

Report on live-streaming infrastructure

4th February 2004

Leader: Alessandro Falaschi

Contributors: Franca Fiumana, Michal Krsek, Egon Verharen, Carlo Sganga

Contents

1	Architectural model	3
2	Technology survey	4
2.1	Protocols	4
2.2	Media Formats	5
2.3	Servers and splitters	6
2.4	Encoders	7
2.5	Viewers	7
3	Design alternatives and solutions	7
3.1	Some terminology	7
3.2	Design alternatives	8
3.2.1	Experimented solutions	9
3.2.1.1	Static topology	10
3.2.1.2	Use of RIPE <i>whois</i> DB	10
3.2.1.3	Multicast distribution	10
3.3	Design goals	12
3.4	Analysis of goals and alternatives	12
3.4.1	<i>Independence on media format and/or streaming protocol</i>	12
3.4.2	<i>Use of regional facilities OR NOT</i>	12
3.4.3	<i>Use of multicast</i>	14
3.4.4	<i>Allowance for an easy growth</i>	15
3.5	Architecture definition	15
3.5.1	Operational phases	15
3.5.2	CDN state	15
3.5.3	Control primitives	16
3.5.4	Requirements	16
3.6	Operations	16
3.6.1	Registration of capability and features by NREN and surrogate candidates	16
3.6.2	Registration of program URI with the portal	17
3.6.3	Selection of a program by clients which visit the announcement portal	17
3.6.4	CDN service setup	17
3.6.5	Procedures to be undertaken at the RRDM for distribution set up	17
3.6.6	Request handling by the nodes, eventually comprising recursion	19
3.6.7	Tear-down of the distribution setup	20

4	Architecture feasibility study	21
4.1	Entities interfaces and components	21
4.1.1	RRDM inner components	22
4.1.2	CDN node inner components	22
4.2	Protocol and architecture definition	22
4.2.1	XML-RPC encoding	23
4.2.2	Messages and methods	23
4.2.3	Message flows examples	23
4.2.3.1	Registration	23
4.2.3.2	Relay Setup	25
4.2.4	Choice of the streaming server and programming language for pilot experimentation	26
4.3	Advanced features	26
4.3.1	Concurrent processing	26
4.3.1.1	Shared state and IPC	26
4.3.1.2	Locking	27
4.3.2	Distributed health information	27
4.3.2.1	Aging of health information	27
4.3.3	Adaptation to network conditions and node load	27
4.3.4	Proximity measures and <i>binning</i>	28
4.3.5	Use of whois servers	28
4.3.6	Nodes autoconfiguration	28
4.3.7	Multicast support	28
4.3.8	Authentication	29
5	Development of the experimental setup	29
5.1	Requirements	29
5.2	Implementation details and interfaces	29
5.3	Configuration of RRDM and NODE entities	29
5.4	CDN service invocation	30
5.5	Porting of adaptation layer to other platforms	31
6	Experimentation results	31
6.1	Demo page	31
6.2	Test CDN	31
7	Appendices	32
7.1	Technology survey about streaming devices	32
7.2	DNS redirection mechanisms and motivations for not to use it	34
7.3	Address sets, footprints, and metric	34
7.3.1	DNS metric	35
7.3.2	IP metric	35
7.3.3	Comparison of the two techniques	36
7.4	Short overview of XML-RPC	36
7.4.1	Definition	36
7.4.2	Request example	36
7.4.3	Response example	36
7.4.4	Payload format	37
7.4.5	Response format	37
7.4.6	Fault example	37
7.5	Adaptation layer synopsis	37

Abstract

This study investigates viable solutions for large scale provision and delivery of live content. It focuses on concept and design architecture, usage of multiple servers, stream splitters, unicast-to-multicast gateways, and other transport mechanisms, along with a system for using one or more of the analyzed solutions, for efficiently deliver live content to a broad international audience.

The document is structured as follows. In § 1 a general architectural model is given, and a common terminology identified. A technology survey about existing specifications, protocols, media formats, devices and architectures is provided in § 2 and Annex 7.1. Design alternatives and architecture proposals are outlined in § 3, and a solution is identified and described. A feasibility study for the proposed solution is drawn in § 4. A report about early development of the setup is given in § 5, and results of pilot experimentation presented in § 6.

1 Architectural model

The terminology adopted in this section is borrowed from that used within the CDI IETF WG¹, although that WG was mainly concerned about *interworking*, while our topic is creation of a single transnational CONTENT DELIVERY NETWORK (CDN) instead.

Large scale delivery of live content is possible, in the more general way, by use of a set of *surrogate* nodes, which *do split* streaming content along the way from source to destinations, performing the so called ALM (*application level multicast*), adding value to the CDN in terms of *scale* (many more destinations can be reached) and *reach* (surrogates placed near destinations avoid network congestion). Where and if native multicast routing can be considered affordable enough, it will be used instead of ALM; yet, the analysis will follow the more general approach, which decomposes a CDN in the following parts:

- **content delivery infrastructure:** is the set of surrogate servers which deliver copies of content to set of users. Sub-section 2.3 surveys some of the more widespread products and solutions, and Appendix 7.1 reports about many other devices and related products;
- **distribution infrastructure:** is the mechanism that move content from the origin server to surrogates, and it needs to be specified in terms of
 - *advertising* - each surrogate must communicate about its own status, footprint and capability informations, to the request routing infrastructure;
 - *replication* - pertains to the way for the content to propagate in between surrogates, either by unicast or multicast, and by push or pull approaches;
 - *signaling* - node by node delivery must be directed, after the outcome of decisions taken by the request routing function. An appropriate control architecture (centralized *vs* distributed) must be identified, protocols and metadata must be defined, or existing ones must be validated, or adapted;

unfortunately, many of the existing delivery devices do not agree on common distribution protocols, thus forcing to define in this study a new, ad-hoc solution.

- **request routing infrastructure:** is the mechanism that move a client toward a rendezvous with a surrogate. Design alternatives are discussed in §3.2, where a clear distinction is made in between
 - a *network-level solution*, such as *DNS redirection* (see § 7.2 for details) and application level switches, and
 - an *application level solution*, based on a *announcement portal*. In this case, the portal knows the address of the client, and can re-write the contact part of the content meta-data, indicating the best surrogate server.

¹The IETF *Content Distribution Internetworking* WG has been closed on June 2003, by releasing the informational RFC 3570, “Content Distribution Internetworking (CDI) Scenarios”.

Completely distributed solutions, such as those afforded by peer-to-peer file sharing protocols, seems not to be appropriate here, a cause of the needs of logging, accounting, and authorization.

After having specified *who* is charged to perform request routing, there still needs to be defined *how to*

- select which is the surrogate *best suited* for delivery of content to a given client. This can be based on advertising from surrogates, metric information in between them, client address, and distribution setup outcome. Such a decision should perform some kind of constrained joint optimization, such as minimization of bandwidth usage, surrogate load, and number of hops. Metric information for the choice of the best surrogates chain can be
 - * CPU load and bandwidth availability, reported by advertising;
 - * latency and packet loss in between surrogates, and in between the client and a set of surrogate candidates, obtained by active probing.
- interact with the distribution advertising and signaling parts, in order to collect information from nodes, and direct the distribution setup.

An alternative to all the above, could be to make no decision at all, and rely on static routing configuration, based on known network topology and administration. Such a solution can be used in first trials, but seems of difficult maintenance in the long term.

- **accounting infrastructure:** is the collection and tracking activity about request routing, distribution, and delivery functions, actuated within the CDN. It basically must allow to report about overall audience of the same content, and about individual surrogate nodes activity. It can be done by known protocols (FTP, SNMP, SIP), but a common data format, as well as a collection and distribution authority, must be defined.

Security considerations As the overall architecture seems to be prone to different kinds of attacks, appropriate security requirements must be identified, such as (at least) authentication of advertising, signaling and accounting components.

2 Technology survey

Streaming is based on media formats, control and transport protocols, and metadata. In a distributed architecture, all of these aspects must be well managed by sources, servers, and players. This section gives a brief survey of the protocols and media formats used, and very briefly outlines the prevalent devices, products and applications, while Appendix 7.1 gives a wider survey about related implementations.

Although a complete cross-compatibility matrix does not exist, full interoperation in between different brands of streaming technologies is difficult to achieve, hindering the development of a vendor-independent streaming architecture. For this reason, in Sect. 3 an *overlay* CDN architecture will be proposed.

2.1 Protocols

They define the rules governing how a multimedia file and/or live content can be distributed.

RTP is defined in RFC 1889, and indicates how encoded media is packetized over UDP: whenever a new media format/container appear, new specifications are published within the AVT IETF WG, defining packetization rules. Basic control on transmission is provided by RTCP messages, exchanged among source and destination.

SDP is defined in RFC 2327, and represents a form of metadata description, about RTP streams which contribute to a multimedia session, as any media (audio, video) is distributed by a different RTP stream, with its own encoding and parameters. Enhancements to SDP (SDPng) are being defined within the MMUSIC IETF WG.

