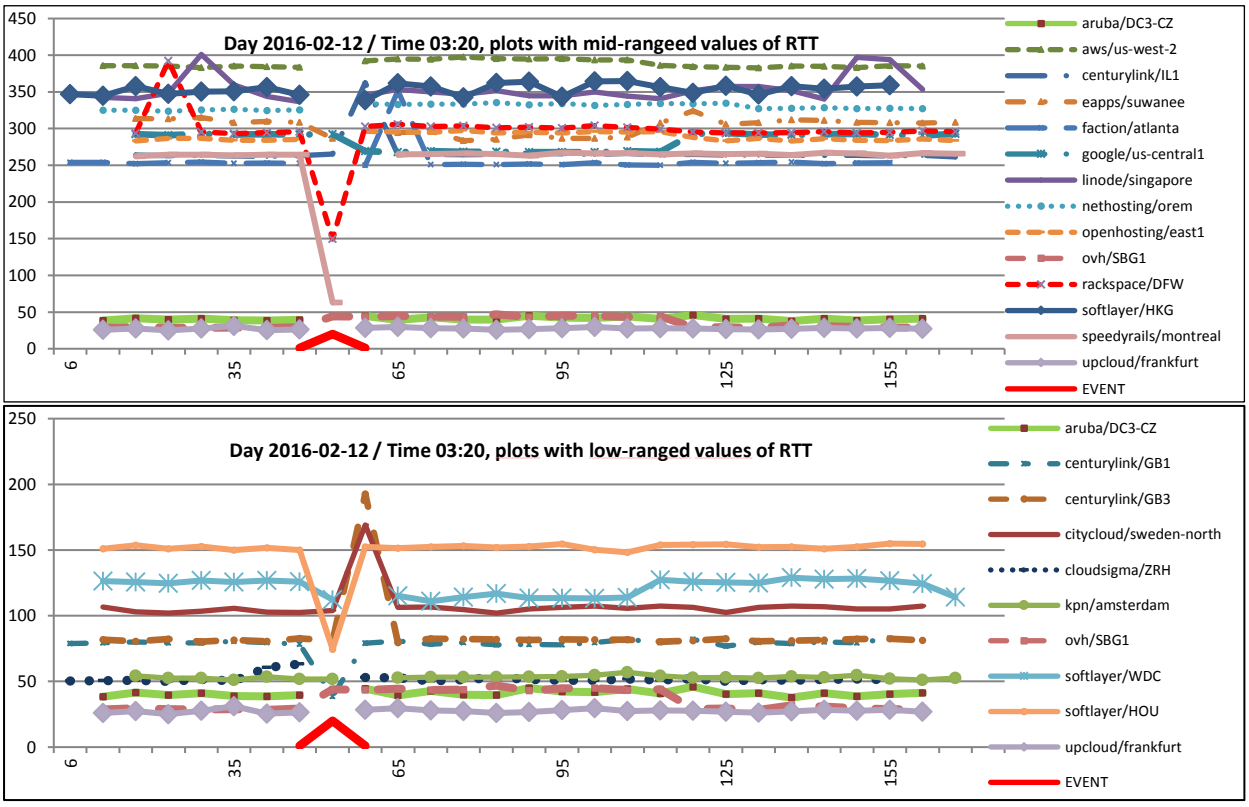


Figure 57 Event with 1200000 of notifications (plots with large values of RTT are in the upper graph while remaining plots for affected destinations are in the bottom graph). Significant number of destinations affected experiencing temporary loss of connectivity; no persistent change of RTT can be observed (except relatively small decrease for faction/atlanta). This kind of variability can matter only for demanding services.

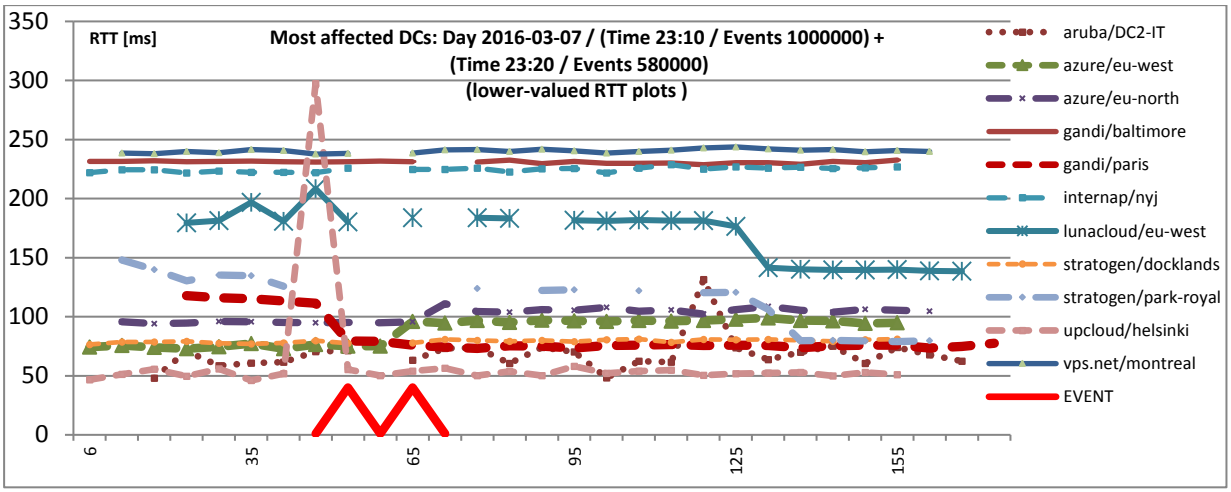


Figure 58 Two events one after another (1000000 + 580000 of notifications). Temporary loss of connectivity for several destinations, some of them suffer for several minutes (consecutive time slots), persistent change for one of them (azure/eu-west). Possible recoveries for lunacloud/eu-west and stratogen/park-royal approx. 1 hour after the event. This kind of variability can be important only from the point of view of demanding services (RTT increase observed for azure/eu-west should be sustainable for majority of services).

