Enriched Network-aware Video Services over Internet Overlay Networks

www.envision-project.org

Deliverable D3.2

Refined Specification of the ENVISION Interface, Network Monitoring and Network Optimisation Functions

Public report, v1.0, 15 January 2012

Authors

UCL	Raul Landa, Richard Clegg, Eleni Mykoniati, David Griffin, Miguel Rio
ALUD	Nico Schwan, Klaus Satzke
LaBRI	Toufik Ahmed, Samir Medjiah, Abbas Bradai, Ubaid Abbasi
FT	Bertrand Mathieu, Irène Grosclaude, Selim Ellouze, Pierre Paris, Valery Bastide, Emile Stephan
TID	Oriol Ribera Prats
LIVEU	Noam Amram, Ran Nachmany

Abstract This deliverable presents the work performed in WP3 during the second year of the ENVISION project, in relation with the CINA (Collaboration Interface between Network and Applications) interface. In this document, we review the main features of the CINA specifications. Details can be found in the drafts submitted to IETF ALTO group or in the Appendixes. We start by presenting the discovery of the CINA server, the first step for an application. Then the mutual exchanges between applications and the network is described via the defined maps (multi-cost map, footprint map, constraint map), done in a way to complement the current ALTO specifications. The network service invocation is also introduced followed by the work performed related to the security of CINA which completes the description of the interface. In order to provide network information to the application, the monitoring architecture collecting them and the implementation work, investigated during the last months, are presented in section 3. Finally, the added-value network services, a network operator can offer, which we investigated in second year are presented (design and implementation) in section 4 with a focus on the multicast service, the high capacity node, as well as the caching functions. The network optimisation logic based on preferences, which aims to shift ISP traffic in time and space to reduce transit bills, has been performed, to complete the research work on the network services with a theoretical study.

Project funded by the European Union under the
Information and Communication Technologies FP7 Cooperation Programme
Grant Agreement number 248565

EXECUTIVE SUMMARY

This deliverable contains the main achievements of work done during the second year in the WP3 workpackage:

- Specification of the discovery mechanisms: part of a draft submitted to the IETF ALTO working group,
- Specifications of the applications to network methods (multi-cost map): part of a draft submitted to the IETF ALTO working group,
- Specifications of the network to applications methods (footprint map and constraint map): possibly part of a draft to be submitted to the IETF ALTO working group,
- Specifications of the network service invocation,
- Security of the CINA interface,
- Implementation of the CINA interface: first version including overlay to network methods,
- Monitoring architecture of the network: implementation running in the Orange testbed,
- Design and implementation of the multicast network service: First version of the multicaster running in the Orange testbed,
- Design and implementation of the caching network service,
- Design and implementation of the high capacity node network service,
- Mathematical study showing interest for ISP to shift traffic in time and space.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

In this deliverable, we present the final specifications of the proposed CINA Interface (Collaboration Interface between Network and Applications), that is defined between overlay applications and network operators. This CINA interface enables mutual exchanges of information between the two actors as well as the invocation of network services provided by the network operators. Security features have been taken into consideration to ensure authentication and secure communications between the involved entities.

Despite the goal not being to design new monitoring mechanisms, the monitoring work was important in the project, so that the network operator could collect network information. Consequently, a monitoring architecture has been defined. This architecture, including the CINA server and its implementation, are presented in section 3, highlighting the information to be provided by the network operator via the CINA interface to applications.

The network services that we decided to deeply investigate in the previous deliverable (multicast, High Capacity Node and caching) have been designed, their implementation has started and a first version is available. This document presents these network services and the technical choices that have been made.

Finally, a mathematical study showing how an ISP can better price paths to incentivize the shifting of traffic in time (non time sensitive traffic) is presented in order to able to quantify the possible gains of such behaviours.

2. CINA INTERFACE

2.1 Problem Statement

The ENVISION project proposes to push the collaboration between networks and applications by defining intelligent cross-layer techniques that offer the possibility not only for exchanging detailed information between networks and applications but also for mobilizing network and user resources to provide better support where needed. This collaboration scheme is intended to offer each side, networks and applications, the opportunity to make use of the information and the resources available to adapt themselves and the content the network is conveying to the distribution context. This collaboration is a step forward towards the Future Internet as it provides a model based on communication between the applications overlays and the underlying networks which are up to now almost completely agnostic to each other.

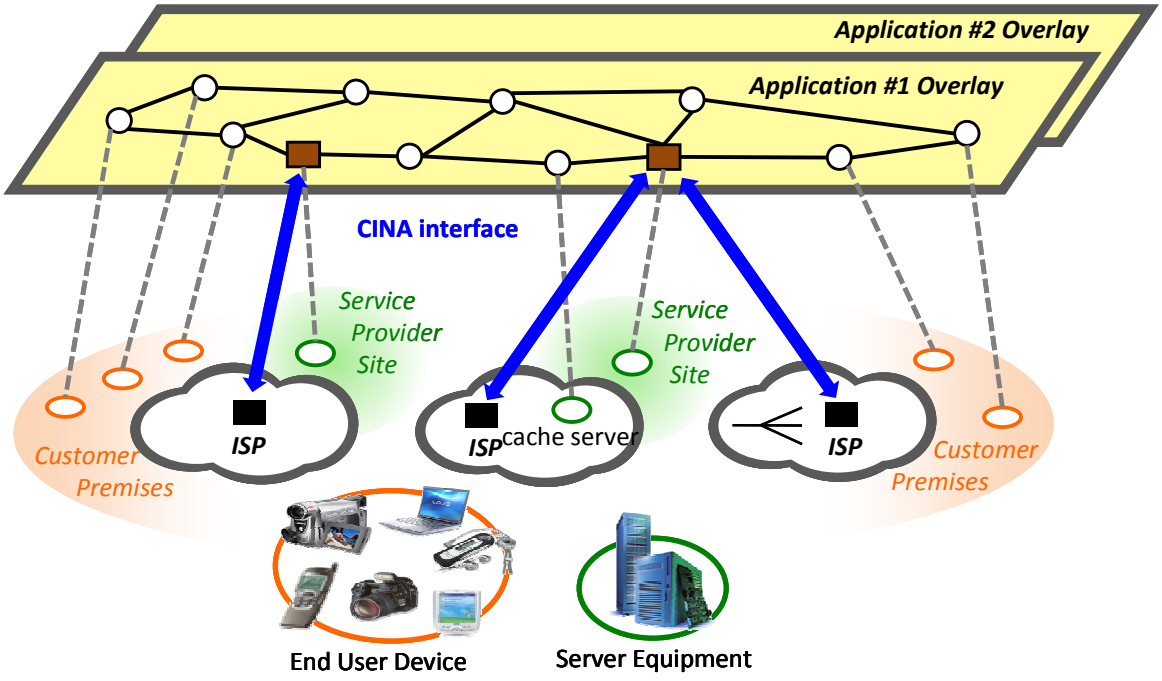


Figure 1. Envision overview: the CINA interface

Figure 1 shows an overview of the system based on the concept of collaboration between overlays relying on different underlying networks through the Collaboration Interface between Network and Application (CINA) interface. It is required from the application to have an interface with each ISP as they are each responsible for their respective domain. The role of the application is to consolidate the information gathered from the different ISPs to make use of it at an end-to-end level. However, ISP is expected to take actions at the ISP level, e.g. setting up a multicast tree for the application within its domain.

When specifying the CINA interface, we identified the information that the overlay applications can request to the network as well as the information that the network operators can request from the applications and how they can be exchanged. We also identified the network services a network operator could offer to the overlay application and how to instantiate them. Finally, this interface should be secured between the actors to avoid any malicious utilization. The following sections of the deliverable aim at presenting the solutions we have defined for the proposed CINA interface, taking into account the current ALTO specifications defined in the IETF standardization body.

2.2 Discovery of the CINA server

In the ENVISION architecture, the application communicates with the ISP though the CINA interface. To ensure this, the ISP deploys a CINA server in its network and the overlay application has an entity (e.g.; the P2P tracker in case of a centralised P2P system) that implements a CINA client. The application, running on top of possibly several ISPs, should have a mechanism to discover the CINA server of the ISPs to communicate with. This is the objective of the defined CINA discovery procedure which tells the CINA client which CINA servers to request. This section describes this discovery mechanism and more precisely a uniform mechanism for all types of CINA client deployments (see D3.1) that is implementable and deployable in today's Internet. We propose a schema which employs the UNAPTR mechanism [RFC4848] to determine the URI of the CINA server and where multiple input methods to the UNAPTR process can be used.

The specification that is described in this chapter is currently being standardized in the ALTO working group. A more detailed description of the procedure can be found in the working group item "ALTO Server Discovery" [I-D.alto-server-discovery].

2.2.1 Protocol Overview

We define multiple alternatives to discover the IP address of the CINA server, as there are a number of possible ways through which such information can be provided to the CINA client. The choice of the method is up to the local network deployment. For instance, there can be deployments where the CINA server in charge for a CINA client is provisioned by the network operator and communicated to the CINA client's host via a DHCP option, while in other deployments such means may not exist. It should be noted that there is one-size-fits-all solution to the CINA server discovery, as there are too many deployment scenarios in the server discovery space.

The following figure illustrates the different protocols that are used to find the URI of a suitable ALTO server.

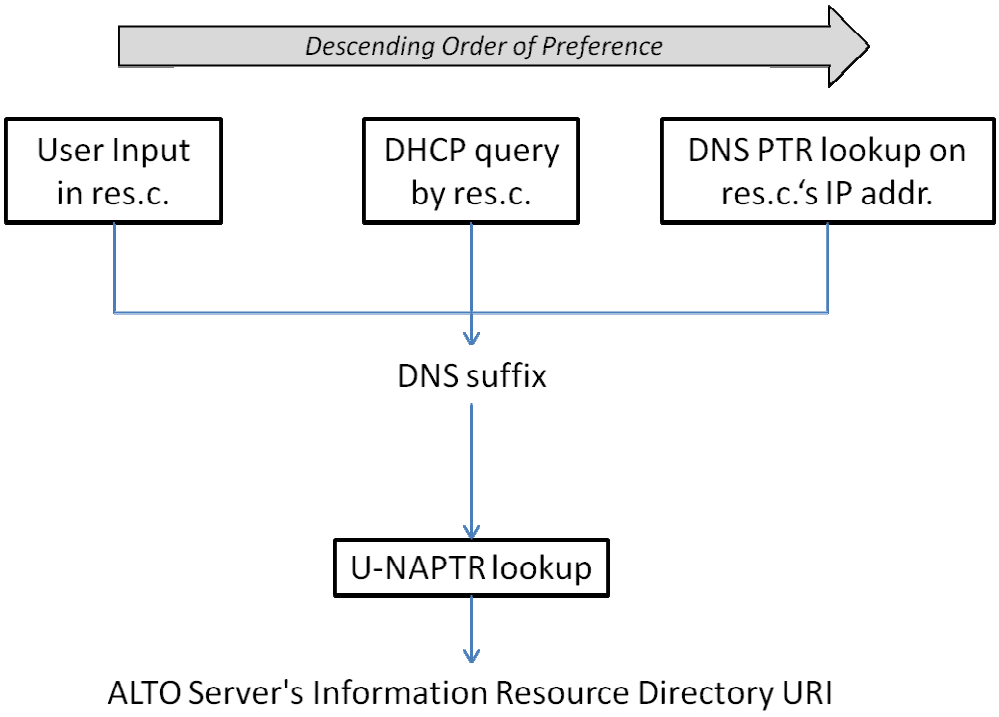


Figure 2. Protocol overview

Figure 2 illustrates the U-NAPTR based resolution process to retrieve the CINA Server URL. As a precondition for resolution the U-NAPTR process needs the right domain name as input. This domain

name is determined by the IP address of the client and the DNS suffix of the access network where the client is registered in. In order to retrieve the DNS suffix, we specify three options, as listed in a descending order of preference:

- User input: a user may manually specify the DNS suffix on his own, either to access a 3rd party CINA service provider, or if he does know such information. This input may also origin from a web page where the user downloads the configuration which is loaded as user input.
- DHCP: a network provider provides the DNS suffix through a DHCP option.
- Reverse DNS: the DNS system can be used to retrieve the DNS suffix through reverse lookup of an FQDN associated with an IP address. This is the last resort if all other options failed.

2.2.2 Retrieving the URI by U-NAPTR

This section specifies the U-NAPTR based resolution process. To start the U-NAPTR resolution process a domain name is required as input. Thus, the section is divided into two parts: Section 2.2.2.1 describes the U-NAPTR resolution process itself. The process through which the client identifies this DNS suffix of the access network and how the resource consumer is registered in is described in Section 2.2.2.2.

2.2.2.1 U-NAPTR Resolution

CINA servers are identified by U-NAPTR/DDDS (URI-Enabled NAPTR/Dynamic Delegation Discovery Service) [RFC4848] application unique strings, in the form of a DNS name. An example is 'cinaserver.example.com'.

Clients need to use the U-NAPTR [RFC4848] specification described below to obtain a URI (indicating host and protocol) for the applicable CINA service. In this document the HTTP and HTTPS URL schemes are defined. Note that the HTTP URL can be any valid HTTP(s) URL, including those containing path elements. The following two DNS entries show the U-NAPTR resolution for "example.com" to the HTTPS URL https://cinaserver.example.com/secure or the HTTP URL http://cinaserver.example.com, with the former being preferred.

```
example.com.
     IN NAPTR 100  10   "u"    "CINA:https"
          "!.*!https://cinaserver.example.com/secure!"  ""
     IN NAPTR 200  10   "u"    "CINA:http"
          "!.*!http://cinaserver.example.com!"  ""
```

2.2.2.2 Retrieving the Domain Name

The U-NAPTR resolution process requires a domain name as input. The algorithm that is applied to determine this domain name is described in this section. We specify three different options. In option 1 the user manually configures a specific CINA service instance that he wants to use. Option 2 defines a DHCP option to allow the network service provider a remote configuration of the client. In option 3 the client tries to get the domain name by performing a reverse DNS lookup on its IP address.

The resource consumer may have private IP addresses and public IP addresses and depending on the deployment it might be necessary to determine for all IP addresses the CINA server in charge of. To determine its public IP address, the resource consumer may need to use STUN [RFC5389] or BEP24 [bep24]. For the following examples we assume that the IP address of the resource consumer is a.b.c.d.

2.2.2.2.1 Option 1: User input

A user may want to use a third party CINA service instance. Therefore we allow the user to specify a DNS suffix on its own, for example in a configuration file option. The DNS suffix given by the user is combined with the IP address of the resource consumer to allow the third party CINA service to direct the client to a suitable CINA server based on the location of the client. A possible DNS suffix entered by the user may be:

```
mycinaprovider.org
```

This DNS suffix is prepended with the IP address of the resource consumer in reverse order to compose the domain name used for the final U-NAPTR lookup Section 3.1. In case there are multiple CINA servers deployed, the third party CINA service instance can direct the CINA client to the CINA server closest to the client based on the IP address.

Multiple lookups with different domain names might be necessary to complete the U-NAPTR resolution process. If there is no response for a lookup, the domain name is shortened by one part for the succeeding lookup, until a lookup is successful, as for example

```
d.c.b.a.mycinaprovider.org.
c.b.a.mycinaprovider.org.
b.a.mycinaprovider.org.
a.mycinaprovider.org.
mycinaprovider.org.
```

2.2.2.2.2 Option 2: DHCP

As a second option, network operators can configure the domain name to be used for service discovery within an access network. RFC 5986 [RFC5986] defines DHCP IPv4 and IPv6 access network domain name options that identify a domain name that is suitable for service discovery within the access network. The CINA server discovery procedure uses these DHCP options to retrieve the domain name as an input for the U-NAPTR resolution. One example could be:

```
example.com
```

2.2.2.2.3 Option 3: Reverse DNS Lookup

The last option to get the domain name is to use a DNS PTR query for the IP address of the resource consumer. The local DNS server resolves the IP address to the FQDN that also contains the DNS suffix for the respective IP address. A possible answer for a PTR lookup for d.c.b.a.in-addr.apra might be, for example:

```
d-c-b-a.dsl.westcoast.myisp.net
```

This domain name can be used for the final U-NAPTR lookup Section 3.1. If there is no response to the lookup the domain name is shortened by one part for one succeeding lookup. If there is still no response we consider the reverse lookup being failed. The domain names used for the example as described above are:

```
d-c-b-a.dsl.westcoast.myisp.net.
dsl.westcoast.myisp.net.
```

2.3 Specifications of the CINA interface

This section contains an introduction to the specification of the Collaboration Interface between Network and Applications (CINA). The interface allows a mutual collaboration between ISPs and overlay applications by providing means to exchange information between each other and by allowing network service instantiation and invocation where needed. This section aims to give an introduction to the entire specification by introducing the respective areas by extracted examples from the specifications. Further details can be found in the annex or in the respective Internet Drafts. References are included in the respective sections.

Due to its inherent close affinity, the CINA specification is closely aligned to the IETF ALTO [I-D.ietf-alto-protocol] protocol. In particular, the underlying protocol (HypterText Transfer Protocol (HTTP)) and the encoding scheme (JavaScript Object Notation (JSON)) are the same for both protocols. ALTO deals exclusively with the provisioning of network related information to the overlay layer. The way ALTO is specified allows a great flexibility for enhancements. The CINA specification thus relies, for the information provisioning from the network to the application, on ALTO which is introduced in section 2.3.1. Some enhancements like multi-cost queries, dynamic cost support and incremental updates of network and cost maps, which are currently missing in the ALTO core protocol, are discussed in section 2.3.1.2. Section 2.3.2, then introduces CINA methods for service invocation and instantiation, using the example of the multicast service specification. Finally, section 2.3.3 introduces how information from the application can be provided to the CINA server which can be used by an operator to optimize its network.

2.3.1 Information from the Network

The following services describe the options a client application has to retrieve information or invoke services by/from the CINA interface. They are based on the services that the IETF ALTO protocol [I-D.ietf-alto-protocol] provides. The ALTO protocol defines a set of basic methods that can be used to retrieve different kinds of information from a network operator. The design is flexible and allows the definition of extensions that go beyond the current scope of ALTO, which is currently limited to static information, such as routing costs. The specification for those methods that will convey information from the network to the application is thus based on ALTO. In the following section the available methods are introduced, before a discussion of the extensions.

2.3.1.1 CINA/ALTO Method Introduction

- Server Information Service:

  The Server Information Service offers information to the client about the services the server supports, as well as the network cost types (e.g. routing-cost, latency) that can be retrieved and the network services that can be invoked.

- Map Service:

  The Map Service allows a client to retrieve the network map from a CINA server. The network map illustrates the network from the view of the CINA server. In addition, a cost map can be retrieved, that depicts routing costs between the defined network regions. A CINA client may specify to retrieve network metrics as cost from the server.

- Map Filtering Service:

  The Map Filtering Service allows a client to retrieve only a part of the whole map from the server.

- Endpoint Property Service:

  The Endpoint property comprises methods to allow clients to query generic properties from endpoints. This could be for example the network ID or any other property.

- Endpoint Cost Service:

  Additionally, the endpoint cost lookup allows clients to explicitly specify endpoints for which it wants to receive cost statements. In addition to generic routing costs, predictions for different network metrics can be requested by the client.

2.3.1.2 CINA Enhancements

This section discusses several enhancements that are not supported by the current version of the ALTO core protocol. In particular, section 2.3.1.2.1 discusses the support of multiple cost types in one request. Section 2.3.1.2.2 discusses enhancements for dynamic costs support, while section 2.3.1.2.3 discusses how incremental updates of network and cost maps can be supported.

2.3.1.2.1 Multi-Cost

The ALTO working group has been created to allow a mutual cooperation between P2P applications and NSPs (Network Service Providers), in particular, because P2P bulk file-transfer applications have created a huge amount of intra-domain traffic. By aligning overlay topologies according to the 'routingcost' of the underlying network both layers are expected to benefit in terms of reduced costs and improved Quality-of-Experience. However other types of overlay applications might benefit from a different set of path metrics. In particular for real-time sensitive applications, such as gaming, interactive video conferencing or medical services, creating an overlay topology with respect to a minimized delay is preferable. However, it is very hard for a NSP to give accurate guidance for this kind of real-time information. Instead, probing through end-to-end measurements on the application layer has proven to be the superior mechanism. Still, a NSP might give some guidance to the overlay application, for example by providing statistically preferable paths with respect to the time of a day. Also static information like hop-count can be seen as an indicator for the delay that can be expected.

A second use case is motivated through draft [I-D.jenkins-alto-cdn-use-cases]. The request router in today's CDNs makes a decision about which surrogate or cache node a content request should be forwarded to. Typically, this decision is based on locality aspects, i.e. the surrogate node which is closest to the client is preferred by the request router. An ALTO server hereby is one promising option to allow NSPs to give guidance to the CDN about which cache node would be preferable according to the view of the network by the 'routingcost' Cost Type. Providing this kind of information is particularly important as the current trend is to place cache nodes deeper into the network, which results in the need for finer grained information.

While today, distance is the predominant metric used for routing decisions, other metrics might allow sophisticated request routing strategies. For example, the load of a cache node in terms of CPU utilization, memory usage or bandwidth utilization might influence routing decisions for load-balancing reasons. There exist numerous ways of gathering and feeding this kind of information into the request routing mechanism. As ALTO is likely to become a standardized interface to provide network topology information, for simplicity other information that is used by a request router could be provided by the ALTO server as well.

The use cases described above show that different type of applications will need different cost-types with respect to their specific requirements. In the current version of the ALTO protocol, it is impossible to query multiple cost types with one request. Instead a client needs to query the server multiple times, leading to unnecessary messages, an increased latency of the service, and a higher load on server and network. Thus, we propose to extend the ALTO protocol to allow querying for multiple cost-types with one request by allowing the client to specify the costs in a vector in the request.

In this document, we describe typical examples for this modification in the remainder of this section. Further details can be found in [I.D.draft-randriamasy-alto-multi-cost], which specifies updates of the ALTO JSON object formats:

- object DstCosts
- object InfoResouceCostMap
- object ReqFilteredCostMap
- object ReqEndpointCostMap
- object EndpointDstCosts
- object InfoResourceEndpointCostMap

2.3.1.2.1.1 Example: Updates on object DstCosts

The term Single Cost tags items as defined in the current ALTO protocol draft version 10[I-D.ietf-alto-protocol]. For the support of multiple cost types, we propose to replace JSONNumber by JSONArray. Note that the evolution from JSONNumber to a more generic type has already been suggested in the ALTO WG.