RTSP is defined in RFC 2326, and is the control protocol by which users can request to receive multimedia content. It uses a request-response clear text paradigm, somehow similar to SMTP and HTTP, but servers do maintain state information about each client connection. Different kinds of requests are named *methods*, the more relevant within them being:

- DESCRIBE is sent by clients, asking for composition of an RTSP URI. The server responds by sending back the SDP which describes the kind of RTP media to be used;
- ANNOUNCE carries the SDP metadata which describes an RTSP URI media composition, now available or changed;
- SETUP is sent by clients, and creates session state information on a server, whose identifier is returned in the SETUP response . If the SDP returned in response to DESCRIBE contains media formats that the client can't receive, the latter should refrain from proceeding to the SETUP phase;
- PLAY is sent by clients, and starts transmission of the media associated with the URI referenced in the SETUP phase;
- TEARDOWN is sent by clients, and destroys session state information at the server.

RDT is a proprietary protocol from *RealNetworks*, and it is used in place of RTP, allowing the client to issue retransmission requests.

MMS is a proprietary protocol from MICROSOFT, used in place of RTSP, and which is undocumented.

HTTP is mainly used for requesting HTML web pages, but can be used for initiating *progressive download* of a multimedia file format, or to request out-of-band metadata information, such as an SDP description.

2.2 Media Formats

Despite of the name, the format is not really pertinent to media encoding, but to the way in which the encoding (often related to different tracks) is multiplexed in a file, and packetized in RTP flows; in such a sense, a format is known as a *container*. The more widely used formats are given below:

WindowsMedia (Microsoft) has free viewers for the major user platforms (Windows, Macintosh, Linux²), and is supported by some non-MS server. The container file format is known as *Advanced Systems Format*, and files have extension .WMV, .WMA, or .ASF, depending on whether they contain video and audio (WMV), only audio (WMA), or (ASF) if the content is not coded by Windows Media codecs.

RealMedia (Real Networks) has free viewers (practically for *any* user platform) and free basic server and encoder, which run on windows and *nix; advanced servers and producer stations are sold. Both the container format and the codecs are proprietary, and files have extension .RA, .RAM, .RM and .RPM.

QuickTime (Apple) has free viewers (Windows & Mac), free encoder (Mac), and free server (Mac, Linux). The content format has been the basis for MPEG-4 definition, is publicly documented, and allows to use third party codecs.

ISMA (Internet Streaming Media Alliance) is a non-profit corporation for definition of vertical specifications for Internet Streaming, based on MPEG-4 Audio and Visual Profiles and Levels for file format and storage, augmented with IETF RTP and RTSP transport specifications for streaming protocols and control, in order to create cross-vendor interoperability. There exists a free encoder/server/player suite (MPEG4IP) built by assembling open source projects and the Apple Darwin Streaming server, and which runs on Windows, *nix and Macintosh.

²<http://www.MPlayerHQ.hu/>

MPEG Previously of MPEG-4, two other specifications were defined by the Moving Picture Experts Group, known as MPEG-1 and MPEG-2, which deliver lower quality (or higher bandwidth) and poorer error concealment capabilities. Anyway, these formats are still widely used, for example in DVD and HDTV, and can be used in live streaming; anyhow, applications and devices which convert from MPEG-1 and MPEG-2 to MPEG-4 do exist.

H.261-H.263 Defined by ITU-T for ISDN videoconferencing, they have been adopted for Internet H.323 videoconferences, and MBONE loosely coupled conferences. As for the MPEG-1 and MPEG-2, the H.26* specifications mainly define the video encoding process, and RTP packetization rules are given by IETF RFC.

2.3 Servers and splitters

These devices are the origin of streaming content, and major vendors often are at the same time those that build codecs, and distribute players. But the real compatibility issue is given by control protocols and multiplexing container formats.

Often, servers can be chained by *relay* configurations, as for tunneling purposes. When the fan-out of a relay spans several destinations, the relay becomes a *splitter*. Splitting of media distribution allows to serve large set of users, and to deploy a CDN. Splitters are also sold by companies which are different from those who sell producer stations and players, and are more inclined to adhere to open, public standard.

WindowsMediaServer Handles HTTP, RTSP and MMS control protocols, but seems to be limited to the streaming of ASF/WMV/WMA container formats. Despite the freeness, you should be licensed for the OS, and interoperation setup in between servers requires an SDK. Finally, it runs only on Windows machines.

Darwin (Apple) is a free, open source server (running on Mac, Linux, Windows) which handles RTSP, and is capable of relaying QT (hinted) data from a broadcaster, from a multicast address, or from other servers, provided that an SDP stream description is available. Programming interface is publicly well documented, and Darwin is integrated with the ISMA-compliant MPEG4IP bundle. It can be directed to *relay* to its clients the live stream coming from another location; the relay works in *pull mode*, and must be set up before of serving requests from clients. Relay set up can be remotely directed, and is accomplished either by direct specification of the source, or by reception of an RTSP ANNOUNCE message, thus providing two operating modes:

- *Listen then Push*: the server waits for an RTSP ANNOUNCE message, containing SDP information for a session. At reception, the server initiates a DESCRIBE/SETUP/PLAY sequence, and starts to receive media streams, making them available to clients. If relay destinations are listed, a new ANNOUNCE is sent to destinations.
- *Pull then Push*: when the relay is enabled, an RTSP DESCRIBE/SETUP/PLAY sequence is sent toward the source, and received media streams are made available to clients. If relay destinations are listed, a new ANNOUNCE is sent to destinations.

The destination of the relay can be a multicast address.

Helix Universal Server (Real Networks) handles RTSP and MMS control protocols, is able to deliver the largest set (REAL, WM, QT, MPEG, *others*) of container formats, and reports about a four-fold delivery capability with respect to WMS. It accepts connections from WM, QT and MPEG4 encoders, and from any of these players. It runs on *nix and Windows platforms. A free server is available, bounded to 1 Mbps outbound traffic; full server as well specialized relay server do cost, in rough proportion to reachable audience. Stream splitting is obtained either by *push* or *pull* techniques. In pull mode, no data will flow until a client makes a request. SDP-based streaming is supported, although it seems that the RTSP ANNOUNCE method is not.

2.4 Encoders

Each of the streaming architectures listed in § 2.3 provides an ad-hoc content encoder, which feeds the first server of the distribution network. Besides of them, there exist other encoding tools, mainly developed in the open source community, such as FFmpeg and XVID encoders; than, other projects use them for live encoding and transmission, such as VideoLAN and MPEG4IP. The latter is fully integrated with Apple Darwin SS; conversely, use of VideoLAN is possible if it operates as a multicast transmitter, and the relevant SDP metadata is used for Apple Darwin SS source description.

Other commercial solutions can be found in Appendix 7.1.

2.5 Viewers

Each of the streaming architectures listed in § 2.3 provides an ad-hoc content player, i.e. Windows Media Player, Real Player, Quicktime player. Besides the proprietary formats, all of them are able to play some common format. Once again, players developed by the open source community (e.g. Mplayer) are a good interoperation example.

3 Design alternatives and solutions

This section analyzes design alternatives for the realization of a *Content Delivery Network* (CDN), and how these are accounted for by existing experimented solutions. Then, our design goals are explicitly stated, as for example scaling properties, independence from vendors, and use of pre-existing facilities. Discussion about alternatives and goals leads to a CDN definition, in terms of operations that it should be able to perform. The proposed solution uses a centralized control entity, named *Request Routing and Distribution Management* (RRDM), and is based on the existence of an announcement portal, where clients do find the surrogate URI best suited for delivery of live streaming contents.

3.1 Some terminology

Before of going too far, let us define some terms which will be used in the following, together with those introduced in § 1.

Relay or *splitter* or *surrogate* or *node*: these are all synonymous for a CDN node which is capable of pulling a live streaming content, and make it available for a number of downstream clients. When not acting as a relay for actual content, it can be indicated as surrogate or relay candidate.

Transport is the protocol/format that a relay does support, and completely defines the protocol used for controlling the media delivery (i.e. RTSP, MMS or multicast reception/transmission), the format/container by which the media is being transmitted, and the set of codecs used. In this study, ISMA, REAL and WM transports are considered.

Distribution chain is a set of one or more relays, all supporting the same transport, through which a live content program is streamed from a content provider to one or more destinations. Media flows from *upstream* to *downstream* nodes.

Program is a specific instance of the live content, and is uniquely identified by the URI to be referenced in order to get content, directly from the content provider. The program URI is the item registered in the announcement portal. The same program is managed by all the relays along a distribution chain.

Origin is the source from where content is received, and should be a streaming server itself. Any relay along a distribution chain receives the same program from the *Origin* offered by the previous hop, and offers the program mounted at its own origin, to all its downstreamers.