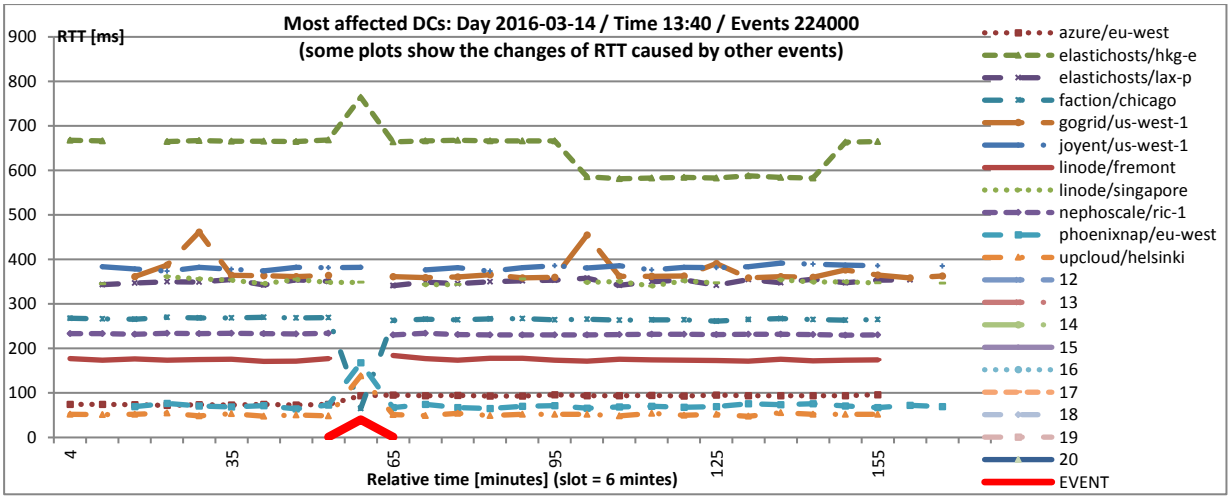


Figure 59 Event with 224000 of notifications. Several destinations affected experiencing a loss of connectivity during the transition period. This kind of impact can be important only for demanding services.

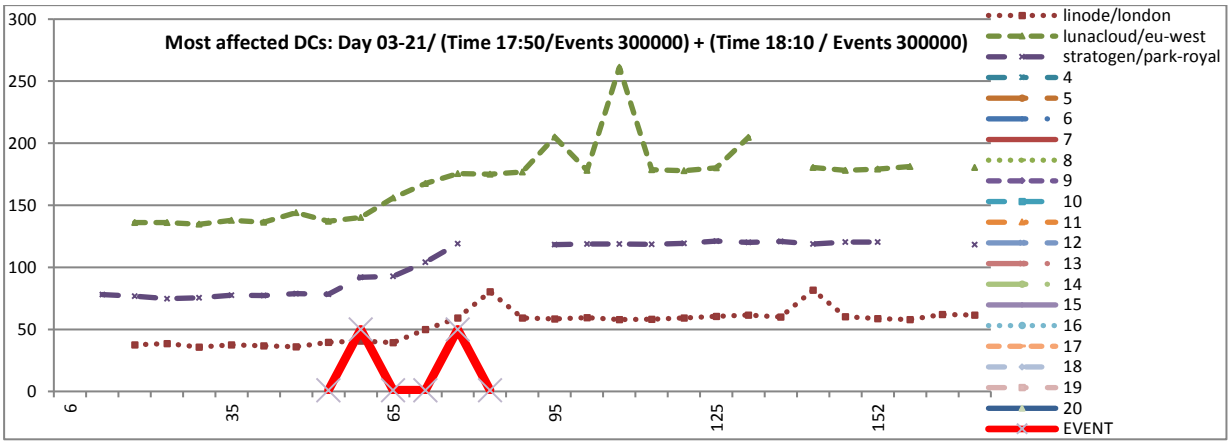


Figure 60 Event with 300000 of notifications. Noticeable change of RTT visible for 3 destinations (e.g., approx. 60% increase of RTT for linode/london). This kind of variability can be important from the point of view of different services and depending on the implementation burden can be considered in the design of service resolution platform.