SINGLE COST ALTO

```
object DstCosts {
     JSONNumber [PIDName];
     ...
   };
```

MULTI COST ALTO

```
object DstCosts {
     JSONArray [PIDName];
     ...
   };
```

2.3.1.2.1.2 Example: Updates on object InfoResouceCostMap

For the support of multiple cost types, we propose to change member cost-mode to member cost-mode<1..*>. Similarly, we propose to change member cost-type to cost-type<1..*>.

SINGLE COST ALTO

```
object {
     CostMode    cost-mode;
     CostType    cost-type;
     VersionTag  map-vtag;
     CostMapData map;
   } InfoResourceCostMap;
```

MULTI COST ALTO

```
object {
     CostMode cost-mode<1..*>;
     CostType cost-type<1..*>;
     VersionTag  map-vtag;
     CostMapData map;
   } InfoResourceCostMap;
```

2.3.1.2.2 Dynamic Costs

Services like Facebook or YouTube rely on data replication across multiple sites for several reasons such as offloading the core network or increasing user experience through short latency. As content is generated constantly, data also needs to be replicated across the various locations of a CDN provider, leading to bulk data transfers between datacenters. Scheduling these data transfers is a non-trivial task as the transfer should not infer with the user peak demand to avoid degradation of user experience and to decrease billing costs for the datacenter operator by leveraging off-peak hours for the transfer. This peak demand typically follows a diurnal pattern according to the

geographic region of the datacenter. However, one precondition to schedule transfers is to have a good knowledge about the demand and to link utilization patterns between the different datacenters and networks. Provisioning this data gets increasingly complex with the number of CDN nodes and in particular the number of datacenter operators that are involved. For allowing CDN operators to better schedule transfers across administrative domains (which becomes even more significant through the CDNi protocol), the ALTO can provide time sensitive utilization maps through a dedicated service.

Another use case that stresses the need for multi-timeframe information is the case where private users or user groups having limitations in their connectivity either in time or resources or both. These kind of users need to plan their data transfers to and from various locations and be sure to optimize for example the 'routingcost' jointly with some 'pathoccupationcost' and possibly the 'hopcount'. These users often have very poor means to have any information on the network that may provide them some guidance and the ALTO protocol could help while optimizing traffic on networks that face continuous resources and/or connectivity challenges.

While these dynamic cost maps have been proposed in early versions of the before mentioned multi-cost draft [I.D.draft-randriamasy-alto-multi-cost], a dedicated draft will detail dynamic costs in future. One example of a network map that provides cost values hourly distributed over the day is depicted in the HTTP response of an alto-endpointmulticast service below:

```
HTTP/1.1 200 OK
Content-Length:
Content-Type: application/alto-endpointmulticost+json

{
  "meta" : {},
  "data" : {
    "cost-type" : ["routingcost", "pathoccupationcost"],
    "cost-mode" : ["numerical", "dynamic"],
    "map" : {
      "ipv4:192.0.2.2": {
        "ipv4:192.0.2.89"    : [1, [7, ..., 24 values]],
        "ipv4:198.51.100.34" : [2, [4, ..., 24 values]],
        "ipv4:203.0.113.45"  : [3, [2, ..., 24 values]]
      }
    }
  }
}
```

2.3.1.2.3 Incremental Updates

The ALTO protocol uses Network and Cost Maps to allow the ALTO server to specify its own aggregated network view. Essentially the Network Map contains information on how the endpoints are grouped together, which is typically done according to their proximity. The Cost Map contains Path Costs between the network regions defined in the Network Map. The size of these maps strongly depends on the scenario an ALTO server is configured for by its operator. While in some scenarios both maps might only comprise small number of PIDs, others need much greater accuracy. For large maps partial updates might become necessary.

Both map types have slightly different characteristics. Network Maps in general are expected to be smaller than Cost Maps. As an example, a Network Map with 5,000 PIDs, each having 10 CIDRs will result in a map with the size of roughly 1.25 megabytes. A Cost Map in contrast contains a m*n matrix for cost entries. Even for short PID names a full cost map for 5,000 PIDs takes up to 417

megabytes. Network Maps are also seen to be less dynamic than Cost Maps. This is due to the fact that the topology an ALTO server sees changes slower than the path costs of the network. Another characteristic is that changes to the Network Map will impact the Cost Map, whereas vice versa is presumably not the case. A final discussion on whether partial updates are useful for both map types is out of the scope of this document.

Draft [I.D.draft-schwan-alto-incr-updates] discusses enhancement options that allow partial updates of Network and Cost Maps. It focuses on two separate problems that need a solution. The first part discusses how an ALTO client and an ALTO server can synchronize their map state without transmitting the whole map. This is needed to identify whether a partial update can be applied and also to calculate the partial update itself. The second part of the document discusses how partial updates can be encoded and sent to the client. While further details can be found in the draft, the remainder of this section gives two examples of how incremental updates could be supported.

2.3.1.2.3.1 If-Modified-Since HTTP Header

To allow a server sending incremental updates to a client, it first needs to know what version of the map a client already has. One possible option is to use the HTTP If-Modified-Since header in the request if the client has previously contacted the ALTO service and thus already has a version of the map. Therefore it caches both, the map as well as the value of the Date header field of the HTTP response that contained the map. A server can then use the value of the If-Modified-Since header to compare if the clients current map is still up-to-date.

The following figure illustrates a GET request for a Network Map Information Resource. Additionally the client indicates the time when it retrieved the Network Map the last time in the If-Modified-Since header field.

```
GET /networkmap HTTP/1.1
Host: alto.example.com
Accept: application/alto-networkmap+json,
        application/alto-error+json
If-Modified-Since: Sat, 24 Dec 2012 19:43:31 GMT
```

A server retrieving this request uses the timestamp provided by the client to decide whether to send a full map, only partial updates of the map or no map at all in case there were no changes.

In case the Network Map has not been modified since the time provided by the client in the request, the server should reply with a 304 HTTP response (meaning “Not Modified”). The client can then cache the updated value of the Date header field for future requests.

In case the server determines that the timestamp provided by the client is out-of-date and cannot be reused for a partial update, it returns the full Network Map, as defined in the ALTO core protocol specification. In case the server determines that there was a change to the Network Map the server may choose not to send the full map, but a partial update only.

2.3.1.2.3.2 JSON patch

JSON Patch [I-D.pbryan-json-patch] defines a JSON document structure that allows partial modifications to a JSON document and defines the associated media type "application/json-patch". Therefore JSON Patch is a suitable option for incremental updates of the Network and Cost Maps. JSON patch supports add, remove and replace operations that can be used in combination with JSON Pointers to modify values and arrays of the JSON document members.

An ALTO client that wants to be updated if the resource has changed, needs to signal to the server that it is able to receive and understand JSON Patch updates. This can be done by including the media type "application/json-patch" in the Accept header field of the HTTP request.

The following figure illustrates one example where the server decides to send a partial update to the client using JSON Patch. The server indicates this in the response Content-Type header. In the following example the Network Map has changed. The map-vtag element has been incremented by 1, which results in a replace operation for the respective element containing the new value. Also two new subnets are added to the Network Map in PID1 and PID2 by two add operations at the indexes 1 and 0 of the ipv4 arrays.

```
HTTP/1.1 200 OK
Content-Length:
Content-Type: application/json-patch
Date: Sat, 26 Dec 2012 19:43:31 GMT
 {"replace": "/data/map-vtag", "value": "1266506140" },
 {"add": "/data/map/PID1/ipv4/1", "value": "198.51.200.0/25"},
 {"add": "/data/map/PID2/ipv4/0", "value": "198.51.200.128/25" }
```

2.3.2 CINA Service Invocation

The CINA interface not only allows application to retrieve network related information, but further also offers the possibility to instantiate network services. Currently, specifications for a multicast network service, the High Capacity Node network service and a Caching Network Service have been defined. The IP multicast service is made available through a multicaster proxy, which transforms content received in unicast to a multicast flow, and a unicaster proxy, which transforms a multicast flows to an unicast flow. More details on this service can be found in [D3.1, section 5.2] and in section 4.2. The second service, which is not further detailed in this section, is the caching service, where the network operators can cache application data and deliver it more efficiently when requested multiple times. More details on this service can be found in [D3.1, section 5.4] and in section 4.3. Finally, the investigated network service is the High Capacity Node service which is able to be part of a peer-to-peer overlay and is instantiated at network regions at the vicinity of many peers of the overlay. In particular, this service runs on a router that is able to host third party applications. More details on this service can be found in [D3.1, section 5.2.7] and in section 4.4.

The list of the aforementioned services, supported by a particular CINA server, can be discovered by a client using the normal ALTO means; i.e., the client can browse the Information Resource Directory which lists all the available services, including a description of the respective capabilities and the contact URIs where a service can be invoked.

In this section we elaborate on the design of the interface for service invocation in general, using the multicast network service as an example and its implementation can be found in section 4.2.2.3. The full interface specification for the multicast service can be found in Appendix A.

Unlike information retrieval where the retrieved resource is fully controlled by the server, some aspects of a network service are controlled by the CINA client, e.g. the points of attachment to the network. Managing a network service, therefore, involves a set of different interactions between the CINA client and server, to invoke, manage the various aspects of and teardown the service. There is not, therefore, a single data format that can capture the service and be used for all methods, while registering with IANA a dedicated media-type for each input and response messages would be unpractical. In order to enable the specification of the request and response message body formats

for a set of operations without the need for standardising dedicated media types a JSON schema [JSchema] is specified that captures all the required formats.

The use of JSON schema also allows extending the basic service with optional capabilities which may introduce additional parameters and operations without altering the specification of the basic service. In the multicast service example, we define two JSON schemas, the multicast-schema for the basic service and the multicaster-schema (see Appendix A) which inherits all the properties of the multicast-schema, extending it with additional parameters that are specific to the use of the multicaster function. At the capability discovery phase, the CINA server would need to advertise both supported schemas at the media-types and accepts fields of the corresponding resource entry (see section Appendix).

The CINA specifications for service invocation follow as closely as possible the RESTful design approach, also adopted by ALTO. The service invocation creates a resource, which can then be accessed with HTTP PUT, POST or PATCH for updating and HTTP DELETE for tearing down the service. In the case of the multicast service example, we chose to group all the updates together to a single message format used with HTTP PATCH. However, other services may choose to use HTTP PUT or POST to implement partial updates of their service, defining the corresponding message formats. The format for the HTTP PATCH message body follows the JSON PATCH specification [I-D.pbryan-json-patch], also documented in section 2.3.1.2.3.2.

2.3.3 Information from the Application

This section specifies the methods providing the Footprint and the Constraint maps of the extension Overlay to Network information.

2.3.3.1 The Application Footprint Map

The Footprint Map provides information on the distribution of the Application end-points over a Network map. The entries of a Footprint Map are taken in the set of PIDs defined by the Network map. Typically, the ALTO client provides information on a restrained set of PIDs corresponding to the application view.

2.3.3.1.1 Footprint Map attributes

The Footprint Map attributes include:

- FootprintType: identifies what the Footprint represents. The types defined in this document include "number" indicating the number of endpoints and "sessions" indicating the number of sessions.
- FootprintMode : identifies how the footprint should be interpreted. The mode defined in this document is numerical.

2.3.3.1.2 Footprint Map structure

The footprint map provides the repartition of the different kinds of end-points over the set of PIDs defined by the Network Map. The media type is "application/alto-appfootprint+json". This resource is sent to the CINA server using the HTTP POST method. Input parameters are at the format of the media type "application/alto-appfootprintparams+json".

The Footprint map “FootprintMapData” is an array indexed by the PID name. The encoding of an instance of type “FootprintMapData” describes the PIDs pid1, pid2, pid3, etc: {"pid1":{data},

"pid2":{data}, "pid3":{data},… }. The “EndpointType” is a string defining the type of the end-point: "source", "client", "cache".

An example of the encoding of 37 clients and 1 source in the pid1

```
{
  "footprint-mode" : "numerical",
  "footprint-type" : "number",
   "map-vtag" : "1266506139",
   "map" : {
   "PID2": { "client" : 37, "source" : 1},
   "PID3": { "client" : 51 }
   }
}
```

The ALTO client provides in the request only PIDs belonging to the network map identified by the field map-vtag in order to be coherent with the network maps update.

2.3.4 The constraint Map

The application constraint information service (refinement of the cost map by the application) provides information on the constraints applied on the network by the overlay data distribution between pairs of PIDs of a network map previously received from the server.

The entries of a Constraint Map are taken in the set of PIDs of the Network map. Typically, the ALTO client provides information on a restrained set of PIDs corresponding to the application view. As an example, this may correspond to the PIDs where the application has end-points, faces bad Quality of Experience or is expecting important traffic.

2.3.4.1.1 Constraint Map attributes

The Constraint Map attributes include:

- ConstraintType : identifies what the constraint represents. The types defined in this document include "traffic-Gbps" indicating the traffic between network PIDs in Gbps and "RTT-ms" indicating the RTT between end-points in PIDs in ms.
- ConstraintMode : identifies how the constraint should be interpreted. The mode defined in this document is "numerical".
- ConstraintPeriod: specifies the period of time during which the data characterizes the application constraints on the network. This period could in the near past, the present or the future. It is defined as an array of 2 numbers: [startlimit, endlimit]. The format of each number is YYYYMMDDHHMM.

2.3.4.1.2 Constraint Map structure

The Constraint Map media type is "application/alto-appconstraint+json". This resource is sent by the CINA client to the CINA server using the HTTP POST method. Input parameters are at the format of the media type "application/alto-appconstraintparams+json". It corresponds to the JSON object “InfoResourceConstraintMap”.

The ALTO client provides in the request only PIDs belonging to the network map identified by the field map-vtag (as done for the Footprint Map).

An example of a Constraint Map showing the traffic constraints of the application between the PIDs 1, 2 and 3 of the network map "1266506139".

```
{
        "cost-mode" : "numerical",
        "cost-type" : "traffic_Gbps",
        "map-vtag" : "1266506139",
        "constraint-period": [201106101245, 201106101300]
        "map" : {
             "PID1": { "PID1": 3,  "PID2": 100,  "PID3": 80 }
             "PID2": { "PID1": 20,  "PID2": 5,  "PID3": 150 }
        }
}
```

2.3.4.2 The Footprint Information Service

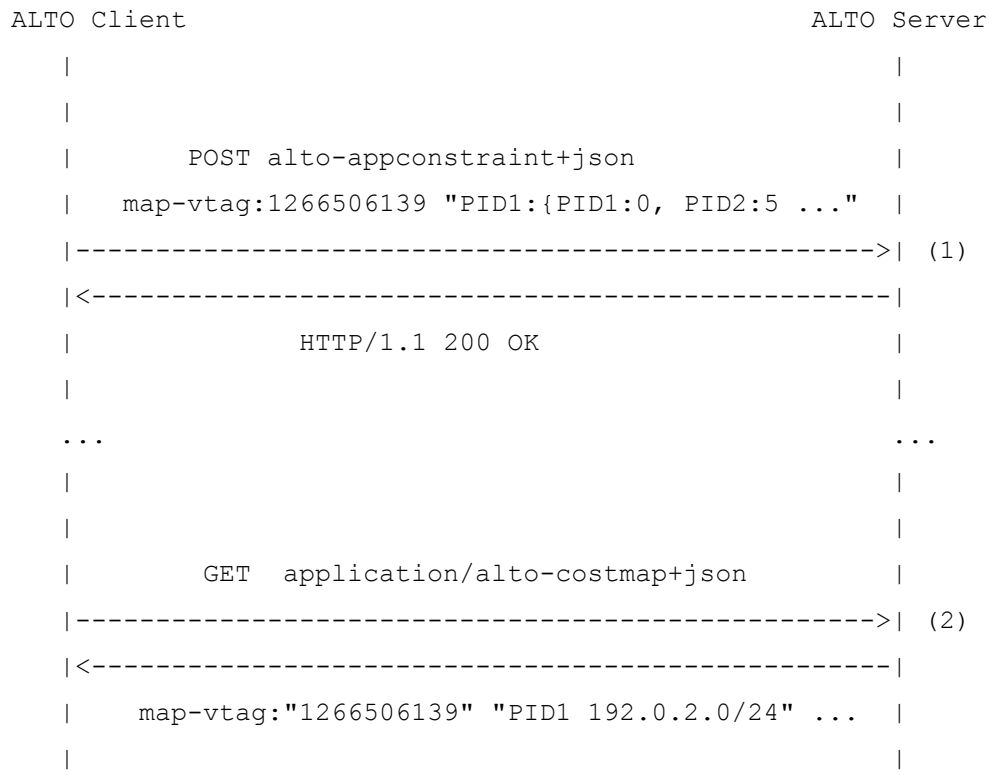
The Footprint Information Service provides an ALTO server with a way to receive additional information on the PIDs of its network map from an ALTO client. The application MUST not query the Footprint Information Service if the server does not expose the service in its Information Resource Directory.

2.3.4.3 The Constraint Information Service

The Application Constraint Information Service provides an ALTO server with a way to receive application constraint between its PIDs. There are various kind of constraints like traffic forecast and network QoS constraints between PID pairs.

The application MUST not query the Constraint Information Service if the server does not expose the service in its Information Resource Directory. In the following example, the Application ALTO client

uploads its Constraint Map and gets a Cost Map taking into account the information provided by the Constraint Map.



2.4 Security of the CINA server

2.4.1 High-level requirements

The AAA & security issues regarding the CINA interface can be split into 4 processes:

1. Authentication: The authentication is the first step of the security process allowing each side to determine the identity of the other side.
2. Authorisation: The second step is to determine the privileges accorded to the applications and their entities. Based on their own policies and agreements with each application, ISPs define different level of access to their information and resources.
3. Data Delivery: the next step is to secure the communication channel between ISPs and applications in a manner that no other entity is capable of intercepting or modifying the exchanged data or CINA methods. It is required that encryption with sufficient strength level and digital signatures are used for the exchanged data.
4. Data usage: the final step of the process is to ensure that the data securely exchanged between the communicating parties remain confidential and used intentionally for the collaboration process only, preventing any illegal distribution.

2.4.2 The Collaboration security system

The security aspects of the investigated collaboration concerns 3 identified actors: the network, the application and the client representing the application as explained in the following section.

2.4.2.1 Context clarification

While the network is represented by the CINA server, the overlay application is represented by a CINA client. Due to scalability and security reasons, more than one client may be used to represent the application within the collaboration procedure (Figure 3-a). Moreover, the same client may handle the collaboration procedure for different applications (Figure 3-b).

The scalability reasons for a multi-client collaboration include a growing load on the client nodes and the proximity to end-users. It may be addressed through:

- The distribution of specific services among different client nodes, e.g. request routing, multicast, adaptation, information consolidation, etc.
- Topological distribution, e.g. each node is responsible of a part of the overlay,
- Simply sharing the load between the client nodes to face a growing number of users.

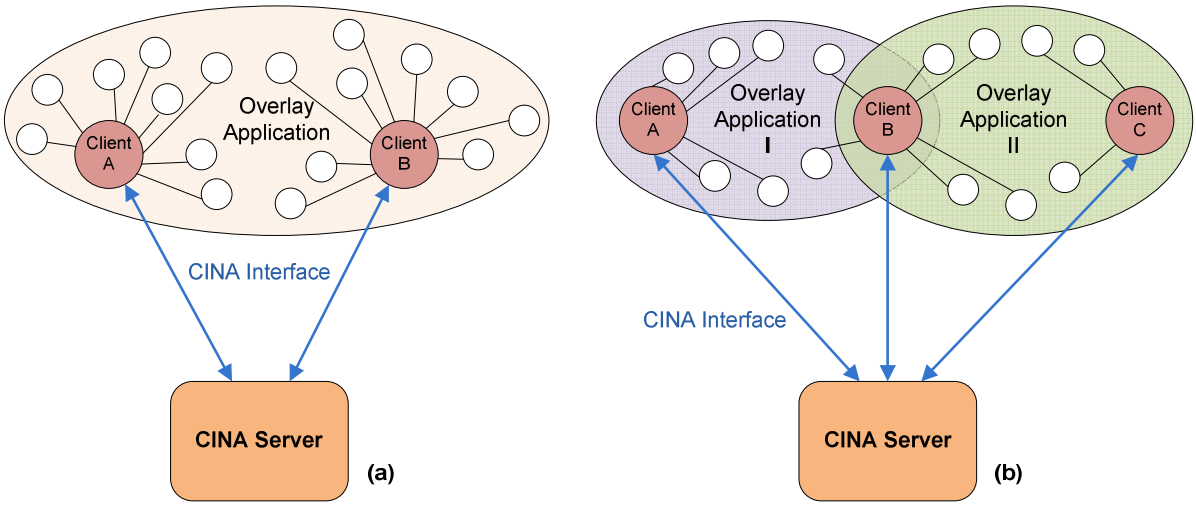


Figure 3. Collaboration context between overlays and networks

The security reasons include attacks against the client nodes and the preference of communicating with a hosted client, e.g. a client within the network, as the security services are easier to be deployed and monitored. It may be addressed through the replication of clients and the designation of clients hosted within the collaborating networks.

2.4.2.1.1 The CINA Server

The CINA server must be authenticated by the CINA clients in order to trust the information provided to the overlay application.

2.4.2.1.2 The overlay application

The overlay application must be authenticated for different reasons:

- Trusting the information provided by the application
- Determining the applications privileges, e.g. for controlling the application access to network information and services
- Sharing the resources and personalizing the services for the different applications
- Billing the offered services when the collaboration is ruled by a lucrative contractual agreement.

2.4.2.1.3 The CINA clients

The CINA clients are authenticated to address the following requirements:

- Determining the client privileges as the different clients dispose of different access levels to information and services
- Authenticating the application in the same time when the client is binded to the application

2.4.2.2 Collaboration Models

Depending on the specific model of collaboration between the applications and the networks, the different high level requirements could be addressed differently. Two main models of collaboration between applications and networks are retained to be investigated for the security process issues:

2.4.2.2.1 Application with a trusted entity

This model includes well-identified applications with at least one centralised entity, application centralised client, capable of ensuring a trusted collaboration with the networks. This entity is authenticated by the CINA servers in place of the application. It must never be a CINA client for another application.

The application is allowed using other nodes, e.g. super peers, as CINA clients to address scalability issues for example. In this case, each client is authenticated. Moreover a second authentication involving the application is carried after the client one. By this way, the CINA server is capable of determining the right privileges for the CINA client when representing a specific application.