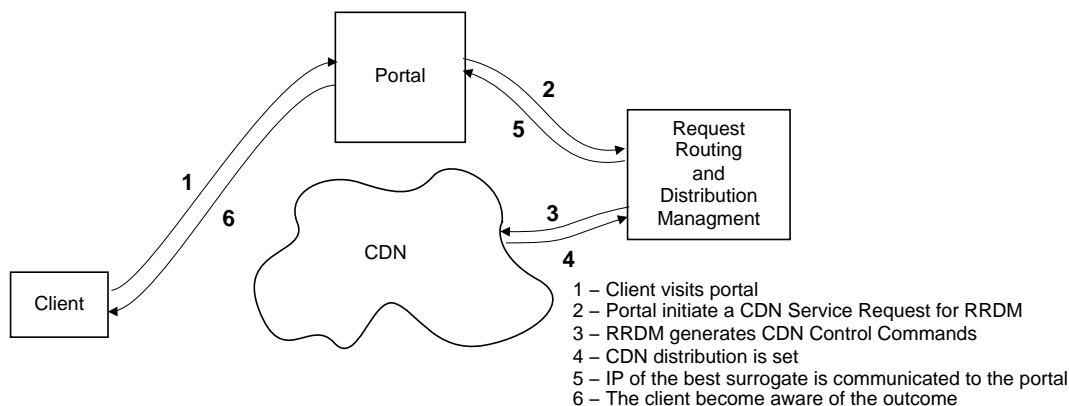


Figure 1: Request routing using application level redirection

First hop is the first relay in a distribution chain, i.e., the one which receives the content directly from the content provider.

Last hop is the last relay in a distribution chain, i.e., the one which actually delivers streaming content to some client. If a surrogate can be a first and last hop at the same time, then the CDN can be just one level deep. Sometimes a LH relay is referred to as *the* surrogate for a given client.

Footprint is the range of downstream addresses which a relay claims to serve. These addresses can either be expressed in form of an IP prefix, or a domain suffix (see § 7.3).

Transit relay is the relay whose footprint is not intended to specify (final) destination clients. Instead, the footprint only identifies other downstream relays (transit or not) which can be put along the distribution chain for a given streaming content. A transit relay cannot be a last hop, but can be a first hop.

Portal is the set of web pages where clients look for announced content. It can interact with the RRDM entity (see below), and return the clients the surrogate address located by the RRDM; otherwise, it can directly return the URI of the content, and defer the CDN service request to a DNS redirection.

Request routing and distribution management (RRDM) It is the entity which locates the *best* last hop surrogate, and directs first hop, transit, and last hop relays, to pull live streaming content. Its operations can be invoked by the portal, or by a modified DNS, according to the different design alternatives.

3.2 Design alternatives

Architectures can be defined after having decided about

1. *Who is charged of performing request routing.* It can be accomplished
 - (a) at the application level (fig. 1), by letting the announcement portal to direct a CDN Service Request toward the RRDM entity, and returning the *best* surrogate address to the requesting client;
 - (b) at the network (IP) level, by letting the announcement portal to return a fixed URI for the live streaming event, and delegating to a modified DNS (authoritative for the URI's domain) the task of emitting a CDN Service Request, and return to the requesting client the IP address of the *best* surrogate which can serve the program identified by the original URI. See fig. 2 for a pictorial sketch. But also see §7.2 for a discussion about good reasons for not proceeding in this way.

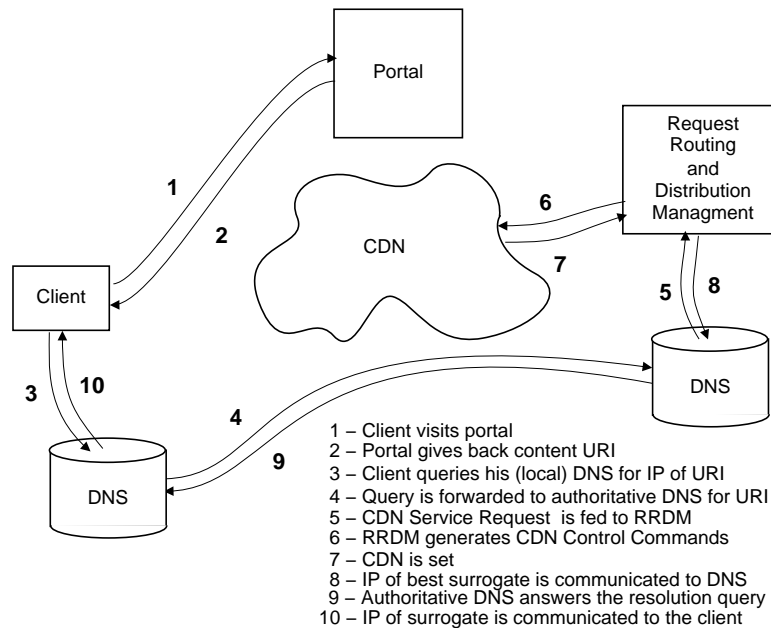


Figure 2: Request routing using DNS redirection

2. *How to collect informations from relay nodes*, pertaining which destinations they intend to serve (the *footprint*), which *transport* they can handle, and informations about load and health conditions, to be used for point 3. It can be accomplished
 - (a) by an ad-hoc information exchange protocol (fig. 3), to be deployed in between the candidate surrogate nodes, and the RRDM node;
 - (b) by usage of already existing protocols, such as BGP³, for advertising intended downstream address, and SNMP for querying about nodes status.
3. *How to select the best surrogate for a given client*. It can be accomplished by using
 - (a) dynamic informations provided by the surrogate candidates and collected by the RRDM entity, and describing surrogates footprint, their load conditions, and supported transport;
 - (b) dynamic informations about network conditions in between the client and a set of surrogate candidates, as the case described in footnote 9;
 - (c) static informations derived from a fixed CDN topology; eventually augmented by node status dynamic updates;
 - (d) static informations located in the RIPE WHOIS database⁴, to be queried about the address range where the client lies, and trying to locate a surrogate node in the same address range.

3.2.1 Experimented solutions

Already experimented architectures are now reviewed, and the limits of these solutions are evidenced.

³Each surrogate candidate should execute a BGP daemon, and be configured as a peer of the RRDM entity, whose BGP instance collects footprint advertisements.

⁴Actually RIPE is the Regional Internet Registry (RIR) only for European (and nearby) countries. If a CDN should be deployed on a wider basis, one should query also other RIRs, namely ARIN (north America), APNIC (Asia), LACNIC (south America), AfrinIC (Africa).

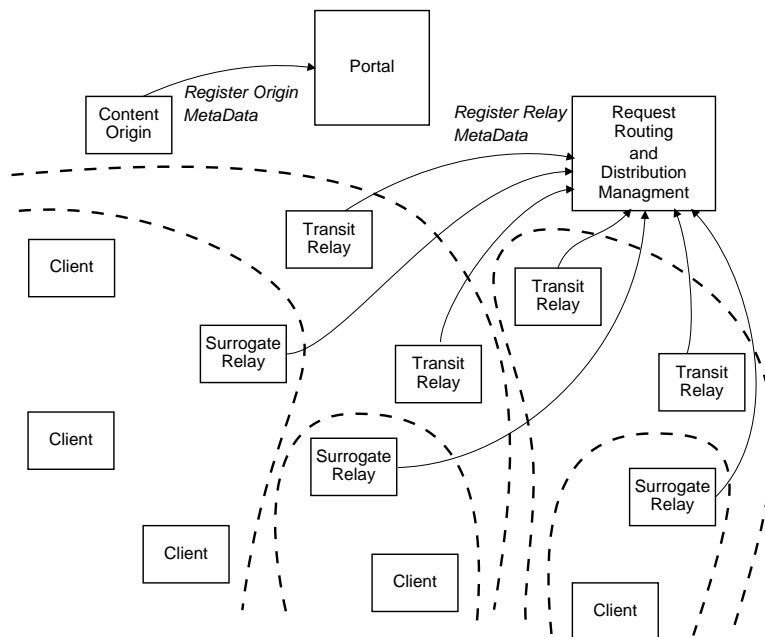


Figure 3: Registration phase with footprint information evidenced

3.2.1.1 Static topology In this case, all the nodes support the same *transport*, and their addresses and intended footprint are known well in advance of the CDN operational service. This solution has been successfully adopted, for instance, for netcasting of the RIPE meeting: a single WM broadcaster acted as a fixed content source, and a different WM splitter was used in each country willing to support the netcast. Content distribution is continuously, statically performed toward regional surrogates (see fig. 4), and an announce page is published, listing the URI of each of the regional surrogate. The person who runs the client will hopefully choose the surrogate pertaining its own country.

The resulting CDN is only one level deep, so that this architecture does not scale well, and it is tied to a particular *transport*. Adding a new content source or program, requires to manually set up a new feed/mount_point for every regional surrogate, and to publish a new announcement page.