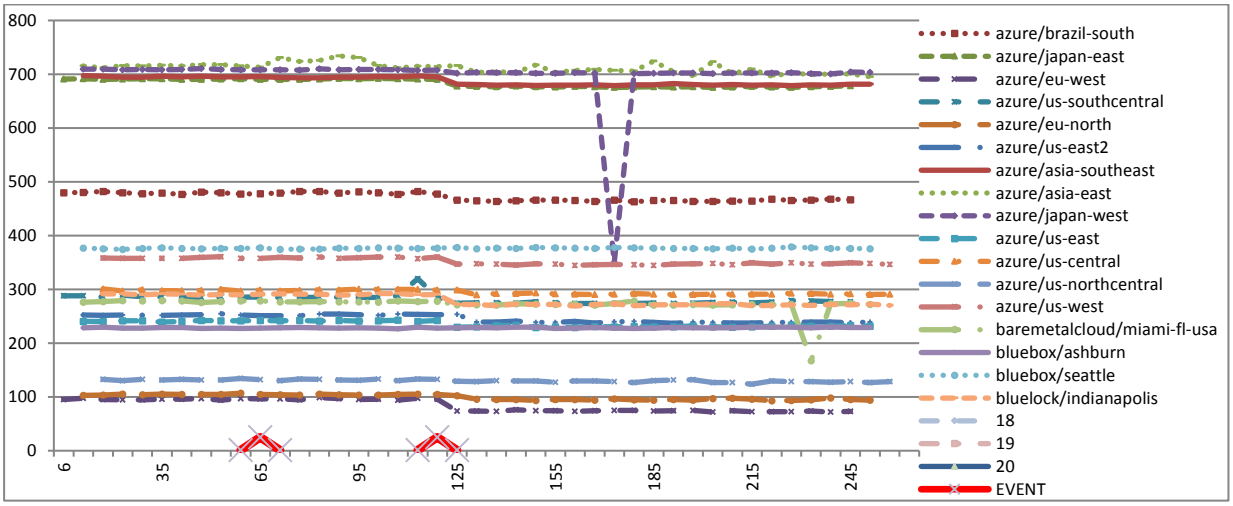


Figure 61 Double event, 300000 of notifications registered each time. RTT curves are consistent, with only a small decrease visible for a significant part of all affected destinations. This kind of variability can be accommodated by probably all services and may be disregarded from the point of view of service resolution.

9.4 Analysis of AS-internal delays

9.4.1 Measurement characterisation

The measurement set contains a selection of traces from speed-test probes located in DSL nodes across the access network of a large ISP. The overall characterisation of the measurement setup is as follows.

Each probe samples a subset of servers with a period of duration ~1 hour (approximately). Servers are located in a couple of main cities in Poland. Note that some of the probes are located in the same city as the sampled server in which case distance-to-delay projection can be very inaccurate.

Below is a list of detailed comments related to the data set we analyse.

- Delay histograms are drawn assuming 1-hour time slot duration - time (X axis) is given in hours. Every sample is mapped onto the corresponding time slot. Latency (Y axis) is given in milliseconds (rounded to the closest integer value).
- On rare occasions, probes provide multiple (two as observed) measurements in one time slot; in such a case the second measurement is taken into account in our analysis; this effect seems to be negligible.
- Sometimes a probe does not sample certain server(s) over some time period and such missing data is represented in the graph as a discontinuity of the corresponding plot; plots can also start in a time slot > 0 or terminate at time < MAX_SLOT (such losses of data result from planned probe switch-offs or unplanned hungs/outages). Moreover, sometimes the original trace contains the value NULL which means unsuccessful probing period (when the probe has not completed a given measurement which can result from server overload/outage; less probable is network congestion). In such a case latency equal to "zero" is assumed in the plots. In the plots, such points are easily identifiable as only failed samples (NULL samples) can have zero latency.
- Original data set contains two files with the suffix "*max*" and "*min*", respectively. This is directly related to the delay measurement methods used in parallel by each probe. One method is based on RTT delay measurement (10 PINGs for a given server every hour; we are provided the average RTT delay for each such sequence of 10 consecutive PINGs). The second method assumes similar timing as above and use the transfer of a small file (for latency estimation, TCP SYNC / SYNC-ACK sequences are used). Thus, every hour each probe provides two values of delay for each of servers assigned to this probe for monitoring. Those values can differ from each other (most often are the same) and we produce two output files. One of them is prefixed *"min"* and contains the minimum of the two values measured by the probe for a given time slot. The other (prefixed *"max"*) contains the maximum of each pair of the values measured for this slot. Comparing *min* and *max* figures against each other, one can analyse the degree to which the two measurement methods diverge and infer about the credibility of data.

9.4.2 Discussion and conclusions

At the end of this section, three sample figures are presented. They were obtained using the lower-bound (minimum) values of the measurement time series (refer to section 9.4.1 for the explanation about those lower-bound values). Although our full data set describes much more probe-server pairs, we have deliberately selected only representative examples to make the discussion compact.

Figure 62 relates to a server located in Warsaw and shows the trace of lower-bound delay between this server and all probes used during our study. The trace covers a period of about 23 days. An important observation for this example is that delays seem to be quite stable over time and the difference of delay from different locations towards the server (situated in the same autonomous system as probes) vary not more than by the factor of two[17] (the values range from 16 ms to 31 ms as shown in the figure).

Figure 63 contains a histogram of the lower-bound delay for another server (in another city) and a selection of probes during the period of about 28 days.

Error! Reference source not found. contains the time series describing the lower-bound delay for a given probe and three servers assigned to this probe.

Key conclusions from the study are summarised in the following:

[17] We know from other sources that RTT between probe nqs_war_ken1 located in Warsaw and some other location in this city is around 10 ms which could rise the value of the ratio to 3.