2.4.2.2.2 Distributed applications

Distributed applications without a trusted entity are difficult to authenticate. Without a trusted entity, the application must carry the information relative to its identity, i.e. name, private key, certificate, etc. Hardcoding such information within the application could represent a solution to this issue. However, this solution presents many practical difficulties (information protection, update and revocation) and security breaches (no control over running platform). Considering these issues, distributed applications are subject to a limited scope of cooperation with networks. They are attributed a temporary identity as well as the CINA clients they are using in order to have a more efficient collaboration if there is a need to. The available information and services are however limited.

2.4.3 The security system specifications

Taking into consideration the clarification mentioned in the previous sections, a three party authentication between the CINA server, clients and the application is required for the collaboration. This authentication will be based on security certificates. Indeed, in opposition to login/password technique or a shared secret key, certificates X.509 [X509] and PGP [PGP] are managed by a key management systems providing creation, update, monitoring and revocation features. They are more appropriate for automated systems.

2.4.3.1 The Key Infrastructure system

The X.509 key management system based on a centralised PKI [X509PKI] offers security guarantees as the certificates are signed by a CA trusted by all the actors. The procedure of certificate creation is however subject to attention. Different cases arise within this context:

- Only public third-party PKIs: The servers and the (application, client) request a digital certificate from a business or public third-party PKI.

- Trusted third-party PKI with private PKI systems: The trusted third-party PKI will be requested to issue certificates to a small set of servers.
- Only private PKIs: The actors will have to trust CAs which identities are not confirmed by trusted CAs.

The first and third use case presents some major drawbacks of exploitation and viability. A better solution is the second use case where an association of public and private PKIs is used for allowing a good-enough trust level with a degree of scalability for huge number of (application,client).

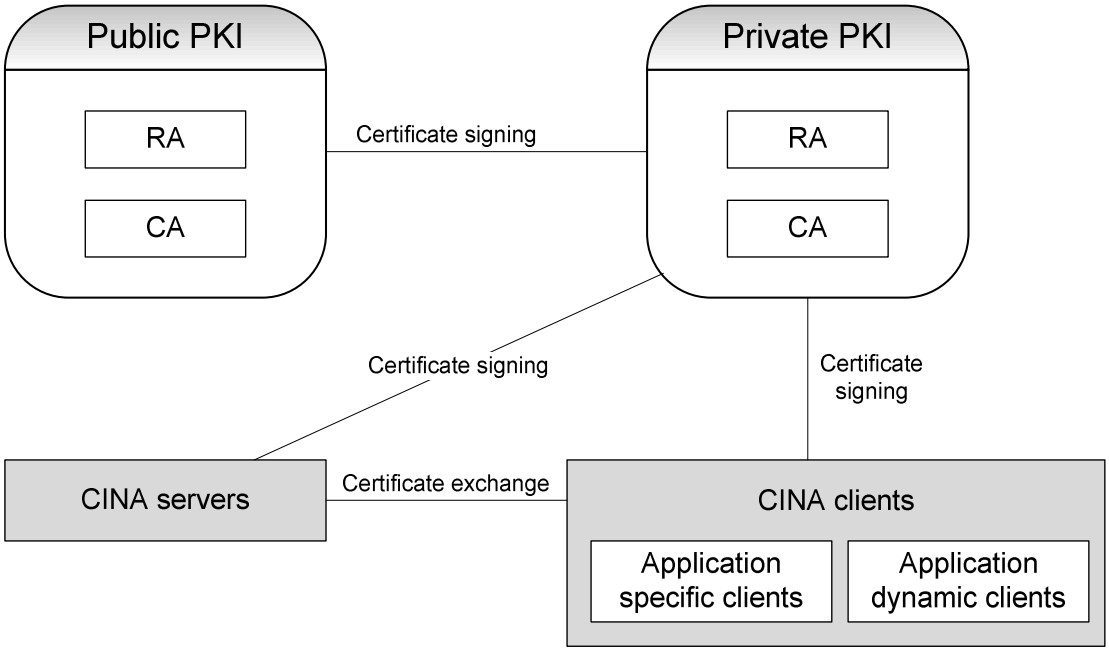


Figure 4. Public-private PKI system for the collaboration model

The public PKI signs the Private PKI certificate including its public key. The CINA servers and clients request from the private PKI system signing their certificates.

This model requires importing the public and private PKI systems certificates into each entity in order to check the integrity of the private PKI certificate first, then the integrity of the servers and clients certificates.

We recommend the following options depending on the targeted node:

- CINA servers and application centralised clients: root CAs certificate that signed the public CA in Figure 4 is carried off-line whenever it is possible. This should be reasonable for a small number of nodes even if the update procedure is manual.
- Big population of clients: we recommend using hardcoding to load root CAs certificates into the application. This possibility requires however security updates whenever a certificate is compromised. Otherwise, the on-line mode could be used but requires a vigilant intervention of the end-user.
-

2.4.3.1.1 Authentication

a) Server certificate generation

The CINA servers' certificates are generated by the Private CA. The certificate fields to be filled with the server identity information include:

- Subject: the most important field is the CN which must carry the server DNS name. An example of a subject field: C=Fr, L=Paris, O = France Telecom, OU= Orange Labs, CN=server1.cinaorange.fr
- Issuer: the private PKI
- Key: the server public key

b) Client certificate generation

This certificate allows the CINA server to identify which client and for which application the collaboration is requested. The certificates are generated by the private PKI system. The registration procedure however is different depending on the client type.

1. Application centralised clients: the procedure is similar to the server one. The difference resides in the way the application will be identified. To avoid the procedure of generating a certificate for the application independently from the clients, we will consider it as fully represented by its centralised clients. The registration procedure for these clients is done off-line to be fully trusted. To differentiate between the different applications and clients, the subject RDN fields must be carefully filled:
    a. Subject: The application is considered as an organisation. Each centralised client is attributed a unique overlay identifier or user identifier. An example of a subject field: C=Fr, L=Paris, O = ENVISION app1, CN=Centralised Client Alpha
2. Application dynamic clients: These clients could be familiar or unknown to the application. We will consider the scenario where these clients are dynamically chosen depending on their available resource to help managing the overlay application. The following steps (Figure 5) when executed lead to a certificate generation for a dynamic client:
    a. A centralised client requests from the dynamic client to start a TLS session.
    b. The dynamic client starts a TLS session and checks the centralised client identity.
    c. The centralised client starts on-line a registration procedure for generating a certificate for the dynamic client.
    d. The private RA/CA attributes a signed certificate to the dynamic client and sends it back to the centralised client.
    e. The centralised client sends the certificate along with the private key to the dynamic client using the secure TLS session.

The subject field of the certificate must contain the same application name in the Organisation RDN field as the one within the centralised client certificate. This name allows the CINA server to determine what application the client is requesting to represent.

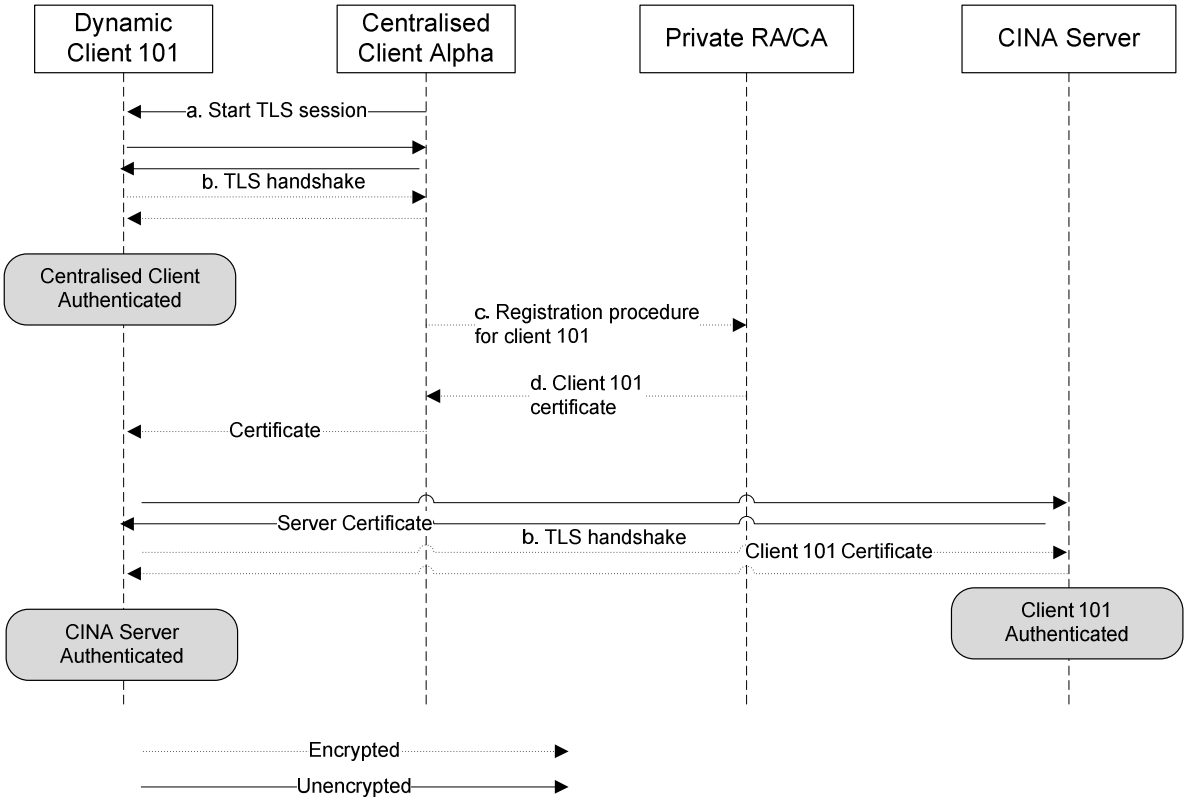


Figure 5. Dynamic client certificate generation and authentication message flow

2.4.4 The security protocol

This work is motivated by the following observations:

- The recommendations of the ALTO WG, protocol and security drafts [I-D.ietf-alto-protocol], [I-D.ietf-alto-reqs], to use the TLS protocol for securing the communication between the ALTO clients and servers
- The CINA interface is based on the same framework as the ALTO protocol
- The TLS protocol is easier to deploy and supported on almost all platforms for application level services

The TLS protocol is used for securing the CINA interface and providing the authentication, data confidentiality and integrity protection services.

The IPsec is investigated to be used for securing special interfaces operating at the network level. Such sessions could be for instance the session between a content source and a multicaster where the traffic is processed at the network level at the multicaster. In this case using IPsec presents a huge saving in terms of processing delay.

2.5 Design and Implementation of the CINA interface

The CINA interface allows client applications to be informed about the network status and vice versa and to access special services in an ISP (for example Multicast). This section gives an overview of the implementation of the protocol in two distinct modules, one server side and one client side, as shown in Figure 6. The two CINA parts, called CINA Server/Client library, communicate via the CINA protocol as specified in section 2.3. Client Applications, such as the Push-based IVCD (Interactive

Video Content Distribution) or the Pull-based LVCD (Live Video Content Distribution) (see D4.2), can integrate the CINA Client Library to access CINA server providing network information and to make use of the special functions they support. On the server side, the library parses the CINA messages and forwards them to the respective modules. For example the information base that contains results of network monitoring (see section 3), or modules that control CINA network services (see section 4).

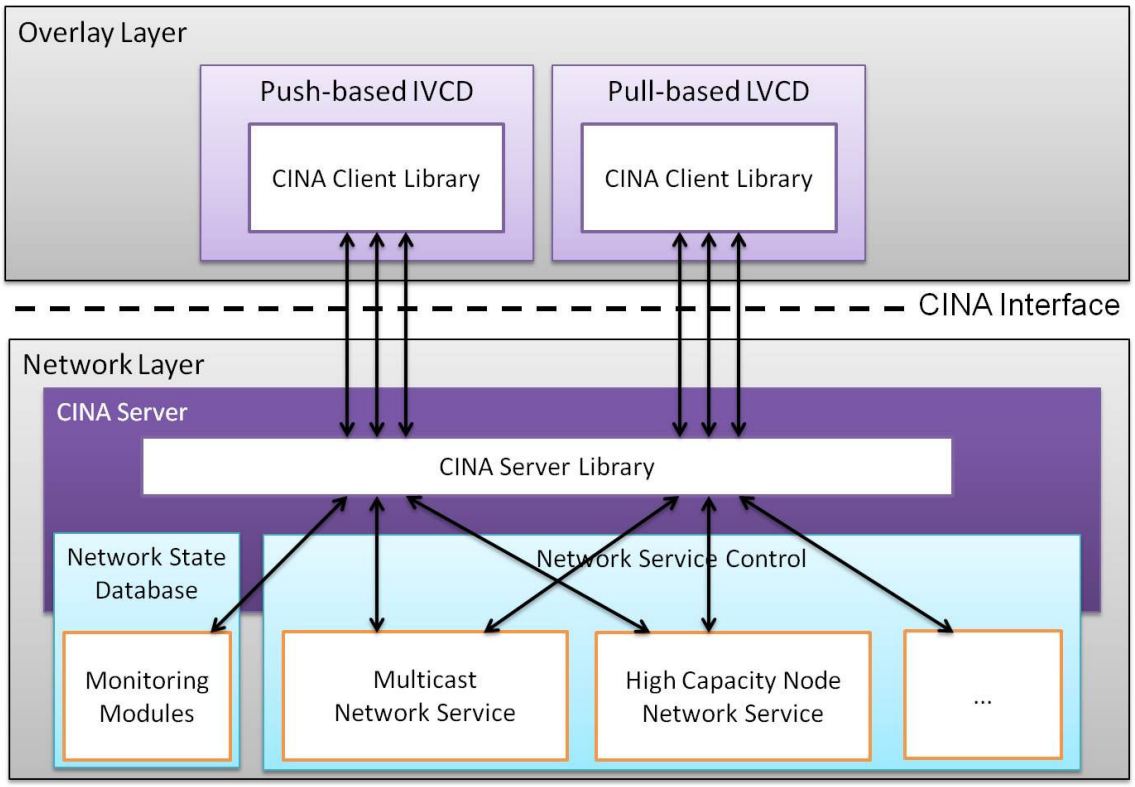


Figure 6. CINA architecture overview

The CINA Server and Client Library communicate with each other through HTTP requests and responses. Along with the HTTP/1.1 protocol, the JSON format is also used for the request and response messages.

2.5.1 Client Side Overview

The CINA Client Library can be integrated into overlay applications to perform the following tasks:

- Discovering a CINA Server and providing the user with the services supported by the server
- Forming and making a new request to the server
- Accepting and parsing the server response
- Storing the information received to appropriate variables
- Providing the user Application with the information requested

More specifically, the CINA Client Library discovers a CINA Server that the client Application can connect to. At the same time, it requests the list of the server supported services and their details, which are then rendered accessible to the client Application. Every time the Application needs to

activate a new CINA service, it consults the supported services list to check whether the specific service is available. If it is, the Application activates the CINA Client Library request mechanism. The latter makes the respective request to the CINA Server and receives the response in the form of a JSON file. After that, the response file is parsed and the information is stored in appropriate variables, accessible to the client Application. In other words, CINA Client Library is the mediator between the client Application and the ISP. It undertakes the task of discovering a CINA Server and communicating with it every time the user Application wants to use the special CINA functions.

2.5.2 Server Side Overview

The CINA Client Library lies in the client side (overlay applications) to enable the communication between the Application and the ISP. Consequently, there is also a corresponding CINA part implemented on the server side (ISPs) which processes client requests and responds with the result. The CINA Server Library can be included in scripts of standard webservers, such as Apache or lighhttpd, to receive, parse and reply to CINA requests, coming from the CINA Client Library. The library parses the requests along with its headers (and its parameters if it is a POST request) and creates CINA formatted responses. Each CINA Server may support all or just a portion of the CINA functions, for example not all CINA operators will support equal network services or the same network information cost types. A basic service that is supported in any case is the Information Directory that contains the methods that are available on the respective server and the URIs that a client has to use in order to invoke them.

2.5.3 Basic Communication Flow Example

One basic example that depicts the communication that is occurring between an application, the CINA client library through its API and the CINA server is illustrated in Figure 7. First, the user Application initializes the CINA Client Library. After performing the discovery process (which is not shown here) the CINA Client Library creates a connection to the CINA Server and performs one or multiple Information Directory Requests. The CINA Server sends the Information Directory response, informing the CINA Library about the supported CINA Services. The CINA Client Library makes the supported services list accessible to the user Application and then, waits for a new CINA Service request. As indicated in Figure 7, every time the client Application wants to make use of a CINA Service (e.g. Network Map, Latency Cost Map), it notifies the CINA Client Library through an API call. The CINA Client Library, performs the corresponding request to the CINA Server and then receives the response, processes it and sends the result back to the client Application.

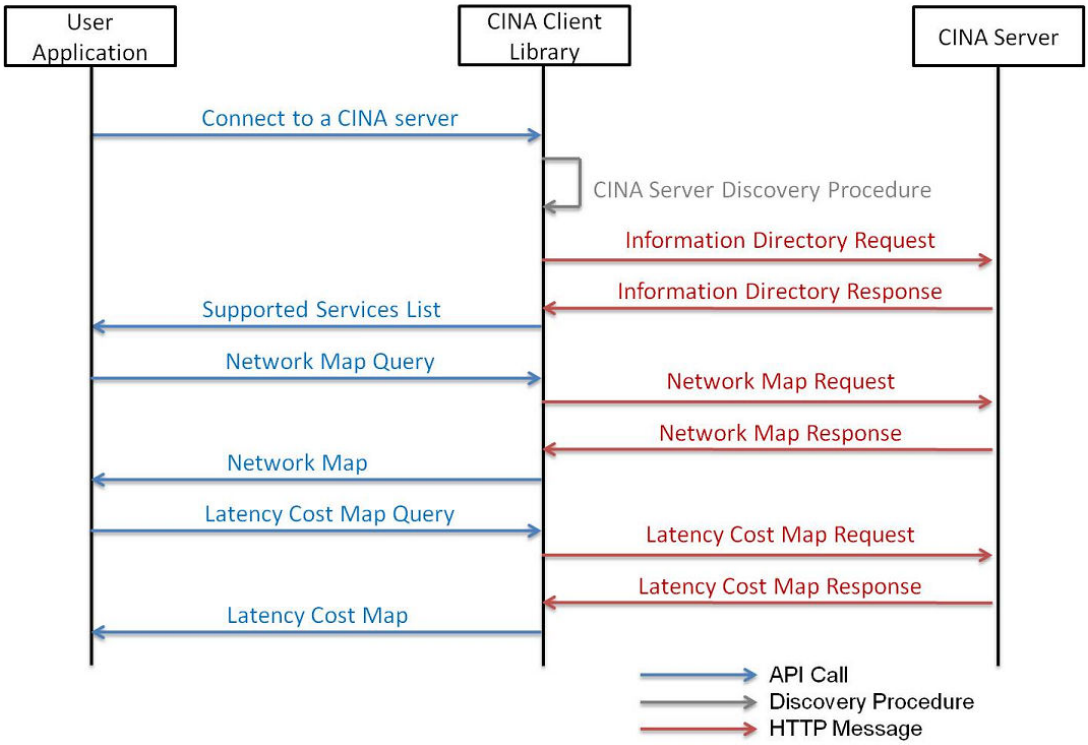


Figure 7. Basic communication flow example

3. MONITORING

3.1 Problem Statement

In ENVISION, the monitoring part is critical as it gathers the performance information and provide it to the overlay application. The first work of this study was to evaluate which network metrics the ISP should monitor and which of them can be useful for internal use by the ISP itself and for external use by the overlay applications. This information is provided using the CINA primitives as defined in section 2. The application could then benefit from this information to improve the delivery of traffic.

In [D3.1], we analyzed the different technologies and tools that could be used for monitoring. After having tested and evaluated some of them (Cacti and MRTG), we decided to go for our own monitoring process. Indeed the existing tools can collect values, using the SNMP protocol, from the equipment but collecting other information, not present in the equipment's MIB was not possible. For instance, it was not possible to collect the network topology (routers connected to others routers, the cost of the links, the latency, the subnets attached to the routers, etc.). Then instead of having a huge development work to adapt such tools (make plug-ins or extensions, integrate them into the core tool etc.) , we preferred to develop our own light-weight application but more focused on our specific needs.

In [D3.1], we identified several network metrics. For the implementation, we only kept the most useful network metrics (useful to the application and network perspective. Indeed, some as defined by IPPM [IPPM] and NM-WG [NMWG] were not really suited to our needs, because we need some topological information to map some network (e.g. delay, bandwidth) information to the topology information. The topological information we get is directly collected from the routers themselves via direct connection to them.

We decided to monitor the following metrics:

1) Load: State of the use of an equipment (CPU, memory)
2) Connectivity: State of a peer (up/down).
3) Bandwidth: Bit rate measurement of available data communication resources (link, interface, etc.)
4) Latency: Time from the start of packet transmission to the start of packet reception (one-way latency).
5) Topology: Layout pattern of interconnections of the various elements of the network in one IPS or more and between them.
6) Connection Hop: Number of hops between the peers. It can be used with the network topology.
7) Packet Loss: Number of lost packets per interface

The objective of this task was to define a monitoring architecture and the technological choices that could allow us to monitor the above mentioned metrics and be able to provide abstracted network information when requested by the application through the CINA interface. The following section presents the main outcomes of this work.

3.1.1 Monitoring architecture

Since the existing monitoring systems were too complex or do not provide the functionalities we require, we decided to design and implement our own light-weight monitoring system. The overall monitoring architecture is shown in Figure 8.