3.2.1.2 Use of RIPE *whois* DB This case is similar to the previous one, but there may exist many more surrogates; when a live program starts, distribution is immediately activated toward all of them. Request routing can be performed either at the application or IP level, by applying the following procedure:

- use the client IP address (or the address of the client's DNS) for querying the RIPE *whois* DB, and find the administrative address block which contains the client address;
- use knowledge about surrogates location, and find the one within the same address block, or within a (larger) block which *contains* the one where the client is located.

This method scales better than the previous, although the resulting CDN still is only one level deep. Media distribution has to be set up in advance for every content source. Large address blocks may hide the real network topology, thus hindering the deployment of dedicated surrogates for bottlenecked remote LANS (see fig. 5).

3.2.1.3 Multicast distribution Dedicated distribution of media between a content source and a large number of surrogates can be avoided, if multicast transmission works well for all of these. Multicast does not need to be supported also in the LAN where the client is located, as the client will receive media from the *best* surrogate, behaving as a splitter. This solution is depicted in fig. 6, and a pilot version is used in

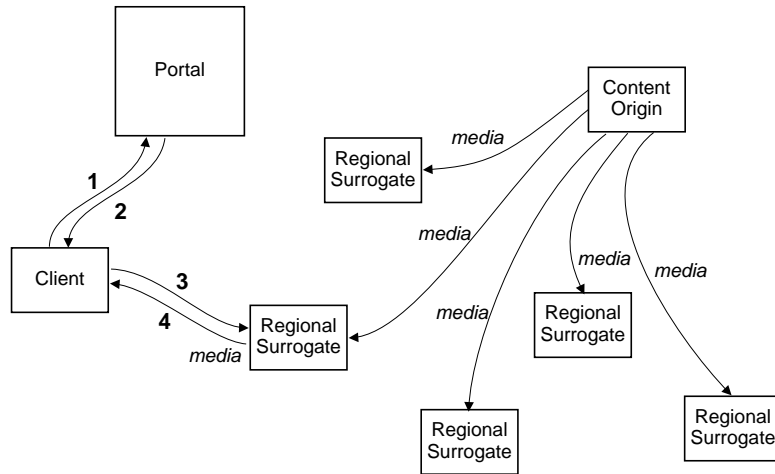


Figure 4: Use of static topology and fixed content distribution scheme

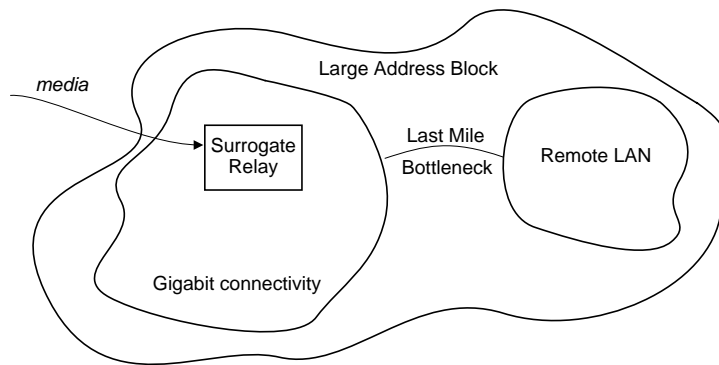


Figure 5: All the clients in the remote LAN will use the same Surrogate Relay, saturating the last mile bottleneck

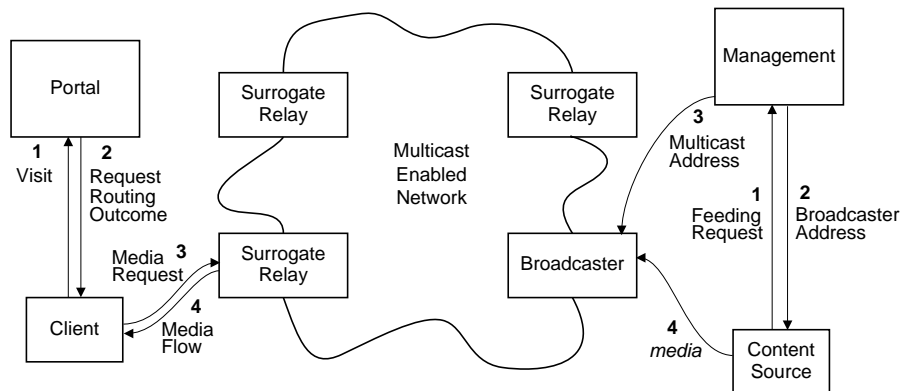


Figure 6: Use of a multicast enabled network

the Polish Optical Internet.

In their solution, the announcement portal uses knowledge about the client and the surrogate addresses, together with active probes about surrogates load and health, and probes about network conditions in between the client and surrogate candidates, for returning to the client the address of the *best* surrogate from where to receive a copy of the content. The content source interacts with a management node, which locates the broadcaster address. Moreover, the management node deals with multicast address allocations problems, and instructs the broadcaster about where to send new content.

This architecture does scale very well, but multicast failures may disrupt CDN service, and restrictions in multicast deployment may show up the same last mile problem described in fig. 5. On the contrary, widespread deployment of multicast routing will fall back to an MBONE-like distribution scheme⁵.

3.3 Design goals

This CDN architecture proposal tries to fulfill a series of goals, namely:

1. allow the use any of media format and/or streaming protocol, but do not rely on any particular choice;
2. make use of regional CDN facilities where they exists, but do not rely on their existence;
3. make use of multicast when possible, but work even without it;
4. allow the CDN to easily grow and scale, thus permitting new nodes to be freely added, typically within network sections where do exists an higher demand for the service.

Each of the outlined issue does imply some considerations about the architecture, which are given in the following § 3.4, together with considerations about the effects of different design alternatives.

3.4 Analysis of goals and alternatives

3.4.1 *Independence on media format and/or streaming protocol*

A new CDN *control protocol* must be defined, allowing the RRDM unit to direct the operation of nodes, by means of an implementation-agnostic layer; such a control layer then (see fig. 7) uses an adaptation commodity, that in every node, translates CDN control primitives to implementation-dependent commands. Examples of such commands are the set up of a relay, and queries about node load. An implication of this is the absence of constraints on content providers, but contents can be distributed only if a distribution chain supporting that particular *transport* can be set up in between source and destination⁶, and the destination is able to deal with the content *transport*.

When existing protocols as BGP, and SNMP, are used to acquire informations from nodes, details about *transport* mechanisms supported by nodes do not seems to be easily expressed, and here the solution seems to be either learning node's capability by trials, by ftp or email, or to fall back to a *transport-constrained* CND.

3.4.2 *Use of regional facilities OR NOT*

A CDN architecture may already be in place in some regions of the Internet, for instance operated by National REsearch Networks (NREN). In this case, we would like to make use of it, thus forcing the choice of a two level (at least) CDN topology. Request routing, which is directed by the RRDM entity, must be split in two phases: the first phase identifies the NREN CDN interface which is pertinent to the client, very likely by means of static routing information, and the second phase identifies the *best choice* surrogate for the client, and is performed by the NREN, by means of local policy and architecture (see left side of fig. 8). This decomposition seems not to be well suited for the case of a DNS based CDN⁷, so that the following

⁵In the MBONE, clients directly *see* multicast, and content reception is simply activated by joining to the relevant multicast group, to be known by (multicast) reception of announcement messages.

⁶This requirement could be relaxed if multicast connectivity would exists in between source and destination.

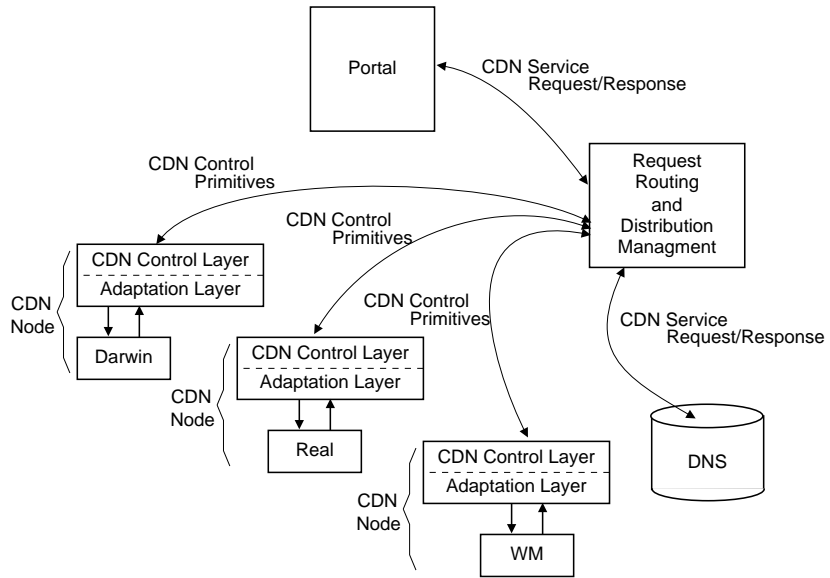


Figure 7: CDN support of multiple transports by definition of a CDN Control Protocol