- Latencies are rather low/moderate and relatively stable (remember this is the intra-domain component of e2e RTT delay, and note also that server locations correspond to possible locations of future mini-data centres of an ISP. Such data centres could be located in future generation Points-of-Presence of ISP designed to host network functions and user services according to new service provisioning paradigms.
- In most cases, one can observe the existence over longer periods of a strong lower-bound for delays, both for the "*max*" and "*min*" files. The most obvious explanation for the existence of this bound is that it indicates pure transmission and switching delay in network devices, i.e., without queueing in routers. That would mean that the load on the links along the paths is rather low and (probabilistically) the majority of the packets do not experience queuing delay. Occasionally, the lower-bound delay has been observed to experience stable jumps by several milliseconds that lasted for quite long periods of time (e.g., hours). We conjecture that such changes can result from changing in network routing. Unfortunately, we have not been able to validate this conjecture because intra-domain routing information was not available.
- The source of higher-level variation component of latencies is not known exactly - occasionally it will be due to link load jitter (queueing effect), but alternatively it may result from varying server load (both explanations can apply to RTT and file transfer measurements). We would have to know server load histograms to eliminate/confirm the server-load component (see the case of GOOGLE where the jitter is highest among all the servers which suggests that server-load may be the dominant component of the higher-level variation of latencies).
- The differences of the pure forwarding delay from different locations towards a given point in the same autonomous system can vary (so far, a confirmed value of the ratio is three). Therefore, placing service instances in the vicinity of users and appropriate service instance resolution can be important for services demanding in terms of latencies.

9.4.3 Figures

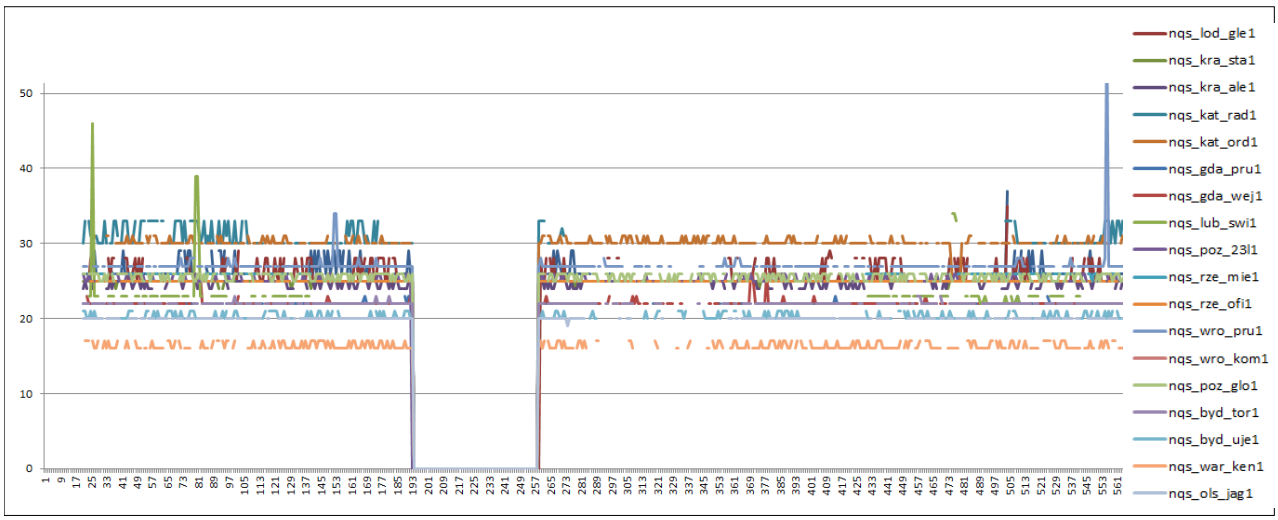


Figure 62 Lower-bound delay time series for a given server and all probes used in the study (one slot on X axis denotes one hour; time on X axis is relative).

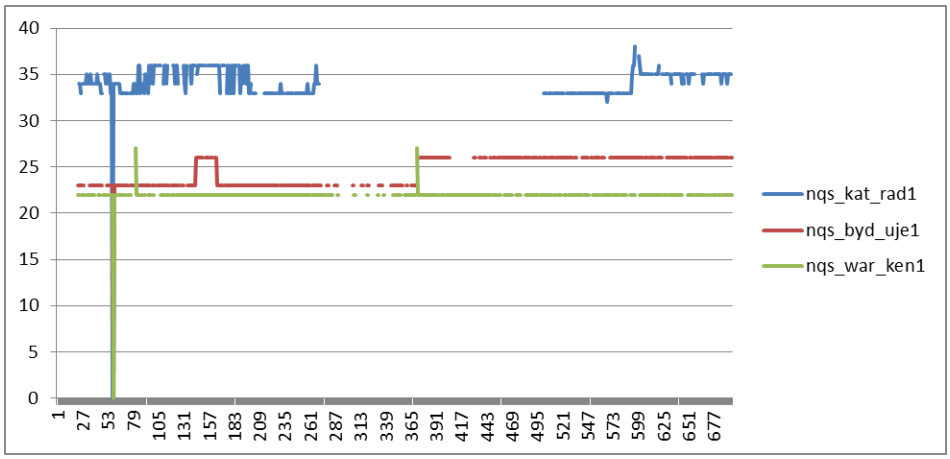


Figure 63 Lower-bound delay time series for a given server and respective probes (one slot on X axis denotes one hour; time on X axis is relative).