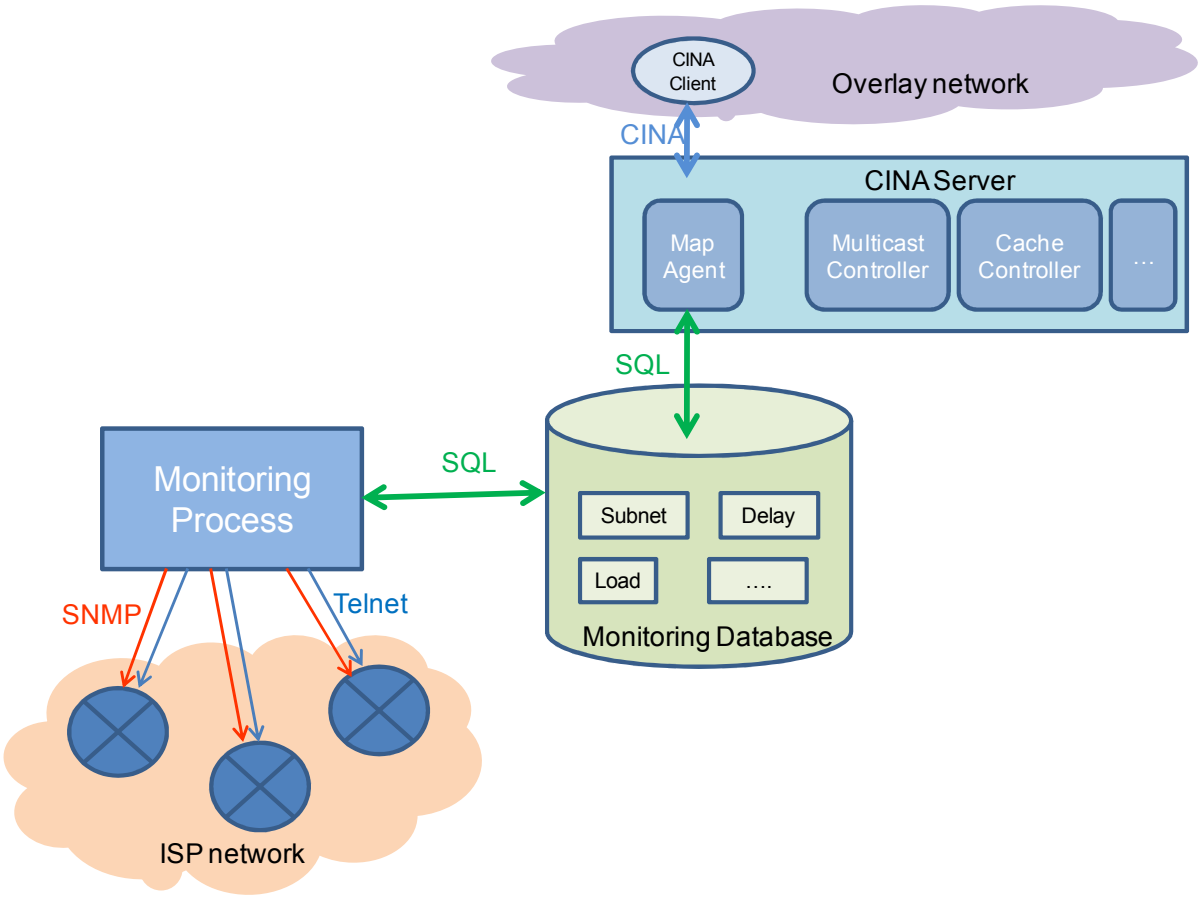


Figure 8. Overall monitoring architecture

In the project, we assumed that every ISP will host its own monitoring platform, to monitor only its own network (no relationship between different ISPs). In the same way, every ISP will have one CINA server, which the application should communicate with. The exchange of information between the two components is done by a shared database that will be filled in by the monitoring process and read by the Map Agent module, which is one module located in the CINA server.

The overall call flow for data exchange between the overlay applications and the ISP is then:

1) The monitoring process collects information from the network, by communicating with the routers using the SNMP protocol [SNMP] for information in the equipment's MIB or using CLI commands for other information (such as the neighbours, the delay between routers, etc.)
2) The monitoring process writes the collected values in the appropriate table of the monitoring database.
3) When receiving a CINA request from the overlay application, the Map Agent will read the necessary values (according to the given request and provided useful metric for the application).
4) The Map Agent builds the map to provide to the applications, depending on the request and level of granularity agreed between the two actors
5) The Map Agent replies to the CINA request with the application-specific map as body of the response.

For scalability and performance issues, the Map Agent could also store locally for a given duration the information retrieved from the database or even the built maps (in order to reply faster to the request). It is an implementation option, which could be useful in case of deployment of the CINA server for many applications.

For the implementation, we selected open source products, that are widely used and that have been proved as performing, reliable and easy to use.

The database is implemented using MySQL.

The protocol to collect classical values (being stored in MIB's equipment) is SNMP and extra information was collected via a direct connection to the routers, using CLI commands. Since in our testbed, not all routers accepted SSH connections yet, we had to come back to a Telnet connection, accepted by all equipment.

The monitoring process is implemented in Python, with the necessary libraries to use with MySQL, SNMP and Telnet.

In the database, we created six tables for storing network information. These tables and their associated parameters are:

- *status*: { router varchar(20) primary key, adress varchar(20), status varchar(10), cpu int, memory double, constructor varchar(20) }

  This table contains information about the network equipment to monitor

- *metrics* : { routerS varchar(40), routerD varchar(40), cost int, interface varchar(20), via varchar(20), ping float, hop float, bandwidth double, error double, PRIMARY KEY (routerS,routerD,interface) }

  This table contains metrics values between routers (e.g., based on IS-IS data)

- *neighbors* : { router varchar(20), routerDC varchar(20), interface varchar(20), PRIMARY KEY (router,routerDC, interface) }

This table contains metrics related to the network topology (e.g., the neighbor routers)

- *subnets* : { router varchar(20), subnet varchar(20), interface varchar(40), PRIMARY KEY (router,subnet) }

This table contains the network subnets connected to the router

- *delays* : {routerS varchar(40), routerD varchar(40), ping float, hop float,PRIMARY KEY (routerS,routerD) }

This table contains metrics related to the delay, the number of hops between routers (collected via CLI commands)

- *isp*: { ispS varchar(40), ispD varchar(40), ping float, hop float, PRIMARY KEY (ispS,ispD) }

This table contains metrics related to the connectivity to other ISPs

The following figure 9 gives an example of the table delays, showing the round trip delay between two routers.

```
mysql> select * from delays;
+------------------+------------------+--------+------+
| routerS          | routerD          | ping   | hop  |
+------------------+------------------+--------+------+
| NrNis202.re1     | NrNis201.re0     |   0.36 |    0 |
| NrNis202.re1     | NCCIR105.re0-WE15 |  0.447 |    1 |
| NRISSY201.re0    | NrNis201.re0     |  12.06 |    1 |
| NrNis201.re0     | NCCIR105.re0-WE15 |  0.398 |    0 |
| NrNis202.re1     | NCCir106.re0-WE15 |  0.401 |    0 |
| NRISSY202.re0    | NrNis201.re0     |  9.131 |    2 |
| NrNis201.re0     | NCCir106.re0-WE15 |  0.461 |    1 |
| NrNis202.re1     | NCNIS207.re0     |  0.536 |    1 |
| NrNis201.re0     | NrNis202.re1     |  0.344 |    0 |
```

Figure 9. Example of the delays table

For collecting information from the routers, the monitoring process follows these steps:

1. Connectivity tests towards routers; in order to know the status of the router (up or down)
2. Collection of the router metrics (e.g., CPU, Memory)
3. Connection to every router to collect network related information (topology, attached subnets, latency, hops, etc.)
4. Processes retrieved messages to keep only useful information
5. Store metrics values in the database (based on described tables)

The functional tests of the monitoring process have been performed in the Orange testbed (see D6.1), composed of various pieces of equipment from different vendors. This enabled us to highlight that the same command cannot be applied to every vendor since each one has its own specificities. Similarly when using SNMP, the OID is often different and we then have to take into account the name of the equipment's vendor before collecting information from the equipment (and thus select the appropriate command).

For testing purposes, we developed a simple home-made module for the Map Agent component and performed some tests, simulating applications CINA requests (e.g. request a delay map or the network map). This module should now be replaced by the real CINA server implementation (the Map Agent adapted to be embedded in the CINA server). Later, we will be able to have the whole chain including the application.

4. NETWORK OPTIMISATION

4.1 Introduction

In [D3.1], we listed several network services that a network operator can deploy in its network and offer via the CINA interface. Amongst this list, we selected 3 services that are the most promising in terms of network load reduction for the ISP and of improvement of QoE for the overlay application. Those 3 services are the dynamic setup of a multicast service when requested by the applications, the use of caching nodes, deployed by the ISP, when needed by the application and the use of a high capacity node, located in the network, which help to improve the delivery of contents. The following sections present those 3 network services, their design and the technical choices that have been done for their implementation.

In the project, we are also exploring the benefits a network operator may achieve by setting the costs of overlay connections announced through CINA with the optimisation criterion of reducing its operational costs, and in particular the transit costs paid to its transit and access providers. While in ALTO it is assumed that ISPs and overlay applications have a mutual benefit from localising traffic in all cases, in ENVISION we are exploring a wider range of scenarios where an ISP may need to reduce traffic on its access network rather than the inter-domain traffic. Because of the pricing model between providers in the Internet today which uses the 95th percentile of the actual traffic to determine the final price, the traffic distribution over time is also an important factor. A mathematical model for setting the costs is presented at the end of this section, while some preliminary evaluation results can be found in deliverable D6.1.

4.2 Multicast

4.2.1 Solution for offering multicast services

In ENVISION, we aim to use multicast services along overlay distribution in order to enhance the delivery of adaptable content to different users, and to reduce ISP costs. As described in the first deliverable [D3.1], the architecture chosen for the multicast network service relies on the use of specific nodes in the network, called “multicasters”, whose function is to receive the traffic from overlay applications peer sources in unicast and translate it in multicast. As explained in [D3.1], this approach allows the ISP to control the multicast traffic in its network without requiring dynamic access control functions on the network access equipment. As shown on Figure 10, the multicasters are managed by a multicast controller integrated in the CINA server. The multicast controller has two functions: it manages the allocation of multicast resources as they are requested by the applications, and configures the multicasters according to the multicast translations which must be set up. In the case of SVC content delivery over P2P networks, each SVC layer may be sent on a separate multicast group. Consequently, the peers can switch from overlay delivery to multicast delivery and so, subscribe to multiple multicast groups according to their context (network conditions, desired quality, etc.).

Figure 10 shows the multicast network service architecture on an example. In this example an overlay P2P application runs on top of 3 Autonomous Systems (AS), managed by 3 ISPs (1, 2 and 3). The 3rd ISP offers the multicast network service in AS3, so that the P2P network can benefit of this service when necessary.

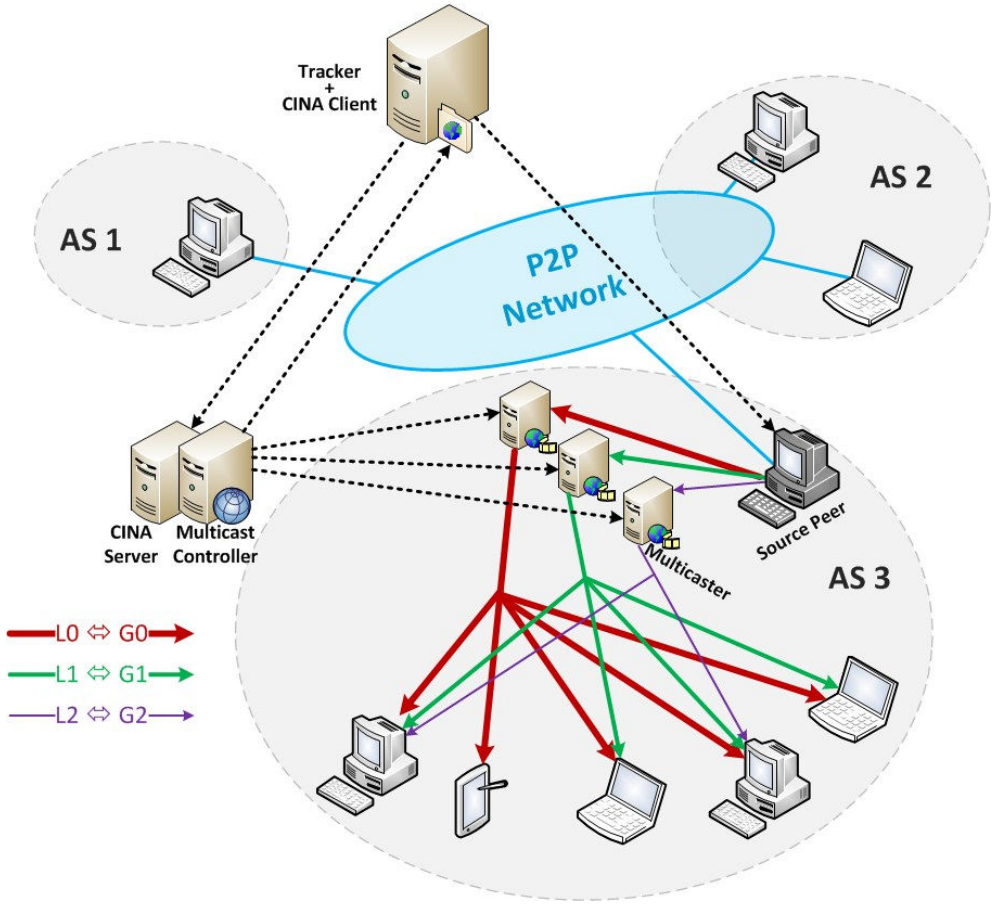


Figure 10. Example of a multicast network service architecture, with SVC stream that is composed of 3 layers (a base layer L0, and two enhancement layers L1 and L2)

Figure 11 gives an example of call flow showing how the multicast service can be invoked by the application through the CINA interface, for instance when detecting that there is a lot of receivers in an ISP domain (or detecting that the available uplink bandwidth is insufficient for the target quality). Using the multicast service offered by the ISP3 would enhance the users QoE and reduce the burden on the network.

In a first step, the CINA client (e.g. tracker of the P2P system) has discovered the ISP CINA server and started a secure session with the server by executing security procedures, i.e. authentication, data encryption and signing, etc (cf section 2.4). Then, the client can obtain the Network Map and the Cost Map (1 and 2). Regularly, the CINA applications send their Footprint and Constraint Maps to allow the CINA server to maintain a (partial) view of the current and/or forecast traffic (3 and 4). At some time the P2P tracker detects that the number of clients is growing and exceeds its multicast switching threshold. It requests the Network Services Map from the ISP (5). The server replies with the map (6) confirming that a multicast service is available in its domain. The tracker decides to request the multicast service (one or several instances of the service in case of SVC stream) using the network service instantiation service and chooses one node[1] (or several nodes in the case of SVC where several quality levels can be multicasted) to fulfil the function of stream source (7). The multicast controller within the CINA server checks the availability of the requested resources, it can also refer to the traffic map, and reserves the required resources (couple of (source, group) multicast

[1] the source node has to be carefully chosen to have sufficient downlink to receive all the layers from the overlay networks and, respectively, to have sufficient uplink to send all the layers in unicast to the multicasters

addresses, port on the multicaster etc.). It then initiates a configuration procedure toward the multicaster (s) (one or several instances in case of SVC streams) (8). The server confirms the multicast instantiation by sending the multicast group addresses and the multicaster(s) unicast address and port number to the client (9). After reception of the confirmation, the tracker requests to the source(s) to stream the content to the multicaster(s) (10). The elected source(s) starts the streaming (11) and reports back the execution to the tracker (12). At this stage, the tracker requests from the P2P nodes located within the 3rd ISP domain to switch to multicast providing them the required parameters (each of the peers subscribes to the multicast groups according to the desired quality level and network conditions: e.g., a peer requesting quality 2 for example has then to subscribe the multicast groups (G0, G1, and G2) in order to receive the layers 0, 1 and 2 respectively) (13). The nodes report back the good reception of the multicast stream (14).

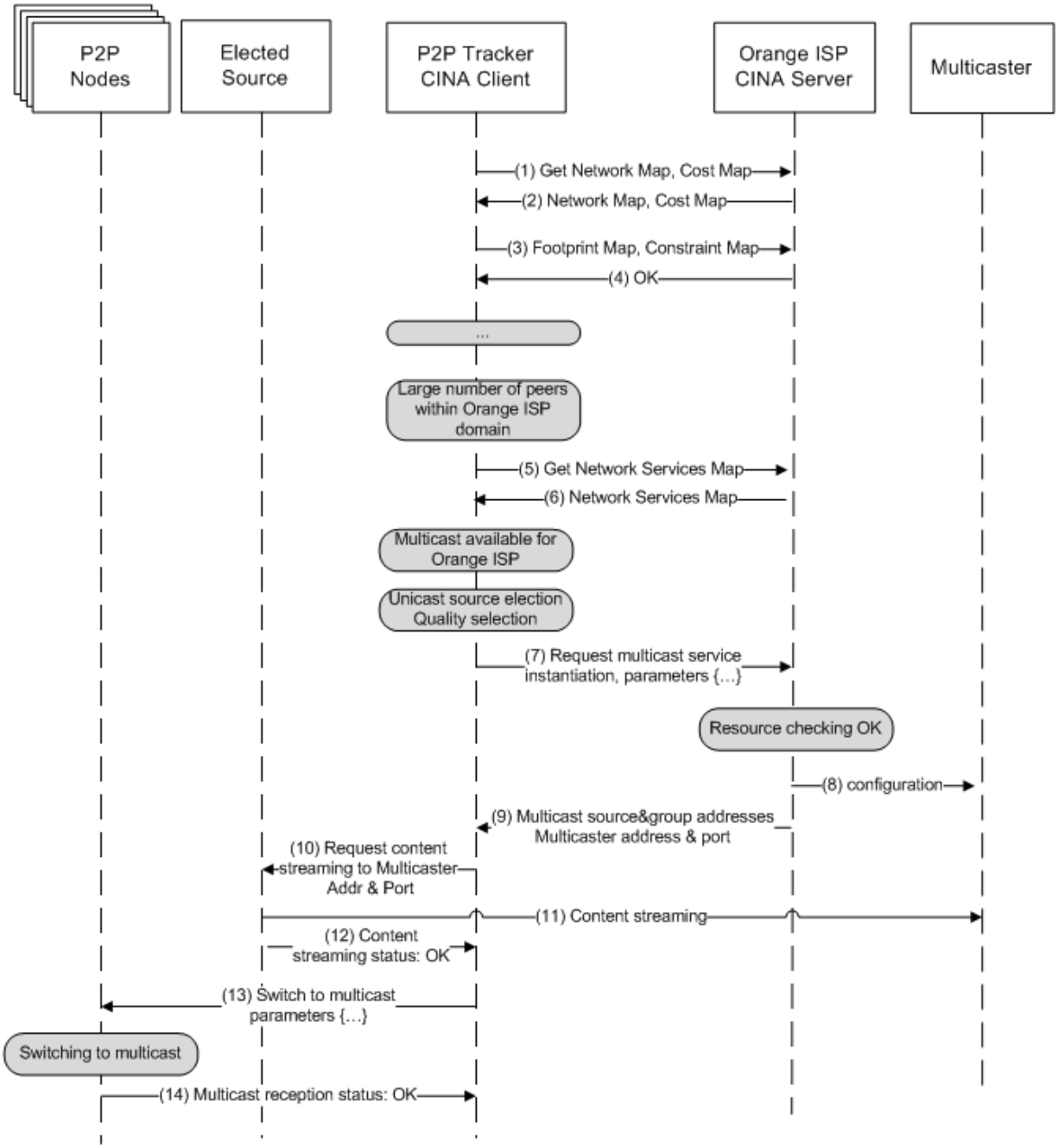


Figure 11. Example of multicast service use call flow

4.2.2 Design and Implementation of the multicast service

4.2.2.1 Multicast service building blocks

The multicast service is composed of:

- The multicast controller, developed in Java, which will be integrated in the CINA server. The methods exposed by the multicast controller to the applications (through the CINA interface) are described in section 4.2.2.3., with a description of the ongoing implementation.
- One or several multicasters, implemented on top of the Linux kernel as described in section 4.2.2.2.

Figure 12 describes how the different building blocks interact. The multicasters are configured by the multicast controller through secure SSH connections.

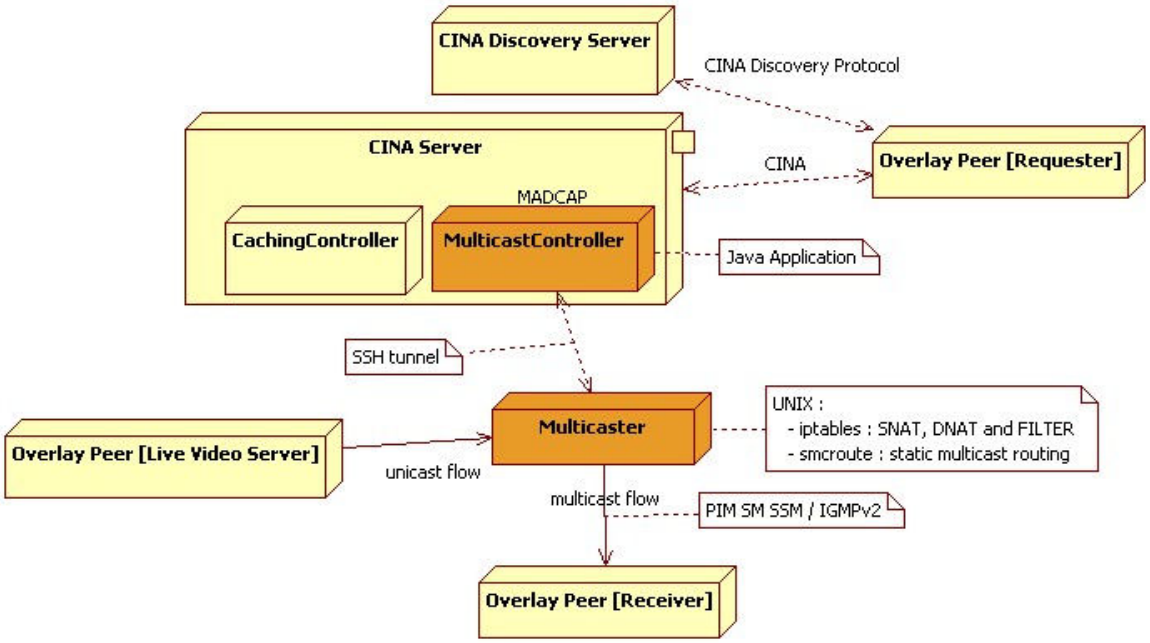


Figure 12. Multicast network service building blocks

4.2.2.2 Multicaster implementation on top of the Linux kernel

Several options have been investigated for the implementation of the multicaster in the network:

- Relying on network address translation (NAT) functions integrated on routers, and on the use of the Network Configuration Protocol (Netconf) for instance, to set up the required configurations. This option has been excluded since routers do not authorize address translation toward a multicast group address
- Developing the required NAT function on open routers (using the Juniper SDK for instance, or some implementation of openflow). This option has been excluded also since it requires specific licenses and equipments.
- Relying on the Linux kernel. This option has been chosen because it seems to be more open and easier to deploy, even if on the contrary it has a relatively low scalability (at least with the servers used).