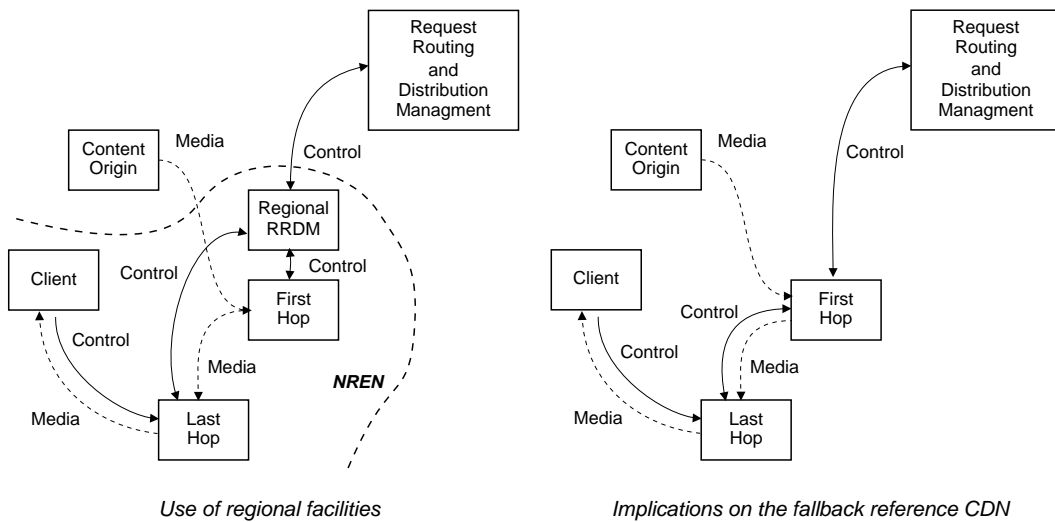


Figure 8: Requirement of using regional facilities constraints the reference CDN architecture

considerations are mainly appropriate for the case where the CDN service request is dealt at the application level, and managed by the portal.

Each half of the transaction do performs path set up operations, enabling (if not already done) relaying of content from source to the NREN, and from NREN to the local surrogates. Depending on the implementation, the NREN distribution setup procedure can terminate with

- a communication of the selected surrogate URI to the RRDM, which has to be returned to the entity (portal or DNS⁸) who initiated the procedure, *or*
- a communication of an HTTP URI located at the NREN premise, to which the portal (this case is not applicable to a DNS-CDN) should redirect the client who initiated the procedure, and which allows to perform more complex operations⁹ for the selection of the best choice surrogate for that particular client.

For an application level CDN, the client visits only one more page, which gives him the address of the best surrogate choice. This second page can be located at the portal itself, or at the NREN which is pertinent to the client.

If routing information available at the RRDM do not lists any regional facility for the client, a fall-back solution must be adopted. In this case, a reference CDN architecture must be defined, and nodes which desire to take part to the CDN must adhere to this specification. A registration phase (see fig. 3), in which surrogate candidates communicate to the RRDM the supported *transport* and intended *footprint*, must exists; such nodes must understand the CDN control protocol primitives, and execute the adaptation layer stack, for the translation of the RRDM-originated commands to locally executed operations.

In order to be consistent with the two-level approach outlined above, for any CDN request the RRDM must be able to identify a *first-hop* (FH) node, on the basis (for example) of footprint reachability information collected so far. The FH node will act similarly to an NREN interface (see right side of fig. 8), i.e. it will be directed to pull media from the source, and to recur the command, searching of further relays to be put along the distribution path, and located nearer to the client location. The outcome of the recursion must be reported back to the RRDM, communicated in turn to the portal or DNS, and then to the client. The candidate surrogates set, used by the FH for recursion, is directly communicated by the RRDM to the FH, and derived from registration informations collected by RRDM.

3.4.3 Use of multicast

The use of multicast for media distribution is possible either end-to-end, if every router in the path from source to destination is multicast enabled, or on different segments of the path, i.e. from source to FH, or within surrogates, or from the LH surrogate to the client.

Actually, a centralized control unit as the RRDM cannot directly probe if multicast transport is enabled in between couple of nodes: then, the choice of using multicast must be decided by each of the node pair, after having directly verified multicast operation in between of them. For this reason, every source and surrogate must be able to test multicast connectivity, and should fall-back to multicast transport if available. Another consequence is that, in order to be consistent with traditional multicast delivery, an SDP description of the media stream must be provided by the content source, or must be generated by the nodes which originate multicast transport. Finally, a chain of nodes which perform relaying by means of multicast makes no sense, and intermediate nodes should help the establishment of direct multicast transmission in between upstream and downstream nodes.

⁷In a DNS-based CDN, resolution is performed by the authoritative DNS for the URI's domain. BIND implementations should be severely hacked, for directing the authoritative DNS to delegate resolution at regional level.

⁸In the DNS case, only a network level (IP) resolution can be performed, and the *path* component of the URI at the surrogate, must agree with the one registered at the portal.

⁹An example is the PYTE system developed at CINECA, in which the *redirection page* that is visited by the client, contains objects (small images) located at the different surrogates available. Download time of objects, from each surrogate to the client, is measured by each surrogate, and reported to the redirection server. As a result, redirection decisions are performed on the basis of an objective throughput measure evaluated for all the surrogate-client pairs.

3.4.4 Allowance for an easy growth

This dictates the definition of a well tested package to be downloaded and executed by contributing nodes, and the definition of an authentication infrastructure by which trusting the more sensible data exchange phases.

3.5 Architecture definition

According to the analysis given in the previous sub-sections, the CDN architecture which will best suit our needs is based on a centralized control unit (the RRDM) and on a set of surrogate nodes, together with a control protocol for distribution management. Request routing is performed at the application level, by use of an announcement portal, and nodes can be equipped with splitters built by any vendor, by writing vendor-specific adaptation layers. Finally, CDN can easily scale by deployment of an arboreal distribution chain, and can make use of regional facilities where these exist.

Now, we proceed to better define the architecture, in terms of prototypical operations, CDN state, primitives and requirements.

3.5.1 Operational phases

The proposed CDN architecture can *operate* according the following phases:

- registration/advertising (fig. 3) of NREN and surrogate candidates with the RRDM;
- registration of *Program* URIs with the portal by content providers;
- selection of a program by clients which visit the announcement portal;
- feeding of a *CDN service request* from the portal (or DNS) to the RRDM (fig. 1/2);
- execution of procedures by the RRDM for distribution set up (see § 3.6.5);
- CDN control primitives handling by the surrogate nodes, eventually comprising recursion (see § 3.6.6);
- communication of the RRDM operations outcome to the portal (or the DNS)
- tear-down of the distribution setup at the end of the program schedule.

3.5.2 CDN state

As it will become clear after the more detailed analysis of the operational procedures given in § 3.6, the RRDM must maintain *state information* about the CDN deployment, namely:

- for each surrogate, a table listing the footprints (either IP prefixes or DNS suffixes) for which it will accept to operate, either as last-hop, or as a transit relay. The table is augmented by the transport(s) afforded by the node, and by its status information;
- for any *Program* which is currently scheduled as on-air
 - a list of all the surrogates which have been so far directed to relay the program, in any of the two possible roles;
 - a table which reports the resulting delivery topology.

3.5.3 Control primitives

Direction of CDN operation is based on the following *CDN control primitives*, invoked by the RRDM and by surrogates which perform recursion, actuated by the recipient party, and whose outcome is reported back to the initiating party:

- query about bandwidth usage, CPU load percentage, health conditions;
- request to set up a relay from a content_provider/surrogate_origin/multicast;
- request to tear-down an existing relay.

3.5.4 Requirements

Operations can be accomplished if NREN and surrogate candidates fulfill these *requirements*:

- ability to report about bandwidth usage, CPU load percentage, health conditions;
- ability to measure network conditions in between nodes, to be used together with health for load balancing and distribution optimization;
- ability to setup a relay for some useful transport, and to report about success/failure;
- presence of an adaptation layer which translates CDN control primitives, in commands understood by the embedded splitter device;
- capability of testing multicast routing effectiveness with respect to another node.

3.6 Operations

In the following, the analysis about the outlined operations, CDN state construction, primitives and requirements is deepened. Moreover, further considerations are given about drawbacks and advantages of design alternatives.

3.6.1 Registration of capability and features by NREN and surrogate candidates

NREN and surrogate candidates must be able to register their capabilities to the RRDM, such as

- transports which can be handled;
- direct footprint coverage, i.e. for which set of destination addresses the node accepts to be the last-hop surrogate
- indirect coverage, i.e. the set of destination addresses for which the node accepts to be a transit relay. If the node is an NREN interface, this should list all the assigned address space; otherwise, generic nodes can still propose themselves to be transit relays;
- supposed multicast capability;
- percent used bandwidth and load conditions.