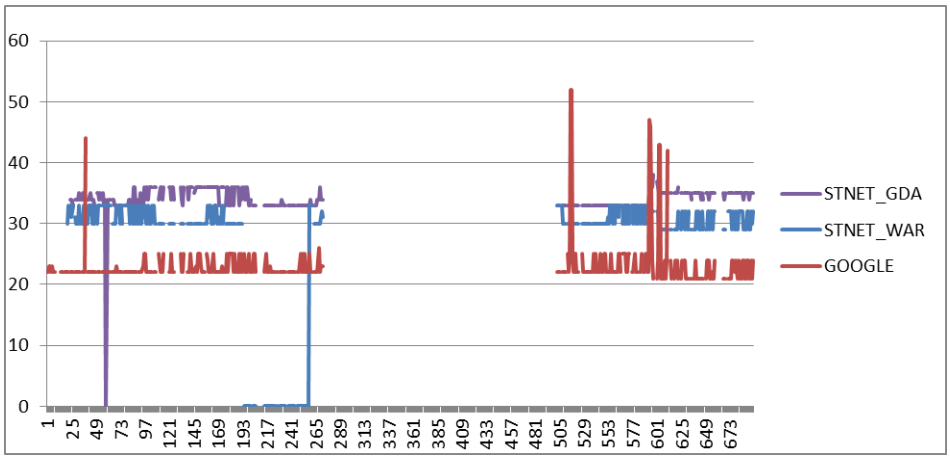


Figure 64 Lower-bound delay time series for a given probe and respective servers (one slot on X axis denotes one hour; time on X axis is relative).

10. ANNEX 2 – SERVICE REQUEST HANDLER INTERFACES

Reference architecture of the resolver is presented in section 5.1 where one can identify all interfaces specified in this section.

10.1 General information

In the following, the basic features common to all interfaces are listed.

- All interfaces and their methods are exposed as RESTful web-services,
- Every single HTTP request does operation on a single object,
- HTTP method defines action with an object:
- GET – get information about object(s),
- POST – insert or update an object,
- DELETE – delete an object.
- Object of an operation is identified by URI,
- Object definition for insert or update is delivered by HTTP request body with Content-Type `application/json or application/xml`,
- Additional arguments of an operation are delivered as `query parameters (in URI), i.e. response format may be requested by adding query parameter format=xml`,
- Content-type of the response may be requested by setting query parameter format. Permitted values are: json, xml, text. Default value (if omitted or value not accepted) of the parameter is json.
- HTTP status code of a response is always 200 – OK.
- Response body has always the same structure (see section 10.4.1).

10.2 Mapping interface (interfaces F1, F3)

Interface to insert, update, delete elements of: forwarding table, and network PIDs' definitions. This include several RESTful web-services:

10.2.1 Ping – to check Fusion Resolver availability

Web-service endpoint (Uri): `/FusionResolver/1.0/ping`
Methods: GET
Request Input Data (Body): None
Response Output Data: None

10.2.2 Networks – to get the information about all networks registered in resolver

Web-service endpoint (Uri): `/FusionResolver/1.0/networks`
`Methods: GET`
Request Path Parameters: None
Request Query Parameters:

Parameter name	Type	Mandatory	Default value	Description

detail	Boolean	no	false	If set to true, list of PID's subnets is included into response output data
cidrFilter	Boolean	no	false	If set to true, network list will be filtered to match cidr value
cidr	String	no	client IP	Parameter represents IP address or subnet to filter out network list

Request Input Data (Body): `None`
Response Output Data: `List of NetworkPID`

10.2.3 Network – to manage data of a requested network PID

Web-service endpoint (Uri): `/FusionResolver/1.0/network/{pid_name}`
Methods: `GET, POST, DELETE`
Request Path Parameters:

Parameter name	Type	Mandatory	Default value	Description
pid_name	String	yes	none	The name of the network PID

Request Query Parameters: `None`
Request Input Data (Body): `None (GET, DELETE), NetworkPID (POST)`
Response Output Data: `None (DELETE), NetworkPID (GET, POST)`

Note: Names of PIDs in Uri and NetworkPID structure must be equal. If not, operation will fail.

10.2.4 Services – get the information about all services registered in the resolver

Web-service endpoint (Uri): `/FusionResolver/1.0/services`
Methods: `GET`
Request Path Parameters: `None`
Request Query Parameters:

Parameter name	Type	Mandatory	Default value	Description
detail	Boolean	no	false	If set to true, list of services' locators (endpoints) is included into response output data

Request Input Data (Body): `None`
Response Output Data: `List of Service`

Note: Parameters to filter out the services list will be implemented later.

10.2.5 Service – to manage endpoints (locators) of the service (including their weights, policies, etc.)

Web-service endpoint (Uri): `/FusionResolver/1.0/service/{sid}`
Methods: `GET, POST, DELETE`
Request Path Parameters:

Parameter name	Type	Mandatory	Default value	Description
sid	String	yes	none	The name of the service

Request Query Parameters: None
Request Input Data (Body): None (GET, DELETE), Service (POST)
Response Output Data: None (DELETE), Service (GET, POST)

Note1: Names of the service in Uri and Service structure must be equal. If not, operation will fail.
Note2: Parameters to filter out the service's endpoint list will be implemented later.