The chosen approach implies the following assumptions:

- There is no need to rely on the application level (even for security purposes)
- Multicast flows are received in UDP

Implementing the multicaster function on top the Linux kernel requires to good understanding of how the packets are forwarded in the Linux kernel. Moreover the Linux kernel must be extended to deal with routing of multicast traffic. Several alternatives have been studied, in particular (dynamic) multicast routing daemons like *pimd* and *mrouted*. These solutions have been proved to be unsuitable. Indeed, they can route multicast streams, but they are not adapted to the case where the server acts as the multicast source. For the same reason, we excluded the Over software routers solutions which implement PIM-SM, like *Xorp* or Q*uagga*. We have chosen to use *smcroute* which is a static multicast router daemon for Linux. Iptables and smcroute allow to statically configure which multicast traffic is sent on which interface, this is simple but sufficient in the multicaster case where we only need to forward the produced multicast traffic on a predefined "output" interface.

Figure 13 describes the different steps where Linux commands can be applied to IP packets, and illustrates the required actions to translate a unicast traffic (source = 11.0.0.2, destination = 13.0.0.2) to a multicast traffic (source = 13.0.0.2, destination = 232.0.0.1).

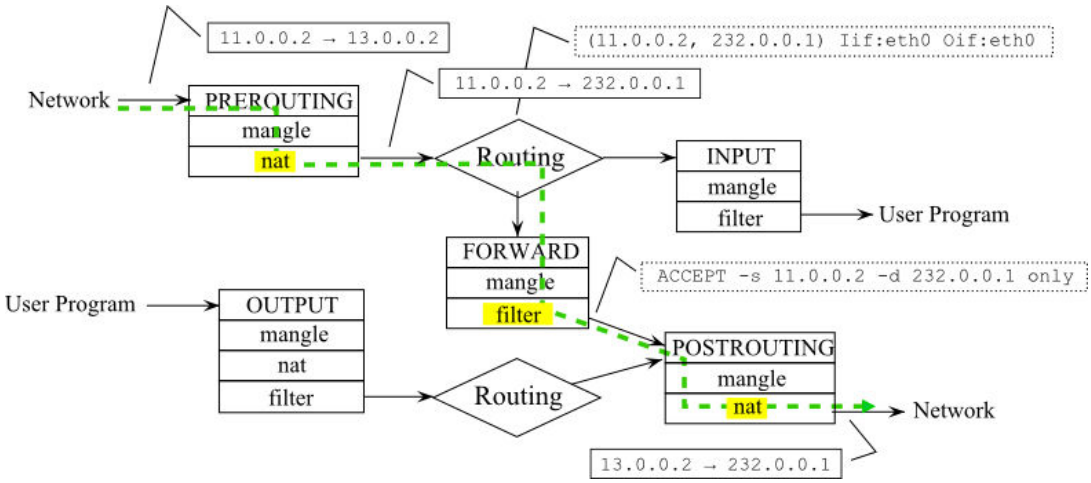


Figure 13. Different steps for "natting" IP packets in the Linux kernel

More precisely, the following commands must be sent to the Linux server by the multicast controller (via SSH) when the following translation must be done:

(<unicast_source_address>:<source_port>,<multicaster_IP_address>:<multicaster_destination_port>) → (<multicast_source_address>:<source_port>, <multicast_group_address>:<multicast_port>

- Twice NAT (Destination NAT + Source NAT) with iptables

A unicast → multicast translation is added with the following commands:

iptables -t nat ***-A*** *PREROUTING -s <unicast_source_address> -d <multicaster_IP_address> --source-port <source_port> --destination-port <multicaster_destination_port> -j DNAT --to-destination <multicast_group_address>:<multicast_port>*

iptables -t nat ***-A*** *POSTROUTING -s <unicast_source_address> -d <multicast_group_address> --source-port <source_port> --destination-port <multicast_port> -j SNAT --to-source <multicast_source_address>*

A unicast → multicast translation is deleted by using the same commands and replacing option *–A* with *–D*.

- Static multicast routing with smcroute:

The following command adds the required multicast route when a translation is added, *<input_itf>* is the interface on which unicast streams are received and *<output_itf>* is the interface on which multicast streams are forwarded:

smcroute -a <input_itf> <unicast_source_address> <multicast_group_address> <output_itf>

A musticast route is deleted by replacing the option *–a* by *–r*.

- Filtering with iptables to allow only the forwarding of the traffic from allowed sources:

By default all the traffic is dropped using the following command:

iptables -P FORWARD DROP

A new unicast stream is allowed with the following command:

Iptables -t filter -A FORWARD -i <input_itf> -s <unicast_source_address> -d <multicast_group_address> -p <transport_protocol> --source-port <source_port> --destination-port <multicast_port> -j ACCEPT

A filtering rule is deleted by replacing the option *–A* by *–D* in the preceding command.

- Rate-limiting functions (still to be specified)

4.2.2.3 Multicast controller

4.2.2.3.1 Multicast Service interface

The multicast service interface has been defined by relying on the two following standardized API:

- Multicast Address Dynamic Client Allocation Protocol (MADCAP) – RFC 2730 [RFC2730]
- An Abstract API for Multicast Address Allocation – RFC 2771 [RFC2771]

Which have been adapted to fit to the use of the multicaster.

The main method MulticastAllocateRequest allows the reservation of a multicast resource with the following parameters:

Input	**Output**
• MulticastAddressFamily: IPv4 or IPv6 • TimeRange startLeaseTime: time from which the multicast resource is requested, under the form [min, max]. When omitted the server assumes the multicast resources are requested immediately. • TimeRange endLeaseTime: time to which the multicast resource is requested, under the form [min, max]. When omitted the default lease duration announced by the server is used to calculate the endLeaseTime.	• String leaseIdentifier: transaction ID • Set MulticasterAddressSet: to each triplet [source-address, source-port, multicast-port] is associated a pair [multicaster-address, multicaster-port] • InetAddress multicastGroupAddress: the multicast group address allocated • Date startLeaseTime • Date endLeaseTime • Date currentTime: used for approximate

• Date currentTime: used for approximate time synchronization • Set SourceSet: the sources which will send the streams to multicast. Each source is defined by 4 data: [source-address, source-port, multicast-port, max-bit-rate] • long forcastedUserNumber	synchronization • Status status: method execution status

The method MulticastChangeLeaseTime allows the modification of the multicast resource lease time:

Input	**Output**
• String leaseIdentifier: transaction ID • TimeRange startLeaseTime: time from which the multicast resource is now requested, under the form [min, max]. When omitted the server assumes the multicast resources are requested immediately. • TimeRange endLeaseTime: time to which the multicast resource is now requested, under the form [min, max]. When omitted the default lease duration announced by the server is used to calculate the endLeaseTime. • Date currentTime: used for approximate synchronization	• String leaseIdentifier: transaction ID • Date startLeaseTime • Date endLeaseTime • Date currentTime: used for approximate synchronization • Status status

The method MulticastAddSourceAddress allows to add sources sending unicast traffic towards an already allocated multicast group:

Input	**Output**
• String leaseIdentifier: transaction ID • TimeRange startTime: time from which the new sources will send traffic [min, max]. When omitted the server assumes that the sources will send traffic immediately. • Date currentTime: used for approximate synchronization • Set SourceSet: the sources which will send the streams to multicast. Each source is defined by 4 data: [source-address, source-port, multicast-port, max-bit-rate]	• String leaseIdentifier: transaction ID • Set MulticasterAddressSet: to each triplet [source-address, source-port, multicast-port] is associated a pair [multicaster-address, multicaster-port] • Date startTime • Date currentTime: used for approximate synchronization • Status status

The method MulticastRemoveSourceAddress allows removing sources sending unicast traffic towards an already allocated multicast group:

Input	Output
• String leaseIdentifier: transaction ID • Date startTime: time from which the sources will stop sending traffic. When omitted the server assumes that the sources will stop sending traffic immediately. • Date currentTime: used for approximate synchronization • Set SourceSet: the sources which will stop sending traffic. Each source is defined by 3 data: [source-address, source-port, multicast-port]	• String leaseIdentifier: transaction ID • Status status

4.2.2.3.2 Multicast Controller Implementation (ongoing work):

The Multicast controller manages a set of multicasters, each multicaster being associated to a set of multicast resources, which correspond to a couple of multicast source and group addresses.

The main Java classes used to implement the multicaster are represented on Figure 14.

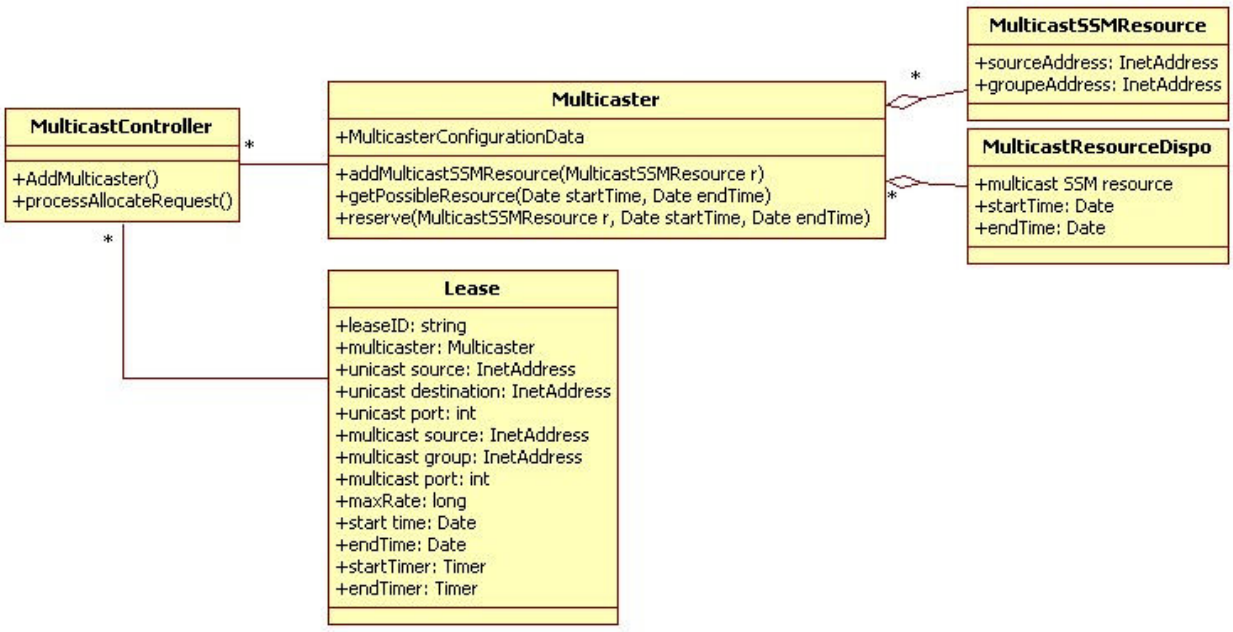


Figure 14. Multicast controller implementation

Each object "Multicaster" contains information about the associated multicaster configuration data (user ID and password to be used for the SSH connection, name and IP addresses of input and output interfaces for instance), of the set of associated multicast resources and of a list of available time intervals for each multicast resource.

When the multicast controller receives a multicast allocation request, it obtains the multicast resources available on the required time interval from the multicasters, chooses on of these resources, updates its list of leases and answers to the application.

Each object “Lease” contains the required information to configure the associated multicaster and is associated to two Java Timers. At the time associated with the *startTimer* an action is triggered which establishes a SSH connection to send the commands for the set up of the unicast-to-multicast translation; at the time associated with the *endTimer* a SSH connection is set up to send the commands deleting the former translation.

Please, note that the multicast controller implementation is an ongoing work.

4.3 Caching

4.3.1 Solution for offering caching services

Historically caching solutions implemented in ISP networks (or private networks as enterprise, university, etc) were to provide these ISPs maximum savings on used bandwidth. First of all, the saving is based on interconnection link costs. With the development of internal network, the location of cache solution near end-users appears also as an important factor to save bandwidth in core network. The second effect of cache presence inside a network is to provide quality of experience to the end-users. This quality of experience is mainly the acceleration of download with its consequences, for example on video visualization exempt of freeze and visible buffering.

Using caching solutions in the ENVISION project answers to the two precedent points, saving and QoE. ISPs can provide to applications cache capabilities for its own interest (bandwidth saving), its customer interests (QoE) or make content providers partnership easier.

Technically, depending on the solution implemented in ISP network, the cache disposal may be possible or not, following the communications protocols used by ENVISION and supported by ISP caching solutions.

On other way, ISPs may wish to give cache service to specific traffic or specific content providers following partnership presence or not.

The idea is to promote caching solutions to help the content delivery made towards ENVISION network, overtaking bandwidth saving aspects to keep the best possible quality for customers.

4.3.2 Design and Implementation of the cache

4.3.2.1 Caching service architecture

The caching solution is based on ENVISION nodes provided by LaBRI. This choice was made to avoid sophisticated software development for one side and avoid usage of existing commercial solutions which would require adaptations to ENVISION protocols. The ENVISION protocol is based on P2P solution using peers also known as seeder when the peer has complete content and leecher when the peer requests content or pieces of content. The ENVISION overlay network is completed with trackers in charge of keeping knowledge of overlay network i.e. connected peers. The tracker is identified in torrent-like file, representing data to retrieve content. This choice has the consequences of adding some specific modules to transform ENVISION peer to ENVISION caches. These modules are in charge of retrieving new torrent from ENVISION tracker. ENVISION peer used for cache will be also completed by content management module.

In order to use the ENVISION specific nodes for caches, it is necessary to make the choice of cache ingestion mechanism. Cache is set of independent ENVISION peers. To ingest a content, cache can pull content from the others ENVISION peers or content can to be pushed inside the cache. With ENVISION mechanism, if the cache doesn't know the content ("torrent"), it will not be able to get the content. Therfore, the cache needs to get the torrent of content to be able to request the content as

soon as possible to the seeder. This means the ENVISION tracker which is aware of new uploaded content with seeder information needs to be aware of cache presence to communicate this data. The ENVISION tracker is ISP independent. The cache presence and other information must be communicated to tracker through CINA.

Cache architecture needs to implement these different modules:

- Content ingestion
- Content delivery
- Content management
- Monitoring/Reporting

All these modules are detailed in ENVISION D5.2.

Call flows for discovery and acquisition of new content by ENVISION cache based on initial content provision by end-user source can be described in figure 15.

When end-user delivers a new content, it creates a torrent to tracker and becomes a seeder. Thanks to CINA, tracker can receive confirmation of ISP cache presence. After communication between tracker and cache the torrent is transferred to cache. This last one as classical leecher is able to retrieve content from a seeder, so that the cache can get new content very quickly after the torrent creation. In addition, there is also possibility to transfer request between different caches if it is allowed. So, ENVISION caches get content independently.

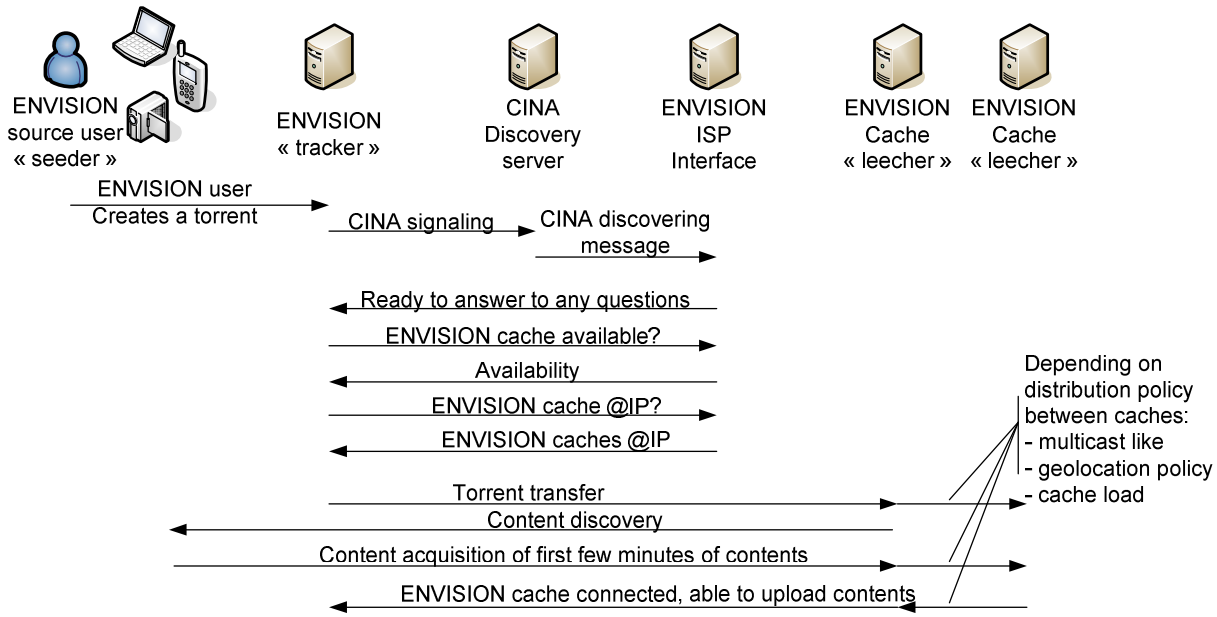


Figure 15. Call flows for cache ingestion

The call flows for delivery from cache is described in figure 16.

End-user gets torrent for a content available in ENVISION network. ENVISION leecher receives the list of peers capable of delivering the content. In this list and following communications between tracker and ISP thanks to CINA there are the addresses of ENVISION caches. Because these last ones are seeders (with the complete content), ENVISION leecher receives most part of content from caches.

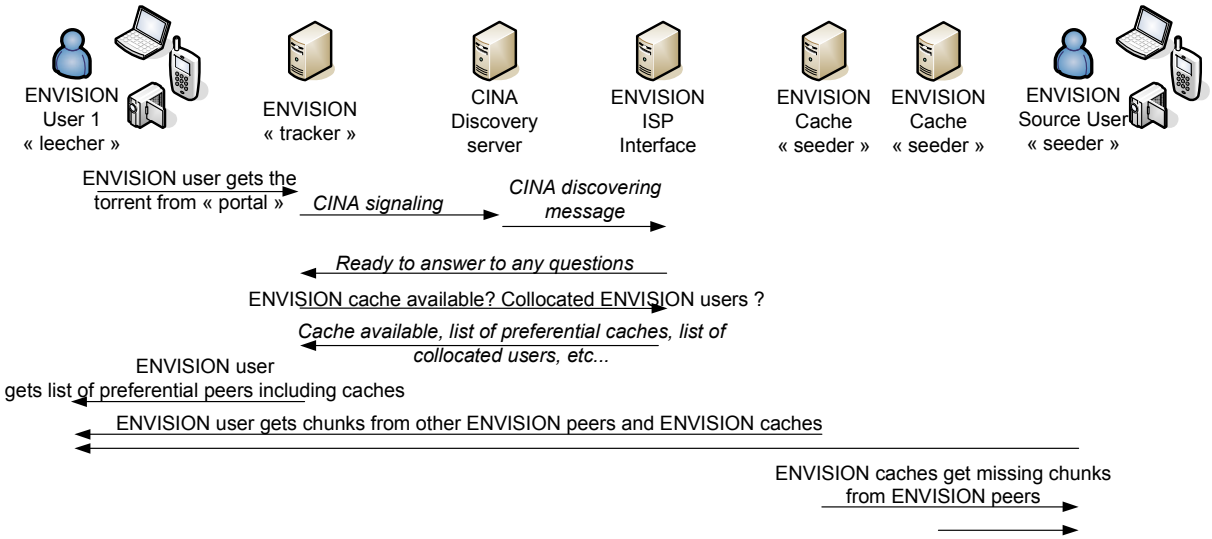


Figure 16. Call flows for content delivery by cache

4.3.2.2 Caching implementation

Implementation of specific modules for ENVISION caches needs to cooperate with development being done in 5.2.

Development is based on Windows platform. This part is detailed in ENVISION D5.2.

4.3.2.3 Caching integration in the network

The cache integration in the network has to fulfill the goal of the ENVISION project, while taking into consideration the engineering rules of ISPs. Thus, the important factors in cache deployment are appropriate cache placement by ISP, cache efficiency and the number of customers accessing the cache. For the appropriate placement of cache by ISPs, it is necessary to precise that application servers are not allowed at core level and limited at specific points in the backhaul network.

Cache can be located at access level, close to the final customers: at this location, connection of ENVISION nodes set (cache system) is authorized by ISP, the bandwidth saving is maximum for end-users inside the PoP (Point of Presence). In the case of independent caches, traffic is saved only in the equipped PoP but probably, traffic is not considerably reduced because the same request in different PoPs supply different caches with similar content. This multiple useless traffic can be avoided with integration of multiples caches, for example one cache per PoP and configuration of caches to transfer data between each others (collaborative caches).

4.3.2.3.1 Technical-economic studies: ISP dilemma

The dilemma is to use a large number of caches (so expensive deployment with low cost equipment) versus few caches with very expensive costs (equipment, networks interfaces).

This point in relation with tech-economical study is described in the following figure which doesn't take in account the ISP limitation in term of cache connection.

4.3.2.3.2 ISP costs

For tech-economic study, costs and gains for ISP need to be identified. For the costs, ISP is in charge to invest in the network transport such as IP (Internet Protocol) or WDM (Wavelength Division Multiplexing), for example in interfaces on routers with 1G to 10G and 40G, 100G in future. The caches lead hardware and software costs, storage capabilities, scalability of solutions.