These informations should be as well updated in re-registration turns, and a node should be able to de-register itself; moreover, RRDM should be able to query nodes about updated status information. The nodes load information can be used to perform load balancing optimizations.

In the case a BGP protocol is used for registration and refreshing of nodes capability, only IP-footprint information can be collected by the RRDM unit. At the same time, node load information can be retrieved by SNMP queries.

Footprint information is given in form of a set of addresses, which can be expressed either as DNS addresses suffixes, or as IP addresses prefixes; in both cases, a metric can be defined, so that different sets

of addresses can be ordered, from more specific to less specific, with respect to a given destination address, as shown in appendix 7.3.

Erroneous surrogate coverage registration can disrupt proper operation of the CDN. For this reason, registrations and updates should be possible only by properly authenticated nodes, or at least submission should be verified by checking of the submitter IP address and/or its DNS name.

3.6.2 Registration of program URI with the portal

The registration of programs with the portal, should provide:

- the transport which will be used for streaming;
- the SDP description of the program if multicast transport is envisaged;

3.6.3 Selection of a program by clients which visit the announcement portal

The portal

- should suggest which application the client must use for reception (it knows the transports used by the program), eventually listing multiple choices for different clients architectures;
- should offer links to where download the application needed to receive the program;
- must check the announced program time schedule, and eventually publish a link to an “Not on air” trailer;
- must allow the user to click on the requested program;
- must return to the client a page indicating how to proceed, either publishing the URI of the surrogate which will serve the program to the client, or redirecting the client to a page served by the pertinent NREN CDN interface;

If the CDN RRDM function operates under control of a DNS, the last point can not be actuated by the portal. This hinders interfacing to existing NREN CDNs.

3.6.4 CDN service setup

This phase involves asking the RRDM to find the *best* LH surrogate for the given client, and to setup distribution of the streamed program till there. The outcome of this process is communicated back to the requesting party, which can be:

- the portal: in this case (fig. 1), the outcome can either be the *LastHop* surrogate URI, or a web page URI located at a NREN point of contact. In either case, the result is communicated to the requesting client by the portal itself;
- the authoritative DNS for the domain of the URI published at the portal: in this case, the result can only be the IP address of the LH surrogate, and it is communicated back to the clients through its DNS. See fig. 2 for a pictorial sketch.

3.6.5 Procedures to be undertaken at the RRDM for distribution set up

When a CDN service request arrives at the RRDM, coming either from the portal or the DNS, the RRDM itself must

1. *if* the client is served by a NREN CDN¹⁰:

¹⁰As already outlined, NREN interfacing seems not to be applicable when a DNS-based redirection is used.

- (a) contact the NREN CDN interface, and ask to setup a distribution path in between the program source and a surrogate near to the client; then, wait for a response.
2. *otherwise*, if the client is *not* served by a NREN CDN¹¹:
- (a) check if a LH surrogate has already been activated for the selected program and the destination client address, and if it reports a load condition not exceeding some warning level. In the affirmative case, return the URI of that node, to be returned to the client, and stop.
 - (b) check if a LH surrogate set, and a FH transit relay set, both of which must support the transport of the program, do exist¹².
 - i. in the negative case, i.e. if either the LH or the FH relay set are empty, report the error condition to the requester and stop. If the requester was the portal, it should suggest a way to fix the problem: for example, hint to ask a network admin to set up a new relay, reporting about contact points for directions. If the RRDM function is invoked by a DNS, it should simulate a DNS error, such as a *not found* domain.
 - (c) check if some FH transit relay has already been activated for the program, and filter this list, retaining only those active relays which advertise a transit footprint containing the client address. If the resulting list is not empty:
 - i. sort this active relays set, by applying one of the metric evaluation methods discussed in 7.3, and find the *more specific* one. If more than a node results to be the preferred one, query all of them about load conditions, and choose *either*:
 - A. the least loaded *or*
 - B. the one which answers first
 - ii. ask to the selected, already active, FH relay, to set up media distribution toward a LH surrogate for the client. The request will contain the list of known LH addresses, footprints and load conditions, related to all the LH candidates which accept to serve the client¹³. Then, wait for a response.
 - (d) in the cases that neither exists an already active LH surrogate, nor an active transit relay, but as we have already checked, a LH candidates set, and a transit relay candidates set, do exist, we must select a FH relay for distribution, and direct it to recurse in finding a LH node, in a way similar to that just discussed in point 2(c). So:
 - i. extract a FH relay candidates set, by applying one of the metric evaluation methods, and find the *least-specific* (transport compatible) FH relay candidate for that client; if more than one preferred node is found, query all of them about load conditions, and choose *either*:
 - A. the least loaded *or*
 - B. the one which answers first
 - ii. ask to the selected FH relay candidate to set up media distribution from the content provider, toward a LH node for the client. The request will contain the list of known LH addresses, footprints and load conditions, related to all the LH candidates which accept to serve the client¹⁴. Then, wait for a response.
3. After that RRDM has delegated responsibility for distribution setup to the NREN CDN (point 1), to an already active relay (point 2(c)), or to a new FH relay (point 2(d)), it waits for a response from them, serving two purposes: being able to return a contact point to the client, and for collection of state information about the deployment of the CDN. For this reason, the returned response should contain:

¹¹This is the default behavior when DNS redirection is used.

¹²This condition can be relaxed, by requesting for the existence of only the last hop relay set, allowing for the deployment of a CDN which is only one level deep. In this case, the following point 2(c) does not apply.

¹³This list can be augmented, by adding a set of allowed down-stream *transit* relay candidates, which are *more specific* than the active transit relay to which this list is sent. This will allow to define more elaborate routing strategies, and to add further splitting stages in between first-hop relays and last-hop surrogates.

¹⁴See the previous note.

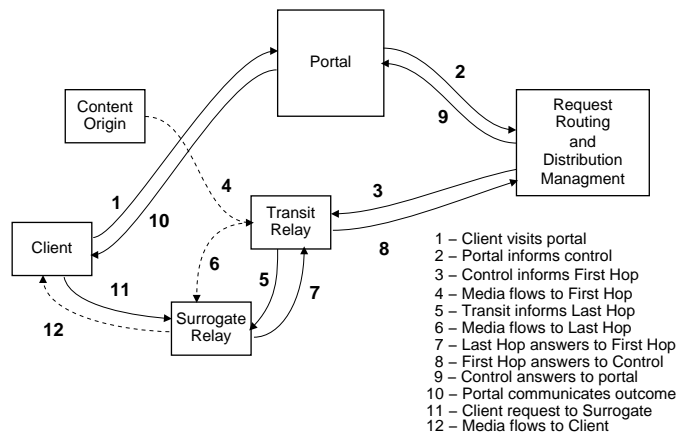


Figure 9: Steps involved for the handling of the first request

- (a) the URI of the LH surrogate, to be published in the page which will be returned to the client¹⁵, and the list of new relays which have been set up for the given program as a consequence of the request submitted by the RRDM, to be used for update CDN status information, or
- (b) the HTTP URI of a page located at the NREN, where the client's browser will be redirected, or
- (c) a negative answer, possibly due to overload conditions. If the negatively answering entity is the NREN CDN interface, report the failure to the client and stop. Otherwise, either iterate the process by making different choices about the delegated entity (a less specific active transit relay, or a more specific first hop, or an equally specific, redundant node), or perform other actions based on the actually returned response code.

3.6.6 Request handling by the nodes, eventually comprising recursion

This section describes request handling by contributing nodes, whether they have been invoked in the roles of FH, in-route, or LH relay. Definition of a common procedure for all these roles will allow to distribute an unique processing module to all the nodes who wish to take part to the CDN. When a request arrives to a relay node (phase 3 and 5 of fig. 9):

1. a sanity check if performed on the transport and on the final destination address: if they do not match against the advertised ones, a negative response is returned, signaling the mismatch error, and reporting about the correct values, to be interpreted as a re-registration¹⁶;
2. a check if performed about load conditions, and if they exceed some given limit, a negative response is returned;
3. this node can either be an (inactive) FH relay or the last-hop surrogate. In both cases, a pull relay is setup, from the up-stream origin indicated in the request, to the node itself (phase 4 and 6 of fig. 9);
 - (a) in the case of failure, a negative response is returned to the caller, and stop;
 - (b) in the case of success, initialize a list of new active relays for this program, by inserting this node address
4. if this node is a LH surrogate

¹⁵Or just the IP address of the LH relay, if the RRDM was invoked by a redirecting DNS. In such a case, the following point neither applies.

¹⁶This re-registration should be authenticated, for security reasons.