10.3 Client-side interface (interface C1)

Interface to resolve service names (FQDN) into locators (IPs). It consists of one RESTful web service

10.3.1 Location – to resolve service name into its locator

Web-service endpoint (Uri): /FusionResolver/1.0/location/{sid}
Methods: GET
Request Path Parameters:

Parameter name	Type	Mandatory	Default value	Description
sid	String	yes	false	Name of the service to be resolved into location

Request Query Parameters:

Parameter name	Type	Mandatory	Default value	Description
client	String	no	client IP address	represents a client address, that should be used into service name resolving process instead of real client (remote) address.
type	String	no	client IP type	Defines types of location IPs to be resolved. Accepted values are: ipv4, ipv6, all. If omitted, or non-accepted value, the type will be calculated from client address type (see client parameter).
debug	Boolean	no	false	If set to true, debug info from resolving process is included into response data.

Request Input Data (Body): None
Response Output Data: Location

10.4 Object definitions

This chapter introduces definition of object structures used by the FusionResolver interfaces.

10.4.1 Response

Response is a structure which represents syntax of response for every single request.

Parameter name	Type	Occurrence		Description
		Min.	Max.	
status	Integer	1	1	status of an operation: 0 – OK, non 0 – error
message	String	1	1	text description of the operation status
data	Mixed	0	1	the real data requested in web-service call

Note: Currently "data" structure always contains array of objects, even if called web-service does operation on single object (then "data" array contains only one object – see JSON example).

Examples of Response structure in XML and JSON formats:

XML	JSON

<response>
<status>0</status>
<message>Operation successful</message>
<data> … </data>
</response>
	{
	"status": 0,
	"message": "operation successful",
	"data": [{ … }]
	}

10.4.2 NetworkPID

NetworkPID represents a structure that defines set of network ranges.

Parameter name	Type	Occurrence		Description
		Min.	Max.	
id	Integer	0	1	Id of a network PID
name	String	1	1	Name of a network PID
range	Array of NetworkRange	0	-	List of network ranges that belong to the network PID

Note: Range list in XML is enclosed in element wrapper named <ranges>.

Examples of NetworkPID structure in XML and JSON formats

XML	JSON

<networkPid id="1">
<name>POLAND</name>
<ranges>
<range> … </range>
<range> … </range>
</ranges>
</networkPid>
	{
	"id": 1,
	"name": "POLAND",
	"range": [
	{…},
	{…}]
	}

10.4.3 NetworkRange

NetworkRange is a structure which represents network subnet as a subset of a network PID.

Parameter name	Type	Occurrence		Description
		Min.	Max.	
id	Integer	0	1	Id of a network range
subnet	String	1	1	Text definition of network subnet in CIDR notation

Examples of NetworkRange structure in XML and JSON formats

XML	JSON

<range id="1">
<subnet>192.168.0.0/24</subnet>
</range>
	{
	"id": 1,
	"subnet": "192.168.0.0/24"
	}

10.4.4 Service

Service represents a structure that defines the service that will be resolved into locator.

Parameter name	Type	Occurrence		Description
		Min.	Max.	
id	Integer	0	1	Id of a service
name	String	1	1	Name of a service (SID)
endpointGroup	Array of EndpointGroup	1	1	List of endpoint groups that belong to the service (may contain many 'EndpointGroup' objects)

Note: EndpointGroup list in XML is enclosed in element wrapper named <endpointGroups>.

Examples of Service structure in XML and JSON formats

XML	JSON

<service id="1">
<name>www.orange.pl</name>
<endpointGroups>
<endpointGroup> … </endpointGroup>
<endpointGroup> … </endpointGroup>
</endpointGroups>
</service>
	{
	"id": 1,
	"name": "www.orange.pl",
	"endpointGroup": [
	{…},
	{…}]
	}

10.4.5 EndpointGroup

EndpointGroup represents a structure that defines set of service's endpoints which are resolvable to clients that belong to defined network PID.

An EndpointGroup of a service is defined by pair: sid and pid_name. It means, that the same IP may be used as locator's address for different network PID.

Parameter name	Type	Occurrence		Description
		Min.	Max.	
id	Integer	0	1	Id of an endpoint
pidName	String	1	1	Name of a network PID that the locator belongs to
endpoint	Array of Endpoint	1	1	List of Endpoints that belongs to the EndpointGroup (may contain many 'Endpoint' objects)

Note: Network PID with the name "pidName" has to be defined earlier. If not, operation will fail.
Note2: Endpoint list in XML is enclosed in element wrapper named <endpoints>.

Examples of Service structure in XML and JSON formats

XML	JSON

<endpointGroup id=”1”>
<pidName>POLAND</pid_name>
<endpoints>
<endpoint>…</endpoint>
<endpoint>…</endpoint>
</endpoints>
</endpointGroup>
	{
	“id”: 1,
	“pidName”: “POLAND”,
	“endpoint”: [{},
	{}
	]
	}

10.4.6 **Endpoint**

Endpoint represents a structure that defines the service’s locator, which is resolvable to clients that belongs to network PID defined in Endpoint’s EndpointGroup. In process of service name resolution, it is the final result.