To get a good return on investment, gains need to be evaluated. The first goal, bandwidth saving, is well known with network probes and ISP monitoring tools. ISP can interpret this gain as delay in network investment but, of course, it's not revenue. The "gain" can be studied on different part of network, interconnection links (between AS), inside core or backhaul ISP network. For QoE, it is really more difficult to measure its impact on ISP gain. With QoE given by cache usage, ISP can avoid possible end-users churn (often caused by bad experience in VOD delivery)

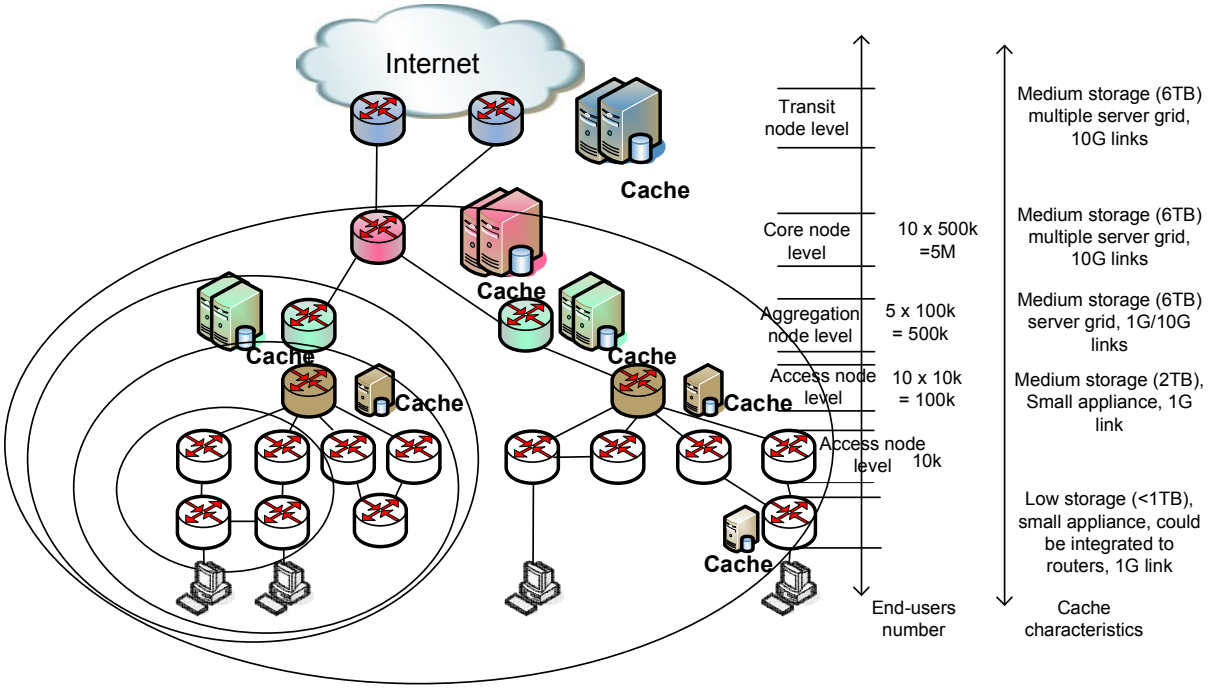


Figure 17. ISP network cache location at different network level

This sort of study based on network costs is difficult because these data are confidential. Following the studies made in OCEAN project, some prices level is chosen to compare the impact of cache location. Transport costs are limited to IP costs and retained as 0.12 for 1G, 1.00 for 10G. Cache solution costs are distributed as hardware (0.10), storage 1TB (0.10), delivery port 1G (0.50), delivery port 10G (5.0). To keep confidentiality, prices are indicated without any unit, it is only comparison of costs level.

Study is based on architecture network described in figure 17.

The delivery for cache is based on byte hit ratio from 20% to 45% depending end-users numbers which can be under cache delivery.

Cache storage calculation is based on request numbers multiplied by hit ratio and the average video size.

Cache number is function of delivery bandwidth calculated per location.

The table below gives the results:

	Cache number	Cost level				
		IP	Hard	Delivery	Storage	Total
Access	250	30	25	30	40	**125**
Aggreg.	50	24	5	24	45	98
Core	2	88	0.4	88	19.2	**196**
Interco.	2	88	0.4	88	19.2	**196**

Figure 18. Cache deployment CAPEX cost level depending network location

An ISP with network as described in figure 18 gets the most interesting return of investment if cache location is based on aggregation level. This sort of calculation is not valid for small networks and results give a preferable location at interconnection routers.

4.3.2.3.3 QoE

How to characterize QoE? The quality of experience is subjective but it is a mix of different parameters on end-users side and cache side which can be evaluated as download speed, bandwidth consumption, latency due to divert mechanism (responsible of divert traffic to the cache).

To evaluate the gain of cache on data transfer between two ENVISION clients, we consider the weight of network for each elements. Example is shown in figure 19

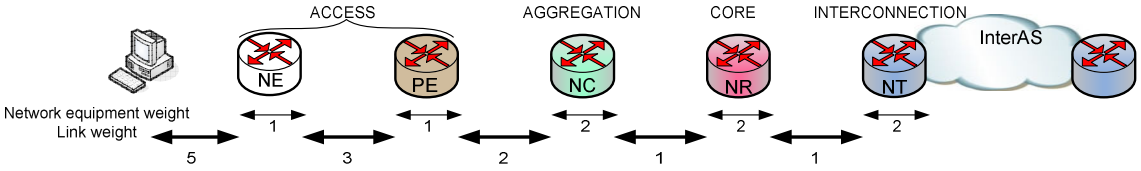


Figure 19. Characterization of network weight

Considering these weights, we can calculate the amount of the points depending paths between two ENVISION end-users. Same calculation is made with cache for different location.

Figure 20 shows the different possible paths following end-users locations in ISP network or inter ISPs (inter AS).

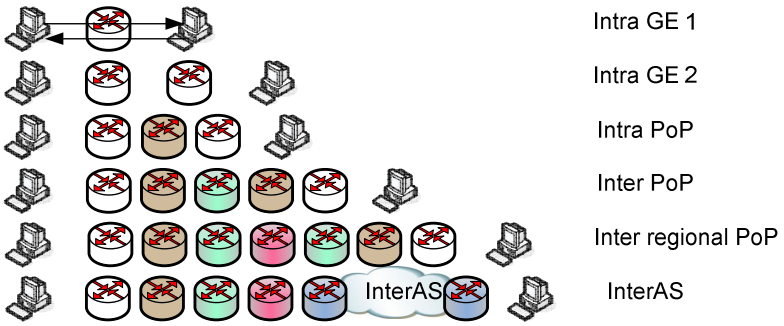


Figure 20. Different paths for ENVISION exchanges

With both figures 19 and 20, we can determine the matrix described in Figure 21. The table provides for each path the possible amount of points (weight) given by crossed pieces of equipment and links between these. The blue cell represents the cache location, green cells represents the saved path (thanks to the cache). For inter-PoP exchanges, 2 cache locations are considered: at core level and at aggregation level. These results can be considered as a picture of QoE for network. To develop QoE results, it is also necessary to consider the download speed for end-users. To measure it, the cache needs to be able to get the ingestion time and delivery time for each content. The first one will be

considered as equal to end-user download time without any cache (cache is an end-user), the second one is the same parameter with the cache. These two values with content volume will give download bandwidth without and with cache and the difference can be represent the QoE for end-users. The results (Gain in the Figure 21) is in accordance with expectations: the cache located in access or aggregation is the most interesting for QoE.

Configuration\Path												QoE		
Intra GE 1	PC-NE	NE	NE-NE	NE	NE-PC							Total no cache	Total with cache	Gain
1	5	1	3	1	5							15	6	60%
Intra GE 2 2	PC-NE	NE	NE-NE	NE	NE-PE	PE								
2	5	1	3	1	3	1						23	14	39%
			NE-PC	NE	PE-NE									
			5	1	3									
Intra PoP	PC-NE	NE	NE-PE	PE	PE-NC	NC	NC-NR	NR						
3	5	1	3	1	2	2	1	2				32	20	38%
	NE-PC	NE	PE-NE	PE	NC-PE	NC	NR-NC							
	5	1	3	1	2	2	1							
Inter PoP 1	PC-NE	NE	NE-PE	PE	PE-NC	NC	NC-NR	NR	NR-NT	NT				
4	5	1	3	1	2	2	1	2	1	2		38	23	39%
	NE-PC	NE	PE-NE	PE	NC-PE	NC	NR-NC	NR	NR-NT					
	5	1	3	1	2	2	1	2	1					
Inter PoP 2	PC-NE	NE	NE-PE	PE	PE-NC	NC	NC-NR	NR	NR-NT	NT				
4	5	1	3	1	2	2	1	2	1	2		38	14	63%
	NE-PC	NE	PE-NE	PE	NC-PE	NC	NR-NC	NR	NR-NT					
	5	1	3	1	2	2	1	2	1					
Inter AS	PC-NE	NE	NE-PE	PE	PE-NC	NC	NC-NR	NR	NR-NT	NT	peering			
5	5	1	3	1	2	2	1	2	1	2	10	30	20	33%
6	5	1	3	1	2	2	1	2	1	2	10	30	17	43%
7	5	1	3	1	2	2	1	2	1	2	10	30	14	53%
8	5	1	3	1	2	2	1	2	1	2	10	30	10	67%
9	5	1	3	1	2	2	1	2	1	2	10	30	6	80%

Figure 21. QoE calculation based on subjective network delay saved by cache

4.3.2.3.4 Upstream traffic

Considering cache location at core level in the network, the cache can be the seeder for different PoP, avoiding a big number of caches but the load in the cache system can be important.

Following ISP point of view, it is also necessary to study the impact on upstream traffic generated by ENVISION caches. So, the question is: ENVISION caches must be used to ISP end-users (upstream traffic generated only for ISP customers) or ENVISION caches (part of ENVISION overlay) can be a privileged source for any ENVISION peers, so increasing upstream bandwidth consumption in ISP interconnection network?

For ENVISION, the cache is used only for delivery to ISP end-users, so it bechaves as a leecher for the external ENVISION peers.

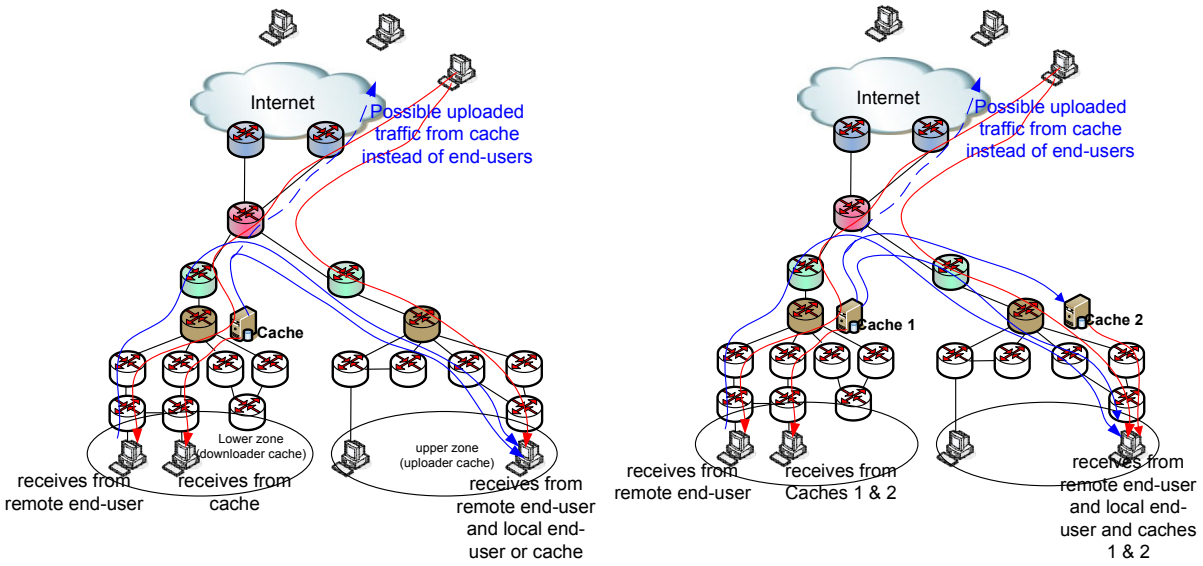


Figure 22. a) 1 cache for 2 PoPs **b)one cache per PoP**

The figures 22a and 22b represent two scenarios for cache location. In both scenarios, caches are located at same network level. Exchanges are more important with two caches and bandwidth consumption should be increase but the presence of cache 2 brings better quality of experience for end-users (under cache n°2 influence).

These improvements have to be checked by testing.

4.3.2.4 Example of use

The following details a possible use of the cache in ENVISION.

1) ISP provides ENVISION cache in its network. This cache is in service. ENVISION ISP interface is aware of cache presence with "in service" status and connected to tracker.
2) A Video-on-Demand (VoD) is published by an external end-user to ENVISION tracker. ENVISION tracker sends the new torrent to the caches which are connected.
3) Caches like any ENVISION nodes send request to the tracker which returns the list of seeders (one for original deployment).
4) ENVISION caches acquiring the chunks of content are seen as seeders by ENVISION tracker and can deliver content for each customers asking this content.
5) ISP can configure its caches as seeders only for specific zone and restrict the upload to external leechers.

4.4 High Capacity Node

Peer-to-peer applications can scale up in available resources with the number of participants. Relying on mutual cooperation between the peers and without pre-provisioned server resources file sharing peer-to-peer application in particular have proven to provide a cost-efficient alternative for distributing content in a large scale. However, participating peers in peer-to-peer applications are usually limited in upload capacity, the aggregate of which determines the total bandwidth resources available to the system. While there are various proposals for utilizing available peer resources more efficiently, e.g., by splitting content and arranging multiple distribution trees, the bandwidth limitations eventually result in distribution paths with delays that are too high for time critical

applications. Consequently, peer-to-peer multimedia applications like videoconferencing (Skype) or livestreaming (PPLive) are typically limited in the number of concurrent users or cannot distribute video feeds in real time to all users.

In this chapter we introduce the High Capacity Node (HCN) network service which aims in boosting the available total bandwidth of the peer-to-peer overlays in a dynamic and on-demand fashion. The HCN service allows the overlay to request the instantiation of nodes with high fan-out capacities in ENVISION enabled network domains through the CINA interface. The instantiated nodes with their high capacities are integrated into the overlay, which is then enabled to support higher quality streams and construct distribution trees with maximum delay suitable for time critical applications. The HCN service is designed as a network service that runs on a specialized routing platform deployed at strategic locations within the network. Thus HCNs can be instantiated at critical parts of the network, for example where the overlay network currently has to cope with bottleneck links.

In this section we introduce the general architecture of the HCN network service and its relationship to other ENVISION software modules. Subsequently, we introduce the Research Routing Platform, which the network service is based on. Finally, we discuss and compare implementation options of the HCN network service.

4.4.1 Architecture Overview

The architecture overview illustrated in Figure 23 shows the main modules of the HCN network service and their relationship to other ENVISION components. The overlay layer including the push-based peer-to-peer IVCD system, which is further described in section 5 in [D4.2], is shown in the upper half of the figure. The peer-to-peer overlay uses the CINA interface to retrieve information about the network state as well as to request network services from a CINA server. In a typical scenario a peer-to-peer network spans multiple network domains, some of which are ENVISION enabled and support the CINA interface, whereas others only provide basic transport service. Therefore the overlay needs to deploy a strategy for deciding where in a multi-domain to invoke a network service. One algorithm addressing this question is described in more details in section 5.2 in [D4.2]. Once the decision for invocation of the HCN network service had been made, the overlay uses the CINA Client Library (see section 2.5) for allocating, deallocating and modifying parameters of the network service. The CINA Server and Client library implement the CINA protocol specification for the available network service methods, which can be found in the Appendix.

The implementation of the HCN network service has two main components: the CINA Server and the Research Routing Platform. The network service control is integrated in the CINA Server and negotiates instantiation with peer-to-peer overlays. It is connected to (a number of) a control module in the Research Routing Platform, which instantiate the actual service instances.

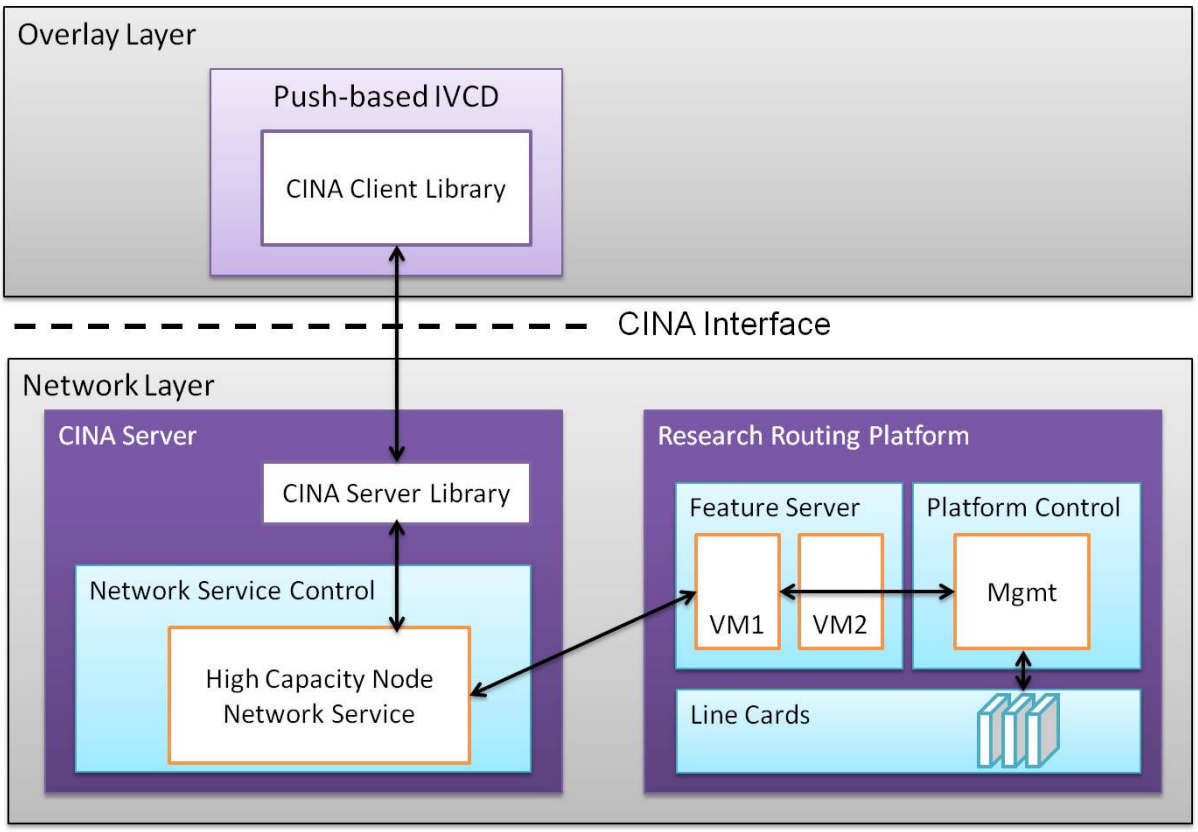


Figure 23. HCN architecture overview

4.4.1.1 Research Routing Platform Overview

The Alcatel-Lucent Research Routing Platform hosting the HCN network service is a carrier-grade media processing platform. It provides fundamental network functionalities like routing and switching as well as advanced packet processing.

The platform is realized in a standardized 14 slot Advanced Telecom Computing Architecture (ATCA) chassis, which allows for an open architecture by providing the ability to use blades from different manufacturers. The platform is based on a Linux (kernel series 2.6) as an operating system, which allows for porting third party software onto the platform.

The platform is composed of at least one of each of the supported blade types:

- Integrated Hub Card (IHC): The IHC serves as the Platform Control and controls all core system functions. Furthermore it provides access to the management interfaces of the system via serial connection or secure shell (SSH).
- Line Service Card (LSC): The LSC is the main networking component of the platform. It provides 16 high-density Ethernet ports with switching capabilities.
- Feature Server Card (FSC): The FSC is a computer card with 8 CPU cores. It is used for intensive processing tasks that cannot be executed

on the IHC, like third party applications hosted in Virtual Machines.

- Storage Resource Card (SRC): The SRC enhances the FSC with an additional 1.2 TB of RAID disk storage.

Traffic entering through a LSC is forwarded to outgoing ports according to a set of configurable rules. Traffic that requires processing of packets is redirected to one or more FSCs before being forwarded to an outgoing port.

4.4.1.2 Applications on the Research Routing Platform

There are two types of applications that can be hosted on the platform: native and loosely coupled applications. Native applications run directly on the platform and have access to an API (Application Programming Interface) that enables interacting with the platform to control routing rules, query interface state, etc. Loosely coupled applications are Linux processes that usually run in a virtual machine (VM). They can only receive traffic if configured routing rules trigger their invocation. At present, VMs are permitted access to up to two traffic network interfaces. The platform uses paravirtualization, a technique with which the guest is aware that it is a virtual machine, and thus, is able to communicate more efficiently with its host system. VMs are installed, configured and launched with the platforms Command Line Interface (CLI), which connects to the VM via ssh when it is launched. A software development kit (SDK) containing a preconfigured Linux kernel with the necessary applications and gcc/g++ compilers for the platforms architecture is available.

4.4.2 Implementation options for the HCN network service

There are several implementation options for the HCN service, two of which we discuss in more detail in the following subsections. For the discussion, it is helpful to recall the main purpose of the HCN network service, which is the provisioning of the peer-to-peer overlay with additional bandwidth to boost its overall capacity.

While the performance aspect is of foremost importance, the HCN design is also influenced by several other aspects, one of which is the flexibility of the service to support a large class of overlay applications. Peer-to-peer networks usually use proprietary, sometimes even closed protocols. Supporting a large number of different protocols or versions of a protocol requires either reverse engineering the protocols or integration of the third-party software needs into the service, either implying high development or integration effort, respectively.

Another option is for the HCN service to only support generic packet handling (replication, forwarding) without support for overlay control protocols. This application agnostic approach can support a larger number of applications in parallel; however, in this case the overlay application needs to have some knowledge about the HCN service behaviour and implies development effort from the client developer.

4.4.2.1 P2P software as loosely coupled application

The first option that we consider is to integrate the overlay application as third party server onto the research routing platform. Therefore the application needs to be compiled with the platform SDK, which creates an image of a virtual machine that can be hosted by the platform. The main advantage

of this option is that the HCN network service appears to the overlay application as another peer in the overlay network. For regular peers it is transparent whether they connect to the node provided by the network service or another end-user hosted peer. In order to make the HCN flexible enough it is required to integrate peer software for each supported protocol type into the Research Routing Platform. When an overlay wants to invoke an HCN during runtime, it needs to negotiate via the CINA protocol whether its protocol version is supported or not. Another disadvantage of this option is that both media and control traffic is processed in the virtual machines instance of the application perceived as additional latency in the overlay application.

4.4.2.2 IP forwarding by line cards

The second implementation option we consider is to design the HCN service only as a control module on the Research Routing Platform, which is able to modify the forwarding tables according to the needs of the peer-to-peer overlay and keep the traffic in the fast path. In this option the HCN node does not participate in the overlay topology as a regular entity that speaks the control protocol. In contrast it would serve only as a multicast reflector for media traffic. Incoming IP media packets would be rewritten and send out to multiple destinations. In this way the sending peer only needs to transmit one IP packet, but is able to serve several neighbouring peers.