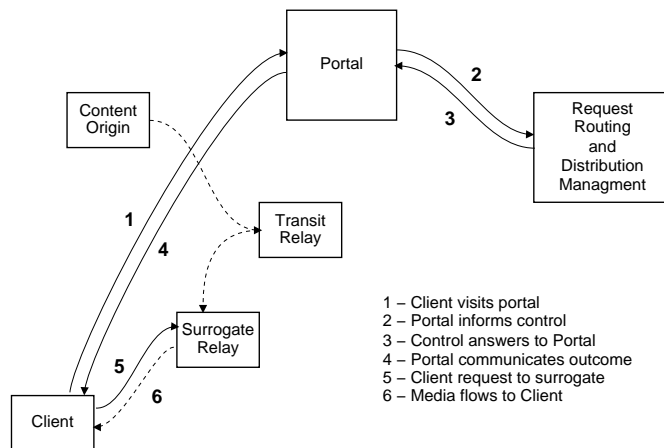


Figure 10: (fever) steps for subsequent requests

- (a) return a success completion code to the caller (phase 7 of fig. 9), reporting the URI to be used by the client for receiving the program, together with the list of new active relays for this program (in this case, just this node), and stop;
5. this node is a (now active) FH, or an in-route relay. Examine the list of LH candidates received from up-stream¹⁷, and sort them in specificity descending order. Then, for each of them:
- (a) recurse this procedure for the more specific LH candidate, asking it to set up a pull relay from here to there;
 - (b) if a positive response code is received, merge the received (from down-stream) list of the new activated relay for this program with the actual one¹⁸, and return the merged list to the caller, together with the received response code, and stop.
 - (c) if a negative response code is received, for example a cause of excessive load:
 - i. if it was the last candidate, return to the caller the list built so far, reporting the activated relays, together with the received response code, and stop;
 - ii. if it was not the *last* candidate, try again with the next, less specific one.

Fig. 10 shows the steps involved in a example case, in which requests come from the same footprint domain, after that the CDN has already been settled up.

Fig. 11 shows a possible CDN resulting topology, after that 5 clients have joined the program.

3.6.7 Tear-down of the distribution setup

When a live program ends, all the relays carrying it must be dismantled. The entity who knows about the programs time-schedule is the announcement portal, so that is its duty to send a *teardown request* toward the RRDM. This action should be performed also if the CDN is based on DNS redirection.

When RRDM receives the *teardown request*, it forwards the request to all the NREN interfaces/FH nodes which are relaying the program, and do not needs to wait for any type of response. The forwarded teardown

¹⁷If the list not only contains last-hop surrogates for the destination, but also candidate transit relays which are more specific than this one, more elaborate routing strategies can be applied. For instance, if the fan-out, or the load, of this node is exceeding some defined threshold, a new, more specific, transit relay can be tried to be activated, so that subsequent routing for similar destinations will be directed to that intermediate relay. This will result in augmenting the number of hops, and gracefully scale the architecture.

¹⁸In the simple, two level case outlined in the text, the merged list will only contain two nodes, namely this first-hop relay and the last-hop surrogate. If more elaborate routing strategies are applied, the down-stream entity which returns the list can be an in-route, more specific, transit relay, which has itself recursed in the search for a last-hop surrogate. In this more general case, the returned list can theoretically contain any number of newly activated nodes.

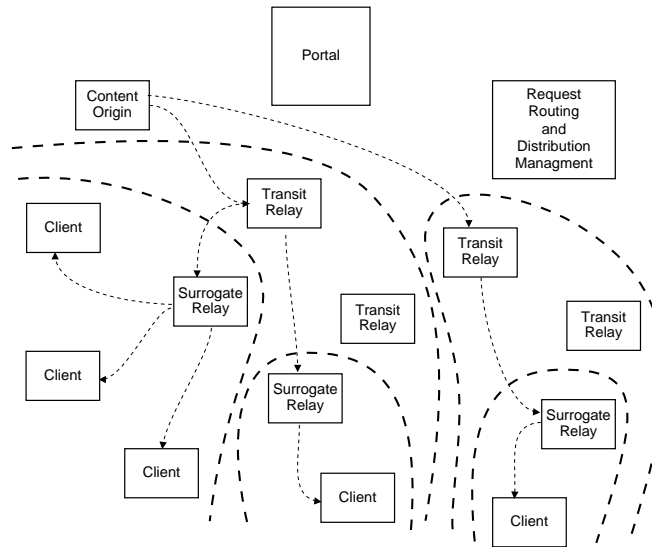


Figure 11: Final topology with footprints evidenced. Two transit relays remain idle

request also contains the list of downstream nodes (with respect to the FH/NREN) which have been reported to relay the same program.

The NREN interface, and the FH nodes, then start to ask the downstream nodes to stop the relaying of the program, eventually allowing for recursion. Teardown errors are not reported backward to the *Requester* (or the RRDM, or an upstream transit relay) but are written in a local log file, to be inspected by the node administrator.

4 Architecture feasibility study

This section describes an actual implementation design for a CDN, which adhere to the architecture and operational procedures described in § 4.2 and 3.6.

Entities which constitute the CDN architecture are now re-defined in a more formal way; some hints about their inner structure are given, and the interfaces by which these entities communicate are evidenced. Messages to be exchanged by entities are enumerated, and some examples about their appearance after XML-RPC encoding is given. After having chosen the programming language and streaming device to be used during development, some further aspect relevant to the implementation phase is detailed, such as concurrent processing, adaptation to network conditions and node load, proximity measures, whois server usage, and authentication.

4.1 Entities interfaces and components

The CDN components are shown in Fig. 12, where can be found a *CDN Service Requester* (in the following, simply the *Requester*), which can be the announcement portal or the authoritative DNS, then the *Request Routing and Distribution Management* (RRDM) entity, and then one or more *CDN node (node)*. Fig. 12 also shows that these entities communicate trough the *CDN Service Interface* in between the *Requester* and RRDM, and trough the *CDN Control Interface* in between RRDM and FH, or in between nodes. As a consequence, the RRDM must implement message processing components for both interfaces, while *nodes* deals only with control interface message processing.

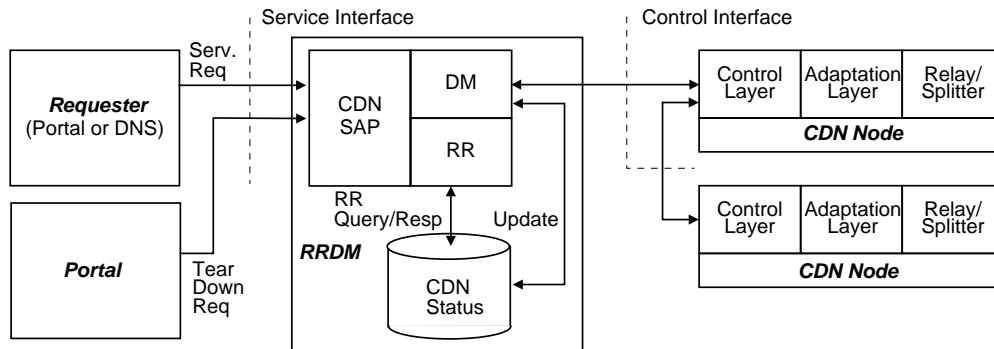


Figure 12: CDN entities, components, and interfaces

4.1.1 RRDM inner components

Within the RRDM entity, the functions of *Request Routing* (RR) and *Distribution Management* (DM) can be thought of as separated, letting us to define some more interfaces, not required to be fully specified, as they are hidden in the RRDM implementation.

The *Service Access Point* (SAP) of RRDM deals with *CDN Service* and *CDN Teardown* requests flowing through the *CDN Service Interface*, while the DM component manages *relay Setup* and *Teardown* requests, as well *registration requests* and *node status queries*, through the *CDN Control Interface*.

Collected status information may be held in an external (redundant) storage component¹⁹, which is queried in turn by the RR component, in order to locate the nodes involved in satisfying a specific *CDN Service Request*.

4.1.2 CDN node inner components

As already discussed in § 3.4.1, *CDN nodes* do interface RRDM by means of a *control layer*, which executes the API calls offered by an *adaptation layer*, which interacts in turn with the *Relay/Splitter* present in the node. Any splitter to be utilized requires an ad hoc adaptation layer implementation: for this reason, in the following the API service primitives will be described as well. Moreover, as shown in fig. 12, the node control layer itself may need to communicate with other peers, performing request recursion, as explained in § 3.6.6.

4.2 Protocol and architecture definition

Communications in between the *Requester* and RRDM, in between the RRDM and the *nodes*, and in between nodes, may be performed by XML encoding of the request/response parameters, and by encapsulation of these parameters in the body of HTTP messages. This behavior is typical operation for *distributed systems* known as *Web Services*, i.e. application to application communication, which have the good qualities of

- being untied from data representation, and independent of the host architecture;
- can use any TCP port, including port 80, not blocked by firewalls;
- ease of the debugging process, as messages are human readable;
- build an object-oriented framework for distributed processing problems.

Some web services are built on a protocol named SOAP (*Simple Object Access Protocol*), which allows for *routing* and in-route *enrichment* of the messages. In our case, messages are exchanged within pair of nodes which communicates directly, so that it is preferable to use the *simpler XML-RPC*²⁰ approach.