An endpoint of a service’s endpoint group is defined by IP. It means, that the same IP may be used as locator’s address for different endpoint group or even for different service (sid).

Parameter name	Type	Occurrence		Description
		Min.	Max.	
id	Integer	0	1	Id of an endpoint
address	String	1	1	IP address of a locator
weight	Integer	1	1	Weight of a locator in that network PID
hits	Integer	0	1	Represents counter of hits for this locator. It is used rather to get information, and not to set it.

Examples of Service structure in XML and JSON formats

XML	JSON

<endpoint id=”1”>
<address>192.168.0.10</address>
<weight>2</weight>
<hits>1</hits>
</endpoint>
	{
	“id”: 1,
	“ip”: “192.168.0.10”,
	“weight”: 2,
	“hits”: 1
	}

10.4.7 **Location**

Location represents a final result of a service name resolution process.

Parameter name	Type	Occurrence		Description
		Min.	Max.	
request_id	Integer	0	1	Id of an request (if any)
service	String	1	1	Name of a service to resolve
ipv4address	String	0	1	IPv4 address of a locator
ipv6address	String	0	1	IPv6 address of a locator
debugInfo	String	0	1	Debug info from service resolving process. It has to be explicitly requested.

Examples of Location structure in XML and JSON formats

XML	JSON

<location>
<request_id>12230</request_id>
<sid>www.orange.pl</sid>
<ipv4>192.168.0.10</ipv4>
<ipv6>FE80::0202:B3FF:FE1E:8329</ipv6>
</endpoint>
	{
	“request_id”: 12230,
	“sid”: “www.orange.pl”,
	“ipv4”: “192.168.0.10”,
	“ipv6”:
	“FE80::0202:B3FF:FE1E:8329”
	}

12. ANNEX 3 – NETWORK COST/MAP INTERFACE

ALTO messages are implemented according to rfc7285. Any differences result from implementation simplifications.

12.1 Obtaining the list of network maps

Method: GET

WebService: http://217.96.70.102/alto/networkmap

or http://app.ott.rd.tp.pl/alto/networkmap

Parameters:

None

Example result:

```
{"maps":["map1","map2","default-map"]}
```

Comment:

This service allows to obtain all network maps defined in the ALTO Server.

12.2 Obtaining a network map

Method: GET

WebService: http://217.96.70.102/alto/networkmap?map=default-map

Parameters:

map

Example result:

```
{"meta":{"vtag":{"resource-id":"default-
map","tag":"151a05a2b45"}},"network-
map":{"PID1":{"ipv4":["126.20.49.1","126.20.49.2"],"ipv6":["2001:0db8:0123:
4567:89ab:cdef:1234:5678"]},"PID2":{"ipv4":["126.20.49.3"]},"PID3":{"ipv6":
["2002:0db8:0123:4567:89ab:cdef:1234:5679"]}}}
```

Comment:

This service allows to obtain map definition for a given map. *tag* element is a timestamp and changes when the map information has been changed.

12.3 Obtaining a cost map

Method: GET

WebService:

http://217.96.70.102/alto/costmap?map=default-map

Parameters:

map, cost-mode, cost-metric

Example result:


```
{"meta":{"dependent-vtags":{"resource-id":"default-
map","tag":"15148a85ebc"},"cost-type":{"cost-mode":"numerical","cost-
metric":"routingcost"}},"cost-
map":{"PID1":{"PID2":5,"PID3":7},"PID2":{"PID1":6,"PID3":9},"PID3":{"PID1":
8,"PID2":10}}}
```


Comment:

This service allows getting the cost map for a given map . tag element is a timestamp and changes when map information has been changed. In a cost-map, the first PID is the source, and the next one is the destination, e.g., cost count *from* PID1 *to* PID3 is 7.

12.4 ALTO Administration

http://217.96.70.102/alto-admin

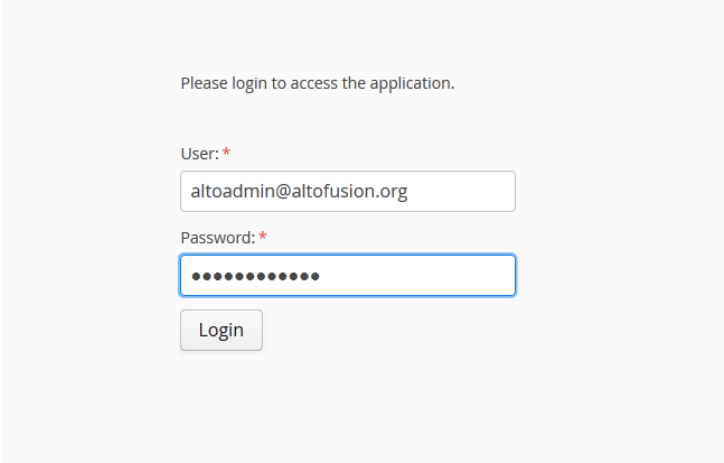
This application allows to define maps, pids and costs. Costs can be entered manually or can be read from the network (future step).