The advantage of this option is that no third party software needs to be deployed on the platform, which makes the HCN much more general. In contrast to native IP multicast, the HCN service does not use the IGMP protocol, which allows the HCN being deployed across network boundaries. Also receiving peers typically do not need to know that a HCN network service is active. Only the sending peer needs to be updated to support and configure the HCN. This flexibility comes for the price of being less effective compared to native IP multicast. And IP forwarding as described previously only works well for UDP traffic, as TCP connection states cannot easily be supported by this method. Thus, it is only suited for overlay applications supporting the UDP protocol for media transmission.

4.4.2.3 Conclusion

For the prototype implementation in ENVISION we will implement the HCN for the push-based IVCD system (see section 5 in D4.2). The IVCD system is a tree-based overlay using UDP as its transport protocol for media traffic, for which the second implementation option is very-well suited. In addition, this option can much more efficiently leverage the capabilities of the Research Routing Platform as there is no need to process data other than in the fast path. The HCN implementation will comprise the following modules:

HCNServiceController: This module is responsible for direct negotiations with the overlay. Consequently, this module will be integrated with the CINA server and will mediate with the HCNController(s) on the Research Routing Platform(s). It keeps state about which HCNs are activated for which peer-to-peer overlays and suspends the service once a subscription has expired.

HCNController: This module is responsible for configuring the Research Routing Platform itself. Therefore, it is integrated with the platform and registers itself with the HCNServiceController on the CINA server. The HCNController is implemented as a loosely-coupled application, thus, running inside a virtual machine on the Research Routing Platform.

4.5 Network optimisation Logic based on preferences: Shifting ISP traffic in time and space to reduce transit bills

This section addresses the problem of customer Internet Service Providers (ISPs) who wish to reduce their bills for transit. Recently a number of publications have used the idea of incentivising users to time delay traffic [LR08, JWHC11a, JWHC11b]. A related thread of work has considered the benefit of user traffic routing via other destinations to save ISP costs for example ALTO/P4P [XYKLS08] and ONO [CB08]. In both of these cases, the altered user behaviour can improve the situation for the ISP by reducing their bills from transit providers or by reducing their peak traffic.

The proposed technique combines the temporal and spatial shifting of traffic into a single mathematical framework. It shows how ISPs can increase profits by incentivising users to move downloading habits in both time and space. A key insight here is the much studied nature of 95th percentile pricing [DHKS09, SLR10] where ISPs are often billed by their transit providers not according to peak or average network usage but according to the 95th percentile rate.

The rest of this section has been suppressed from the public version of this deliverable.

5. CONCLUSION

During the second year of the ENVISION project, a considerable work has been performed on the specifications of the CINA interface. This interface is now well specified, from the mutual exchanges of information between the overlay applications and the network (multi-cost map, footprint map, constraint map) to the activation of available network services. This specification is completely aligned with the current IETF ALTO framework, so that part of the ENVISION work has been proposed to the ALTO working group. The implementation of CINA interface has been started recently and will be continued duing the 3rd year of the project so as to be used later in WP6 with the demonstrators.

The monitoring part, initiated in year 1 with a survey of network metrics and existing tools has been performed via the selection of the most pertinent metrics for the project as well as with the implementation of a monitoring module, associated with a database, storing the values and being requested by the CINA server when necessary.

The network services we identified as the most important ones for the objectives of the project (multicast, caching , high capacity node) were further investigated, designed and the implementation of them has been started so as to prepare the demonstrators for the final year, to be tested in the available testbeds. A mathematical study to quantify the network gains for an ISP has also been performed during this year, highlighting the interest for an ISP to shift the data traffic in time and space when it is possible.

For the final year of the project, only the third task will still be active (network services) for some months. During this time, the implementation of the CINA interface as well as the network services will be completed before being integrated and evaluated in WP6.

REFERENCES

[AMD09] Antoniades, Demetris and Markatos, Evangelos P. and Dovrolis, Constantine. One-click hosting services: a file-sharing hideout. *Proc. of Internet Measurement Conference (IMC)*, pages 223-234, 2009.

[bep24] Harrison, D., "Tracker Returns External IP", BEP http://bittorrent.org/beps/bep_0024.html.

[CB08] David R. Choffnes and Fabián E. Bustamante. Taming the torrent: a practical approach to reducing cross-ISP traffic in peer-to-peer systems. *Proc. of ACM SIGCOMM*, 2008.

[CLRS10] Parminder Chhabra and Nikolaos Laoutaris and Pablo Rodriguez and Ravi Sundaram. Home is where the (fast) Internet is: Flat-rate compatible incentives for reducing peak load. *Proceedings of ACM HomeNETS*, 2010.

[CRLYSR11] Ruben Cuevas Rumin and Nikolaos Laoutaris and Xiao Yang and Georgos Siganos and Pablo Rodriguez. Deep Diving into BitTorrent Locality. *Proc. of IEEE INFOCOM*, 2011.

[D07] John R. Douceur. Lottery trees: motivational deployment of networked systems. *Proc. of ACM SIGCOMM*, pages 121-132, 2007.

[D3.1] Envision D3.1 deliverable, “Initial Specification of the ENVISION Interface, Network Monitoring and Network Optimisation Functions”, Public report, final version, 7 February 2011

[DHKS09] Xenofontas Dimitropoulos and Paul Hurley and Andreas Kind and Marc Ph. Stoecklin. On the 95-percentile billing method. *Proc. of Passive and Active Measurement Conference (PAM)*, 2009.

[I.D.draft-randriamasy-alto-multi-cost] Randriamasy, S., Schwan, N., “Multi-Cost ALTO”, draft-randriamasy-alto-multi-cost-05, 2011

[I.D.draft-schwan-alto-incr-updates] Randriamasy, S., Schwan, N., “ALTO Incremental Update”, draft-schwan-alto-incr-updates-00, December 2011

[I-D.alto-server-discovery] Kiesel, S., Stiemerling, M., Schwan, N., Scharf, M., Song, H., "ALTO Server Discovery", draft-ietf-alto-server-discovery-02, September 2011.

[I-D.ietf-alto-protocol] Alimi, R., Penno, R., and Y. Yang, "ALTO Protocol", draft-ietf-alto-protocol-10, June 2011.

[I-D.ietf-alto-reqs] S. Kiesel, S. Previdi, M. Stiemerling, R. Woundy, and R. Y. Yang, "Application-Layer Traffic Optimization (ALTO) Requirements," IETF draft-ietf-alto-reqs-11.txt, July 11, 2011.

[I-D.jenkins-alto-cdn-use-cases] Niven-Jenkins, B., Watson, G., Bitar, N., Medved, J., and S. Previdi, "Use Cases for ALTO within CDNs", draft-jenkins-alto-cdn-use-cases-01, June 2011.

[I-D.pbryan-json-patch] Bryan, P., "JSON Patch", draft-pbryan-json-patch-02, October 2011.

[IPPM] IETF IP Performance Metrics group, https://datatracker.ietf.org/wg/ippm/

[JSchema] A JSON Media Type for Describing the Structure and Meaning of JSON Documents, http://tools.ietf.org/html/draft-zyp-json-schema-03

[JWHC11a] Carlee Joe-Wong and Sangtai Ha and Mung Chiang. Time-Dependent Broadband Pricing: Feasibility and Benefits. *Proc. of International Conference on Distributed Computing Systems (ICDCS)*, 2011.

[JWHC11b] Carlee Joe-Wong and Sangtai Ha and Mung Chiang. Time-Dependent Internet Pricing. *Proc. of Internet Technologies and Applications Conference (ITA)*, 2011.

[LR08] Nikolaos Laoutaris and Pablo Rodriguez. Good things come to those who (can) wait: or how to handle Delay Tolerant traffic and make peace on the Internet. *Proc. of ACM HotNets*, 2008.

[NMWG] OGF Network Measurement Working Group, http://nmwg.internet2.edu/

[O01] Andrew Odlyzko. Internet pricing and the history of communications. *Computer Networks*, 36:493-517, 2001.

[PFAS10] Poese, Ingmar and Frank, Benjamin and Ager, Bernhard and Smaragdakis, Georgios and Feldmann, Anja. Improving content delivery using provider-aided distance information. *Proc. of Internet Measurement Conference (IMC)*, pages 22-34, 2010.

[PGP] C. Jon, D. Lutz, F. Hal, S. David, and T. Rodney, "OpenPGP Message Format," IETF RFC 4880, November 2007.

[RFC2730] S. Hanna, B. Patel, M. Shah, Multicast Address Dynamic Client Allocation Protocol (MADCAP), RFC 2730, December 1999

[RFC2771] R. Finlayson, An Abstract API for Multicast Address Allocation, RFC2771, February 2000

[RFC3958] Daigle, L. and A. Newton, "Domain-Based Application Service Location Using SRV RRs and the Dynamic Delegation Discovery Service (DDDS)", RFC 3958, January 2005.

[RFC4848] Daigle, L., "Domain-Based Application Service Location

[RFC5986] Thomson, M. and J. Winterbottom, "Discovering the Local Location Information Server (LIS)", RFC 5986, September 2010.

[S84] Michael J. Smith. The stability of a dynamic model of traffic assignment -- an application of a method of Lyapunov. *Transportation Science*, 18(3):245-252, 1984.

[SLR10] Rade Stanojevic and Nikolaos Laoutaris and Pablo Rodriguez. On economic heavy hitters: Shapley value analysis of 95th percentile pricing. *Proc. of Internet Measurement Conference (IMC)*, 2010.

[SNMP] J. Case, M. Fedor, M. Schoffstall, J. Davin, A Simple Network Management Protocol (SNMP), RFC 1157, May 1990

[W52] J. G. Wardrop. Some theoretical aspects of road traffic research. *Proc. of the Inst. of Civil Engineers II*, 1:325-378, 1952.

[X509] C. Adams, S. Farrell, T. Kause, and T. Mononen, "Internet X.509 Public Key Infrastructure Certificate Management Protocols," IETF RFC 4210, September 2005.

[X509PKI] D. Cooper, et al., "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," IETF RFC 5280, May 2008.

[XYKLS08] Haiyong Xie and Yang Richard Yang and Arvind Krishnamurthy and Yanbin Liu and Avi Silverschatz. P4P: Provider portal for applications. *Proc. of ACM SIGCOMM*, 2008.

APPENDIX A – MULTICAST NETWORK SERVICE SPECIFICATIONS

A1. CINA Base Schema

This schema defines base data types that may be useful across different CINA network services.

A1.1. Address Type

```
{
  "id" : "AddressType",
  "type" : "string",
  "enum" : [ "ipv4", "ipv6" ]
}
```

A1.2. Typed Endpoint Address

```
{
  "id" : "TypedEndpointAddr",
  "type" : "string",
  "pattern" : "(ipv4:(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-
5]|2[0-4]\d|1\d\d|[1-9]?\d)){3})|(ipv6:([0-9A-Fa-f]{1,4})(:[0-9A-Fa-
f]{1,4}){7})"
}
```

A1.3. Endpoint Port

```
{
  "id" : "EndpointPort",
  "type" : "integer",
  "minimum" : 0,
  "maximum" : 65535
}
```

A1.4. Time Range

```
{
  "id" : "TimeRange",
  "type" : "object",
  "properties" : {
    "min" : {
      "type" : "string",
      "format" : "date-time"
    },
    "max" : {
      "type" : "string",
      "format" : "date-time"
    }
  }
}
```

A2. Multicast

A2.1. Multicast Allocate

A2.1.1. Media Type

```
   The media type is
   "application/json;profile=http://www.envision-
project.org/cina/multicast-schema#AllocateResponse".

A2.1.2. HTTP Method

   This method creates a multicast lease resource using the HTTP POST
   method.

A2.1.3. Input Parameters

   Input parameters are supplied in the entity body of the POST
request.
   The data format of the input parameters is a JSON object described
by
   the
   http://www.envision-project.org/cina/multicast-
schema#AllocateRequest
   JSON schema:

   {
     "id" : "Source",
     "type" : "object",
     "properties" : {
       "src-address" : {
         "$ref" :
         "http://www.envision-project.org/cina/base-
schema#TypedEndpointAddr",
         "required" : true
       },
       "src-port" : {
         "$ref" :
         "http://www.envision-project.org/cina/base-
schema#EndpointPort",
         "required" : true
       }
     }
   }

   {
     "id" : "AllocateRequest",
     "properties" : {
       "multicast-address-family" : {
         "$ref" :
         "http://www.envision-project.org/cina/base-
schema#AddressType",
         "default" : "ipv6"
       },
       "start-time" : {
         "$ref" :
         "http://www.envision-project.org/cina/base-schema#TimeRange"
       },
       "lease-time" : {
         "$ref" :
         "http://www.envision-project.org/cina/base-schema#TimeRange"
       },
       "current-time" : {
```

```
          "type" : "string",
          "format" : "date-time"
        },
        "srcs" : {
          "type" : "array",
          "required" : true,
          "items" : {
            "id" : "SourceRequest",
            "extends" : {"$ref" : "Source"},
            "properties" : {
              "max-bitrate" : {
                "type" : "number"
              }
            }
          },
          "minItems" : 1
        }
        "forecasted-user-number" : {
          "type" : "number"
        }
      }
    }
```

with properties:

multicast-address-family The address family for the requested multicast address.

start-time The time moment when the lease of the multicast resources will begin. When omitted the server assumes the multicast resources are requested immediately.

lease-time The time period for which to allocate the multicast resources to the application.

current-time The current time at the CINA Client; used for approximate clock synchronisation with the CINA Server.

srcs The list of multicast sources. For each source, the host address and port are provided in order to establish the corresponding access rules at the network. The max-bitrate parameter denotes the maximum bitrate a source at which the source may be transmitting at any point in time.

forecasted-user-number The number of receivers that will be receiving on average from the multicast address, based on the application forecast. This parameter MAY be omitted when the application cannot predict this number.

A2.1.4. Response

After successfully creating the resouce, HTTP returns response code 201 (Created). The data format of the response is a JSON object described by the

```
   http://www.envision-project/org/cina/multicast-
schema#AllocateResponse
   JSON schema:

   {
     "id" : "AllocateResponse",
     "properties" : {
       "multicast-address" : {
         "$ref" :
         "http://www.envision-project.org/cina/base-
schema#TypedEndpointAddress",
         "required" : true
       },
       "start-time" : {
         "type" : "string",
         "format" : "date-time",
         "required" : true
       },
       "lease-time" : {
         "id" : "LeaseTime",
         "type" : "string",
         "format" : "date-time",
         "required" : true
       },
       "current-time" : {
         "type" : "string",
         "format" : "date-time"
       },
       "srcs" : {
         "type" : "array",
         "required" : true,
         "items" : {
           "id" : "SourceResponse",
           "extends" : {"$ref" : "Source"}
         },
         "minItems" : 1
       }
     }
   }


   with properties:

   multicast-address  The multicast address to be used for receiving
       from the multicast tree.

   start-time  The time moment when the lease of the multicast
resources
       will begin.

   lease-time  The time period for which the multicast resources are
       allocated to the application. When the lease-time is specified
in
       the HTTP Request message, the value of this field in the
       corresponding HTTP Response message MUST be within the range
       provided in the HTTP Request.

   current-time  The current time at the CINA Server; used for
```

approximate clock synchronisation with the CINA Client.

srcs The list of multicast sources. For each source, the provided host address and port are used to identify the source.

A2.1.5. Capabilities

The protocols and protocol versions used to provide this resource may differ. The capabilities of a CINA Server URI providing this resource are defined by a JSON object described by the http://www-envision-project.org/cina/multicast-schema#MulticastCapability JSON schema:

```
{
  "multicast-address-family" : {
    "type" : "array",
    "required" : true,
    "items" : {
      "$ref" :
      "http://www.envision-project.org/cina/base-schema#AddressType"
    },
    "minItems" : 1
  },
  "IGMP-version" : {
    "type" : "array",
    "required" : true,
    "items" : {
      "type" : "string",
      "enum" : [ "IGMPv1", "IGMPv2", "IGMPv3" ]
    },
    "minItems" : 1
  }
}
```

with properties:

multicast-address-family Specifies the supported address families.

IGMP-version Specifies the IGMP protocol version supported by the ISP.

Further, this resource may be provided with additional functionality, including a multicaster component. In this case a set of additional parameters are required for this resource. The following section provides more details.

A2.1.5.1. Multicaster Transmission Mode

When a multicaster component is used, the data format of the input parameters and the response message changes as described in the

following sections.

A2.1.5.1.1. Media Type

The media type becomes
"application/json;profile=http://envision-project.org/cina/multicaster-schema#AllocateResponse".

A2.1.5.1.2. Input Parameters

The data format of the input parameters is a JSON object described by
the
http://www.envision-project.org/cina/multicaster-schema#AllocateRequest
JSON schema:

```
{
  "id" : "Source",
  "extends" : {
    "$ref" :
    "http://www.envision-project.org/cina/multicast-schema#Source"
  },
  "properties" : {
    "destination-port" : {
      "$ref" :
      "http://www.envision-project.org/cina/base-schema#EndpointPort",
      "required" : true
    }
  }
}

{
  "id" : "AllocateRequest",
  "extends" : {
    "$ref" :
    "http://www.envision-project.org/cina/multicast-schema#AllocateRequest"
  },
  "properties" : {
    "srcs" : {
      "items" : {
        "id" : "SourceRequest",
        "extends" :
          [
            {
              "$ref" :
              "http://www.envision-project.org/cina/multicast-schema#SourceRequest"
            },
            { "$ref" : "Source" }
          ]
      }
    }
  }
}
```

```
   with properties:

   srcs  The list of multicast sources. Additionally to the base
       properties, the destination-port is set by the CINA Client,
       denoting the destination port used by the multicaster to
transmit
       the multicast flow to the final receivers.
```

A2.1.5.1.3. Response

```
   The data format of the response is a JSON object described by the
   http://www.envision-project.org/cina/multicaster-
schema#AllocateResponse
   JSON schema:

   {
     "id" : "AllocateResponse",
     "extends" : {
       "$ref" :
       "http://www.envision-project.org/cina/multicast-
schema#AllocateResponse"
     },
     "properties" : {
       "multicaster-address" : {
         "$ref" :
         "http://www.envision-project.org/cina/base-
schema#TypedEndpointAddress",
         "required" : true
       },
       "multicast-source-address" : {
         "$ref" :
         "http://www.envision-project.org/cina/base-
schema#TypedEndpointAddress",
       }
       "srcs" : {
         "items" : {
           "id" : "SourceResponse",
           "extends" :
             [
               {
                 "$ref" :
                 "http://www.envision-project.org/cina/multicast-
schema#SourceResponse"
               },
               { "$ref" : "Source" }
             ],
           "properties" : {
             "multicaster-port" : {
               "$ref" :
               "http://www.envision-project.org/cina/base-
schema#EndpointPort",
               "required" : true
             }
           }
         }
       }
```

```
    }
  }
```

with properties:

multicaster-address The address of the multicaster device to be used as a destination address from the multicast sources.

multicast-source-address The multicast source address to be used in IGMP. This parameter MUST be omitted when IGMP-version is earlier than IGMPv3 in the multicast service capabilities.

srcs The list of multicast sources. Additionally to the base properties, the multicaster-port is specified, denoting the destination port the application will use to transmit to the multicaster. The CINA Server sets this property.

A2.1.6. Example

```
POST /multicast HTTP/1.1
Host: custom.cina.example.com
Content-Length: [TODO]
Content-Type:   application/json;
                profile=http://www.envision-project.org/cina/multicaster-schema#AllocateRequest
Accept:  application/json;
         profile=http://www.envision-project.org/cina/multicaster-schema#AllocateResponse

{
  "multicast-address-family" : "ipv4",
  "start-time" : {
    "min" : "2011-12-19T16:31:57",
    "max" : "2011-12-19T18:39:57"
  },
  "lease-time" : {
    "min" : "2011-12-19T16:59:57",
    "max" : "2011-12-19T18:29:57"
  },
  "current-time" : "2011-12-19T16:29:57",
  "srcs" : [
    {
      "src-address" : "123.123.123.12",
      "src-port" : 1111,
      "destination-port" : 1111,
      "max-bitrate" : 512
    },
    {
      "src-address" : "123.123.123.13",
      "src-port" : 1111,
      "destination-port" : 1111,
      "max-bitrate" : 256
    }
  ],
```

```
     "forecasted-user-number" : 555
   }


   HTTP/1.1 201 Created
   Location: http://custom.cina.example.com/multicast/12345678
   Content-Type: application/json;
            profile=http://envision-project.org/cina/multicaster-
schema#AllocateResponse

   {
     "id" : "12345678",
     "multicast-address" : "ipv4:123.123.123.11",
     "start-time" : "2011-12-19T16:34:57",
     "lease-time" : "2011-12-19T17:34:57",
     "current-time" : "2011-12-19T16:30:11",
     "srcs" : [
       {
         "src-address" : "123.123.123.12",
         "src-port" : 1111,
         "destination-port" : 1111,
         "multicaster-port" : 2222
       },
       {
         "src-address" : "123.123.123.13",
         "src-port" : 1111,
         "destination-port" : 1111,
         "multicaster-port" : 2223
       }
     ],
     "multicaster-address" : "ipv4:123.123.123.10",
     "multicast-source-address" : "ipv4:123.123.123.10"
   }
```

A2.2. Multicast Update

A2.2.1. Media Type

The media type is "application/json;profile=http://www.envision-project.org/cina/multicast-schema#AllocateResponse".

A2.2.2. HTTP Method

This method requests updates to the current multicast session using the HTTP PATCH method. Valid updates to a multicast session initiated by a CINA client include increasing or decreasing the multicast lease time, adding, replacing and removing overlay source addresses.