12.4.1 Login in

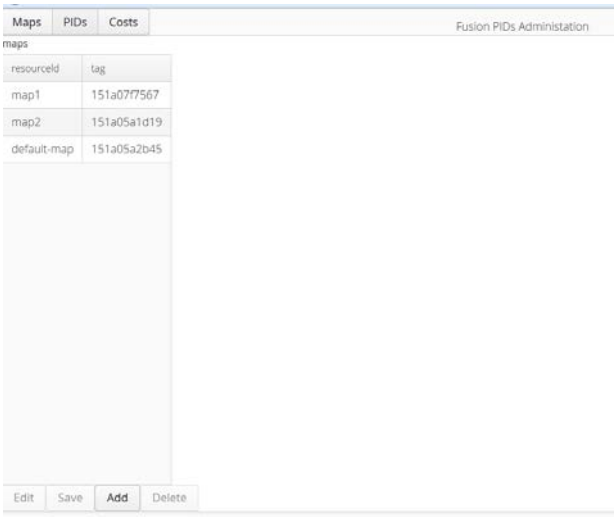
http://app.ott.rd.tp.pl/alto-admin

user: altoadmin@altofusion.org

pass: alt0pa$$w0rd



12.4.2 **Maps**

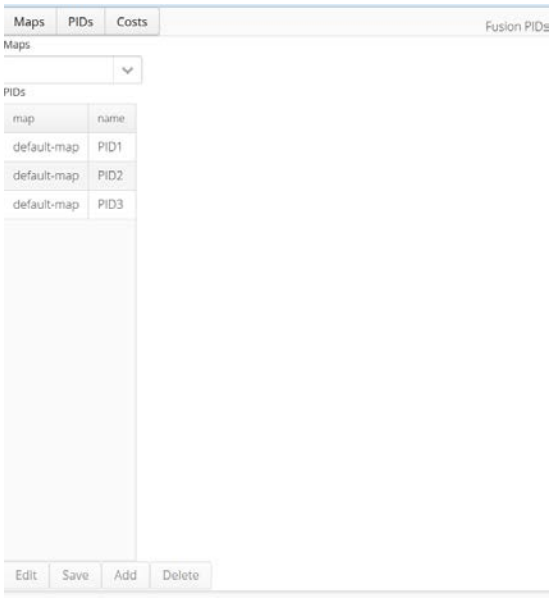


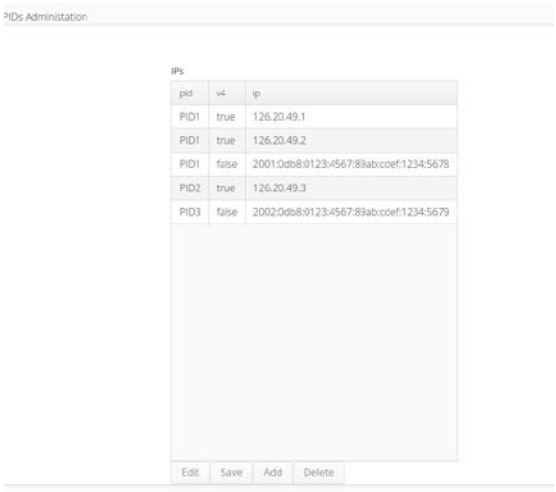
Choose Maps from the menu. Here we can define new map – Add button, delete a map – Delete button, and modify a map – Edit button. To confirm changes Save button should be pressed.

Edit and Delete operations are possible only on a selected (clicked) map. During adding a new map, the name of a new map should be unique, no consistency check has been implemented for now.

Tag is a timestamp that indicates the time of the last change in maps information. To change a tag simply press Edit button and finalise by pressing Save button.

12.4.3 **Pids**



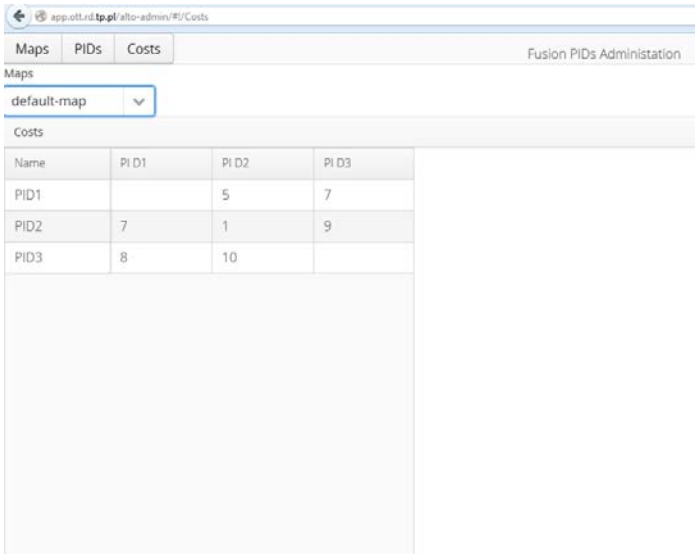


Choose Pids from the menu. Next choose the map needed. The map has a list of pids and connected to a given pid is a list of ips.

Operations on tables PIDs and IPs are similar as described for Maps.

Selecting one PID we can see IPs only for this PID. Be sure that pid names are unique inside one map, but pid names can be shared between different maps.

12.4.4 **Costs**



Choose Pids from the menu. Next choose the interested map. In the cost map row PID is “from” and column PID is “to”.

Duble click on a row o press Enter and you can edit a whole row. Empty value is changed to null and is deleted from the database. Change null to a value for adding into the database. And Change a value to change in the database. Confirm your changes by pressing Save.