A2.2.3. Input Parameters

The data format of the input parameters is a JSON object described by the http://www.envision-project.org/cina/multicast-schema#UpdateRequest

JSON schema that is compatible with the [I-D.pbryan-json-patch] JSON PATCH format:

```
{
  "id" : "PatchOperation",
  "type" : "object",
}

{
  "id" : "UpdateRequest",
  "type" : "array",
  "items" : "PatchOperation",
  "minItems" : 1
}

{
  "id" : "PatchEditLeaseTime",
  "extends" : {"$ref" : "PatchOperation"},
  "properties" : {
    "replace" : {
      "type" : "string",
      "enum" : ["/lease-time"],
      "required" : true
    },
    "value" : {
      "type" : {
        "$ref" :
        "http://www.envision-project.org/cina/base-schema#TimeRange"
      },
      "required" : true
    }
  }
}

{
  "id" : "PatchAddSourceAddress",
  "extends" : {"$ref" : "PatchOperation"},
  "properties" : {
    "add" : {
      "type" : "string",
      "enum" : ["/srcs"],
      "required" : true
    },
    "value" : {
      "type" : { "$ref" : "SourceRequest" },
      "required" : true
    }
  }
}

{
  "id" : "PatchEditSourceAddress",
  "extends" : {"$ref" : "PatchOperation"},
  "properties" : {
    "replace" : {
      "type" : "string",
      "pattern" :
```

```
          "/srcs/((ipv4:(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-
5]|2[0-4]\d|1\d\d|[1-9]?\d)){3})|(ipv6:([0-9A-Fa-f]{1,4})(:[0-9A-Fa-
f]{1,4}){7})):(6553[0-5]|655[0-2]\d|65[0-4]\d\d|6[0-4]\d\d\d|[1-
5]\d\d\d\d|[1-9]\d\d\d|[1-9]\d\d|[1-9]?\d)",
          "required" : true
        },
        "value" : {
          "type" : { "$ref" : "SourceRequest" },
          "required" : true
        }
      }
    }

    {
      "id" : "PatchRemoveSourceAddress",
      "extends" : {"$ref" : "PatchOperation"},
      "properties" : {
        "remove" : {
          "type" : "string",
          "pattern" :
          "/srcs/((ipv4:(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-
5]|2[0-4]\d|1\d\d|[1-9]?\d)){3})|(ipv6:([0-9A-Fa-f]{1,4})(:[0-9A-Fa-
f]{1,4}){7}))",
          "required" : true
        }
      }
    }
```

A2.2.4. Response

After successfully updating the resouce, HTTP returns response code 200 (OK). The response contains the full description of the updated resource, using the data format of a JSON object described by the http://www.envision-project/org/cina/multicast-schema#AllocateResponse JSON schema.

A2.2.5. Capabilities

The update operation inherits the capabilities of its target resource. In the case when the parent multicast resource is using a multicaster component the PatchAddSourceAddress and PatchEditSourceAddress operations of the UpdateRequest will use the http://www.envision-project.org/cina/multicaster-schema#SourceRequest in their value field.

A2.2.6. Example

```
PATCH /multicast/12345678 HTTP/1.1
Host: custom.cina.example.com
Content-Length: [TODO]
Content-Type:   application/json-patch;
                profile=http://envision-
project.org/cina/multicaster-schema#UpdateRequest
Accept:  application/json;
          profile=http://envision-project.org/cina/multicaster-
schema#AllocateResponse
```

```
   [
     { "replace" : "/lease-time", "value" : "2011-12-19T17:44:57" },
     { "add" : "/srcs",
       "value" : {
         "src-address" : "123.123.123.14",
         "src-port" : 1111,
         "destination-port" : 1111,
         "max-bitrate" : 256
       }
     },
     { "replace" : "/srcs/ipv4:123.123.123.12:1111",
       "value" : {
         "src-address" : "123.123.123.12",
         "src-port" : 1111,
         "destination-port" : 1111,
         "max-bitrate" : 1024
       }
     }
   ]


   HTTP/1.1 200 OK
   Location: http://custom.cina.example.com/multicast/12345678
   Content-Type: application/json;
             profile=http://envision-project.org/cina/multicaster-
schema#AllocateResponse

   {
     "id" : "12345678",
     "multicast-address" : "ipv4:123.123.123.11",
     "start-time" : "2011-12-19T16:34:57",
     "lease-time" : "2011-12-19T17:44:57",
     "current-time" : "2011-12-19T17:20:11",
     "srcs" : [
       {
         "src-address" : "123.123.123.12",
         "src-port" : 1111,
         "destination-port" : 1111,
         "multicaster-port" : 2222
       },
       {
         "src-address" : "123.123.123.13",
         "src-port" : 1111,
         "destination-port" : 1111,
         "multicaster-port" : 2223
       },
       {
         "src-address" : "123.123.123.14",
         "src-port" : 1111,
         "destination-port" : 1111,
         "multicaster-port" : 2224
       }
     ],
     "multicaster-address" : "ipv4:123.123.123.10",
     "multicast-source-address" : "ipv4:123.123.123.10"
   }
```

A2.3. Multicast Deallocate Address

A2.3.1. Media Type

None.

A2.3.2. HTTP Method

This method deletes a multicast resource using the HTTP DELETE method.

A2.3.3. Input Parameters

None.

A2.3.4. Response

After successfully deleting the resouce, HTTP returns response code 204 (No Content).

A2.3.5. Capabilities

None.

A2.3.6. Example

```
DELETE /multicast/123455678 HTTP/1.1
Host: custom.cina.example.com


HTTP/1.1 204 No Content
```

A2.4. Resource Entry in Information Resource Directory

The resource entry for the multicast service SHOULD explicitly list all the schemas that the CINA server supports. Below there is an example of a CINA server providing access to the multicast service directly and through a multicaster component. The particular choice for mapping URIs to service flavours is left to the operator.

```
{
  "uri" : "http://custom.cina.example.com/multicast",
  "media-types" : [
    "application/json;profile=http://www.envision-
project.org/cina/multicast-schema#AllocateResponse",
    "application/json;profile=http://www.envision-
project.org/cina/multicaster-schema#AllocateResponse"
  ],
  "accepts" : [
    "application/json;profile=http://www.envision-
project.org/cina/multicast-schema#AllocateRequest",
    "application/json;profile=http://www.envision-
project.org/cina/multicaster-schema#AllocateRequest"
  ],
  "capabilities" : {
```

```
      "multicast-address-family" : [ "ipv4", "ipv6" ],
      "IGMP-version" : "IGMPv3"
    }
  }
```

APPENDIX B : COST TABLE SPECIFICATIONS

B. Service Cost Table

The Service Cost Table resource lists the costs that are associated with the allocation a service instance and the available capacity for a single PID as defined by the CINA Server.

The operational cost gives a measure on how loaded the servers of the NSP at a respective PID are. Lower values indicate a higher preference for a service instantiation. The operational costs MUST be normalized per domain, to allow clients a comparison of different values.

The capacity is a measure on how much bandwidth could be allocated by the client in the respective PID. If the capacity is 0 the service is currently not available for the client. It is given in the metric of Kbyte per second.

B.1 Media Type

The media type is "application/alto-servicecosttable+json".

B.2 HTTP Method

This resource is requested using the HTTP GET method.

B.3 Input Parameters

None.

B.4 Capabilities

None.

B.5 Response

The returned InfoResourceEntity object has "data" member of type InfoResourceServiceCostTable:

```
object CostTable {
  JSONNumber operational-cost;
  JSONNumber capacity;
};

object {
  JSONString  map-vtag;
  CostTable [PIDName]<0...*>;
} InfoResourceServiceCostTable;
```

with members:

- map-vtag
  The Version Tag of the Network Map used to generate the PID names.

- operational-cost
  The operational cost associated for a single instantiation of a service in the PID as a relative measure for a single domain
- capacity
  The currently available capacity in terms of bandwidth that can be allocated in a single PID for the client. The metric is Kbytes per second

  CostTable is a JSON object with each member representing a single PID; the name for a member is the PIDName string identifying the corresponding PID. For each PID, a PIDCosts object denotes the associated operational cost and the available capacity.

B.6 Example

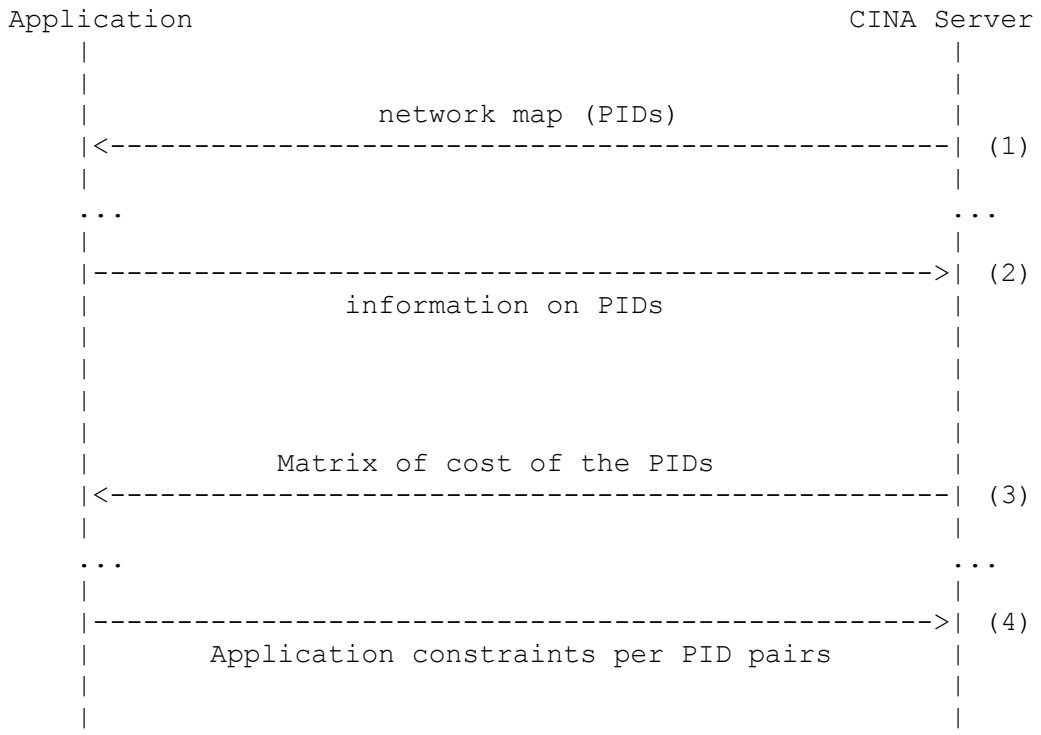
```
GET /highcapacitynode HTTP/1.1
  Host: networkservices.cina.alto.example.com
  Accept: application/alto-servicecosttable+json,
          application/alto-error+json

HTTP/1.1 200 OK
   Content-Length: [TODO]
   Content-Type: application/alto-costmap+json

   {
     "meta" : {},
     "data" : {
       "map-vtag"  : "1266506139",
       "PID1": {
          "operational-cost": 10,
          "capacity": 700,
        }
       "PID2": {
         "operational-cost": 20,
         "capacity": 3000,
        }
       "PID3": {
          "operational-cost": 14,
          "capacity": 0,
        }
     }
   }
```

APPENDIX C : FOOTPRINT MAP & CONSTRAINT MAP SPECIFICATIONS

The framework is based on the providing of an interface to the application to complement the network and the cost map.



Extension framework

(1) The application downloads the network map.

(2) New interface: The application complements the network map of the server with its own view of the map.

(3) The application downloads the cost map of the network.

(4) New interface: The application informs the server of its own network constraints.

4. Requirements

This section summarizes the requirements.

Retro compatibility:

- The extension MUST not modify the Protocol;
- The extension MUST preserve the RESTful design of the ALTO protocol;

Interoperability and internetworking:

- The CINA server MUST indicate the support (or not) of the extension in its Information resource Directory.
- The CINA client MUST only complement the network and the cost map previously downloaded by the server: An element of information provided by the application to a CINA server MUST complement an information resource (e.g. a PID) received from this server;
- The CINA client MUST only complement elements of information (e.g. a PID) of a network map or a cost map previously downloaded from the server;
- The CINA client is not oblige to provide its information;

Application level:
- The application MUST be able to communicate to the server its view of a PID (number of sessions, volume of traffic, number of end-points...).
- The application MUST be able to communicate to the server its forecast of traffic (peak hours);

5. Specification

5.1. Application Footprint Map

The Footprint Map provides information on the distribution of the Application Endpoints over a Network map. The entries of a Footprint Map are taken in the set of the PIDs of the Network map. Typically the CINA Client provides information on a restrained set of PIDs corresponding to the application view.

5.1.1. Footprint Map attributes

The Footprint Map attributes include:
- FootprintType : identifies what the Footprint represents. The types defined in this document include "number" indicating the number of endpoints and "sessions" indicating the number of sessions..
- FootprintMode : identifies how the footprint should be interpreted. The mode defined in this document is numerical.;

```
object {
  FootprintMode footprint-modes <0..*>;
  FootprintType footprint-types <0..*>;
} FootprintMapCapability;
```

5.1.2. Footprint Map structure

The media type is "application/alto-appfootprint+json".
This resource is requested using the HTTP POST method.
Input parameters are at the format of the media type "application/alto-appfootprintparams+json". It corresponds to the JSON object InfoResourceAppFootprintMap:

```
 Object {
  JSONNumber  [EndpointType]<0..*>;
} FootprintPIDinfo;

# FootprintMapData is an array indexed by the PID name.
# The encoding of an instance foo of type FootprintMapData
# which describes the PID pid1, pid2 and pid3 is
# "foo": { "pid1":{ info }, "pid2":{ info }, "pid3":{ info } }
# The encoding of 37 clients and 1 source in the pid1 is
# e.g. "pid1": { "client" : 37, "source" : 1} .

object {
  FootprintPIDinfo [pidname]<0..*>;
} FootprintMapData;

object {
```

```
        FootprintMode       footprint-mode;
        FootprintType       fooprint-type;
        JSONString          map-vtag;
        FootprintMapData map;
      } InfoResourceAppFootprintMap;
```

The EndpointType is a string defining the type of the Endpoint: "source", "client", "cache".

The CINA client MUST include in the request only PIDs belonging to the network map identified by the field map-vtag.

The example below shows the upload of the number of sessions of the application in the PID 2 and 3 of the network map "1266506139" previously downloaded.

```
#
# Request
#
POST /app/footprint/map HTTP/1.1
Host: custom.alto.example.com
Content-Type: application/alto-appfootprintparams+json
Accept: application/alto-appfootprint+json

   {
     "footprint-mode" : "numerical",
     "footprint-type" : "number",
     "map-vtag" : "1266506139",
     "map" : {
       "PID2": { "client" : 37, "source" : 1},
       "PID3": { "client" : 51 }
     }
  }

#
# Response
#
HTTP/1.1 200 OK
Content-Length: [TODO]
Content-Type: application/alto-appfootprint+json

{
  "meta" : {},
  "data" : {},
}
```

The footprint map must give the repartition of the different kinds of end-points over the set of the PID defined by the Network Map.

5.2. Constraint Map

The application constraint information service (refinement of the cost map by the application) provides information on the constraints of the data distribution between pairs of PIDs of a network map previously received from the server.

The entries of a Constraint Map are taken in the set of the PIDs of the Network map. Typically the CINA Client provides information on a estrained set of PIDs corresponding to the application view. As an example this may correspond to the PIDs where the application faces bad QoE.

5.2.1. Constraint Map attributes

The Constraint Map attributes include:

- ConstraintType : identifies what the constraint represents. The types defined in this document include "traffic-Gbps" indicating the traffic between network PIDs in Gbps and "RTT-ms" indicating the RTT between Endpoints in network PID in ms.

- ConstraintMode : identifies how the constraint should be interpreted. The mode defined in this document is "numerical".

- ConstraintPeriod: specifies the period of time during which the data characterizes the application constraints on the network.This period could in the near past, the present or the future. It is defined as an array of 2 numbers: [startlimit, endlimit]. The format of each number is YYYYMMDDHHMM.

```
object {
  ConstraintMode Constraint-modes <0..*>;
  ConstraintType Constraint-types <0..*>;
  ConstraintPeriod Constraint-periods ;
} ConstraintMapCapability;
```

5.2.2. Constraint Map structure

The media type is "application/alto-appconstraint+json".
This resource is requested using the HTTP POST method.
Input parameters are at the format of the media type "application/alto-appconstraintparams+json". It corresponds to the JSON object InfoResourceConstraintMap.

The CINA client MUST include in the request only PIDs belonging to the network map identified by the field map-vtag.

The example below shows the upload of the traffic constraints of the application between the PID1, 2 and 3 of the network map "1266506139" previously downloaded (section 7.7.2.2.6 of[I-D.ietf-alto-protocol]).

```
#
# Request
#
POST /app/constraint/map HTTP/1.1
Host: custom.alto.example.com
Content-Type: application/alto-appconstraintparams+json
Accept: application/alto-appconstraint+json

{
  "cost-mode" : "numerical",
  "cost-type" : "traffic_Gbps",
```

```
    "map-vtag" : "1266506139",
    "constraint-period": [201106101245, 201106101300]
      "map" : {
        "PID1": { "PID1": 3,  "PID2": 100,  "PID3": 80 }
        "PID2": { "PID1": 20,  "PID2": 5,  "PID3": 150 }
      }
  }

  #
  # Response
  #
  HTTP/1.1 200 OK
  Content-Length: [TODO]
  Content-Type: application/alto-appconstraint+json

  {
    "meta" : {},
    "data" : {},
  }
```

5.2.3. Constraint Information

A constraint is an information related to forecast or estimation of network performance like the throughput or the RTT between a PID pair.

The structure of this information should be defined:

- ConstraintType : identifies what the constraint represent
- ConstraintMode : specifies how the constraints should be interpreted
- ConstraintPeriod : specifies the period during which these constraints are to be considered

5.3. Errors

The exchange must conform the ALTO protocols and the handling of the error specified in the section 7.4. of [I-D.ietf-alto-protocol].The extension does not introduce new error code.

6. App2Srv Information Call Flows

This section describes the inter networking between an CINA server and the CINA client of the application.

6.1. App2Srv Extension Support

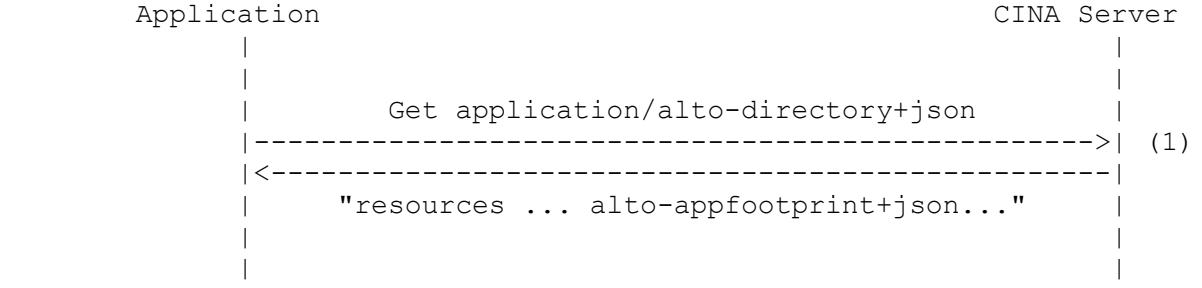
The server MUST expose the "media-types" of the App2Srv in its Information Resource Directory:

```
  HTTP/1.1 200 OK
  Content-Length: [TODO]
  Content-Type: application/alto-directory+json
  {
    "resources" : [
      ...
      {
        "uri" : "http://custom.alto.example.com/app/footprint/map",
```

```
            "media-types" : ["application/alto-appfootprintmap+json"],
            "accepts" : [ "application/alto-appfootprintparams+json"],
            "capabilities" : {
                "footprint-modes" : ["numerical"],
                "footprint-types" : ["number",?session?]
            }
          }
          ...
          {
            "uri"                                                    :
"http://custom.alto.example.com/app/constraint/map",
            "media-types"        :        [        "application/alto-
appconstraintmap+json"],
            "accepts" : [ "application/alto-appconstraintparams+json"],
           "capabilities" : {
                "constraint-modes" : ["numerical"],
                "constraint-types" : ["Throughput-Gbps",?RTT-ms?],
                "constraint-periods" : ["startlimit",?endlimit?],
           }
          }
          ...
        ]
      }
```

Figure 3: Example of Directory

The Application MUST download this server resource directory and check that the server support the App2Srv Extension.



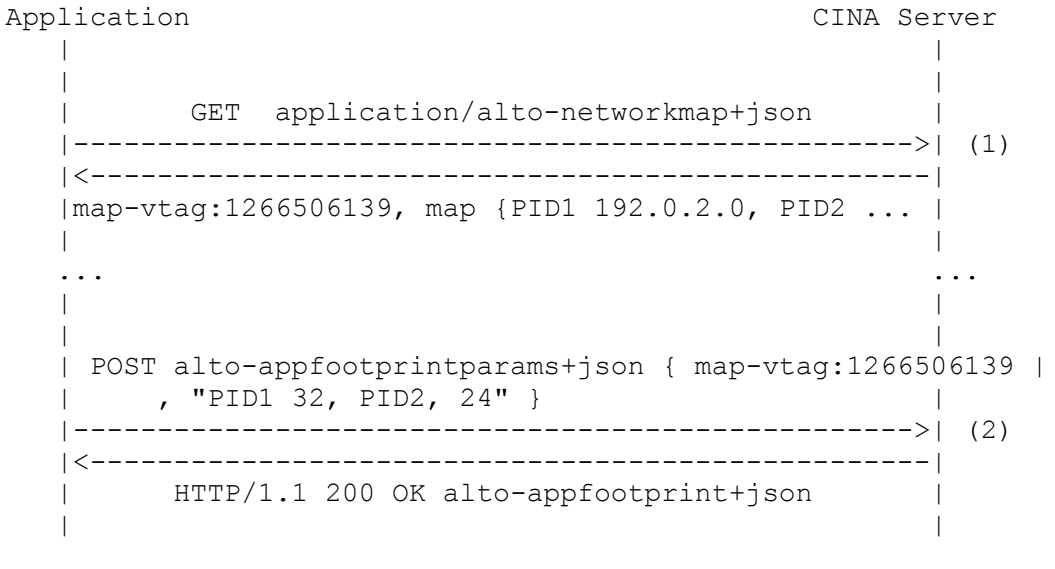
(1) The application downloads the Server Information Directory and checks that the server supports the "media-types" of the App2srv extension

6.2. Footprint Information Service

The Footprint Information Service provides an CINA Server with a way to receive additional information on the PIDs of its network map from a CINA client .

The application MUST not query the Footprint Information Service if the server does not expose the service in its Information resource directory.

N.B.: The server may add some policy in its Information resource directory to limit the usage of the Footprint Information Service (volume, frequency, ...). Some of them should be standardized to ease the interoperability.



Footprint Information

(1) The application downloads the network map identified by map-vtag="1266506139".

(2) Later, the application uploads information on the PID of this instance of the map.

As an example, the application may inform the server of the number (numeric) of sessions, of sources and of sinks. The PIDs MUST belong to the network map previously downloaded and identified by the same map-vtag value.

6.3. Constraint Information Service

The Application Constraint Information Service provides an CINA Server with a way to receive application constraint between its PIDs. There are various kind of constraints like traffic forecast and network QoS constraints between PID pairs.

The application MUST not query the Constraint Information Service if the server does not expose the service in its Information Resource Directory.

The Application downloads a costmap matrix and uploads its constraints for a subset of this matrix.