¹⁹This solution would allow to continue the *CDN service* also in case of RRDM crash and reboot.

²⁰<http://www.xml-rpc.com>

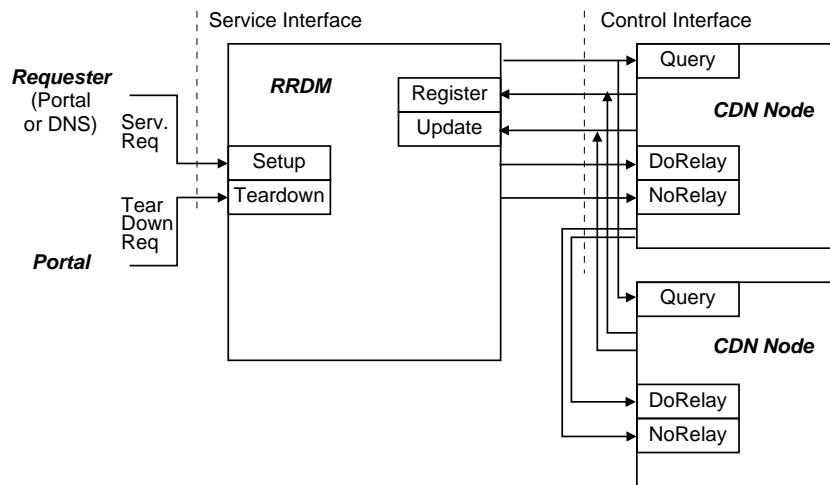


Figure 13: XML-RPC methods as service access points within entities

4.2.1 XML-RPC encoding

XML-RPC spun off from an early draft of the SOAP specification²¹, together with the implementation of a software product (*Frontier*) at UserLand Software, Inc. A large number of implementation toolkits do exist²² for many different programming languages (Perl, Python, C, C++, Java, PHP, .NET, Ruby) so that interfaces which serialize/deserialize data into XML encoding are commonly available. See appendix 7.4 for a short overview of the XML-RPC protocol specifications.

4.2.2 Messages and methods

Table 1 resumes the messages that have been described so far, together with their source and destination, interface and parameters; the last column shows the name of the XML-RPC method in charge of receiving the message, and which performs the requested actions. Fig. 13 maps XML-RPC methods inside of the CDN entities: boxed items represent the *services* that methods implement, and arrows departures indicate the entity which invokes the remote service.

4.2.3 Message flows examples

Here below some examples of actual XML encoding of messages are given.

4.2.3.1 Registration A node registers his address, footprint and transport.

```
<?xml version="1.0"?>
  <methodCall>
    <methodName>Register</methodName>
    <params>
      <param>
        <value>
          <struct>
            <member><name>Address</name>
              <value><string>192.168.0.100</string></value>
            </member>
            <member><name>Port</name>
              <value><string>5500</string></value>
            </member>
            <member><name>IndirectFootprint</name>
              <value><array><data>
                <value><string>192.168.0.0/16</string></value>
                <value><string>151.100.0.0/16</string></value>
              </data></array></value>
            </member>
          </struct>
        </value>
      </param>
    </params>
  </methodCall>
```

²¹<http://www.w3.org/TR/2003/REC-soap12-part0-20030624/>

²²<http://www.xmlrpc.com/directory/1568/implementations>

Message	Source	Destination	Parameters	Return	Interface	XML-RPC method
Registration Re-registration De-registration	node	RRDM	node address footprint (direct or not) transport	OK or unavailable	control	Register
Status Query	RRDM	node	-	CPU load % bandwidth used	control	Query
Status Update	node	RRDM	CPU load % bandwidth used	-	control	Update
CDN Service Request	Requester (portal or DNS)	RRDM	Program URI Requesting Client Addr. transport	Surrogate URI or Redirection NREN page or Failure Code	service	Setup
CDN Teardown Request	Portal	RRDM	Program URI	-	service	Teardown
Relay Setup Request	RRDM	node	Program URI and Requesting Client Addr. and Candidate Downstream Relay List and transport	Redirection NREN page or Surrogate URI and Transit Relay List or Failure Code	control	DoRealy
Relay Setup Request	node	node	Upstream Relay URI and Requesting Client Addr. and Candidate Downstream Relay List and transport	Surrogate URI and Transit Relay List or Failure Code	control	DoRelay
Relay Teardown Request	RRDM or node	node	Program URI and Downstream Relay List	-	control	NoRelay

Table 1: Summary of messages to be exchanged for CDN operation

```

        </data></array></value>
    </member>
    <member><name>DirectFootprint</name>
    <value><array><data>
        <value><string>192.168.100.0/24</string></value>
        <value><string>151.100.122.0/24</string></value>
    </data></array></value>
    </member>
    <member>
        <name>Transport</name>
        <value><string>isma</string></value>
    </member>
</struct>
</value>
</param>
</params>
</methodCall>

```

4.2.3.2 Relay Setup After that the requester has asked to RRDm for CDN setup, RRDm issues a *DoRelay* call to a selected FH (suppose it be at address *192.168.0.100*, advertising a *192.168.0.0/16* indirect footprint), and communicates the *Program* URI of the content origin, the address of the client, and the transport used by the origin. Also, a list of candidate relays is given, along with their advertised footprint.

```

<?xml version="1.0"?>
<methodCall>
  <methodName>DoRelay</methodName>
  <params>
    <param>
      <value><struct>
        <member>
          <name>Program</name>
          <value><string>rtsp://192.168.200.200/netcast.sdp</string></value></member>
        <member>
          <name>Origin</name>
          <value><string>rtsp://192.168.200.200/netcast.sdp</string></value></member>
        <member>
          <name>Transport</name>
          <value><string>isma</string></value></member>
        <member>
          <name>Client</name>
          <value><string>192.168.10.1</string></value></member>
        <member>
          <name>Transit Candidates</name>
          <value><array><data>
            <value><string>192.168.0.103:5503</string></value>
            <value><string>192.168.0.101:5501</string></value></data></array></value></member>
        <member>
          <name>Transit FootPrint</name>
          <value><array><data>
            <value><string>192.168.0.0/18</string></value>
            <value><string>192.168.0.0/17</string></value></data></array></value></member>
        <member>
          <name>LastHop Candidates</name>
          <value><array><data>
            <value><string>192.168.0.107:5507</string></value></data></array></value></member>
        <member>
          <name>LastHop FootPrint</name>
          <value><array><data>
            <value><string>192.168.0.0/19</string></value></data></array></value></member>
      </struct></value>
    </param>
  </params>
</methodCall>

```

The doubling of *Program* and *Origin* information allows to perform *DoRelay* calls either from RRDm to NODE, or in between nodes. *Program* always refers to the URI given by the content provider, and which uniquely identifies the live content, and *Origin* indicates where the node must pick up such content: when the *DoRelay* is issued from NODE to NODE, *Origin* is set to the local mount point of the upstream node. After processing of this message, the FH builds a two-level CDN by *pulling* the media content, and forwarding the request to the *192.168.0.107* LH. Then, it returns the following response to the RRDm:

```

<?xml version="1.0"?>
<methodResponse>
  <params>

```

```

<param>
  <value><struct>
    <member>
      <name>ret_code</name>
      <value><int>200</int></value></member>
    <member>
      <name>ret_val</name>
      <value><string>Relay successfully created</string></value></member>
    <member>
      <name>SurrogateUri</name>
      <value><string>rtsp://192.168.0.107/192.168.0.100_192.168.200.200_netcast.sdp</string></value>
    </member>
    <member>
      <name>RelayList</name>
      <value><array><data>
        <value><string>rtsp://192.168.0.107/192.168.0.100_192.168.200.200_netcast.sdp</string></value>
        <value><string>rtsp://192.168.0.100/192.168.200.200_netcast.sdp</string></value></data>
      </array></value></member>
    </struct></value>
  </param>
</params>
</methodResponse>

```

SurrogateUri is the URI to be returned to the *Requester*, and *RelayList* reports about the activated relays, as well as their associated mount points.

4.2.4 Choice of the streaming server and programming language for pilot experimentation

After discussion with the other components of the TERENA Task Force, it has been decided to use Apple DARWIN as the reference splitter to be used in experimentation. This decision comes for different reasons, among which are freeness of the code, existence of DARWIN implementations for all the existing computing architectures (*nix, Win and Mac), adherence to the ISMA specifications, and support for multicast. The choice of DARWIN should not limit the applicability of experimentation to other transports; for this reason, only its *pull* capability is used.

The choice of *Perl* as the programming language for experimentation comes as well as the result of a series of considerations. Perl code does work fine in all the three (*nix, Win and Mac) worlds, it is a traditional *glue* scripting language for web applications, the resulting code can be very concise and can be broken in different pieces (modules) to be independently developed, and a very good Perl toolkit²³ for XML-RPC processing do exists. The toolkit allows to implement XML-RPC methods as Perl *subs*, and to use as a *listener* (the component that receives XML-RPC calls embedded in HTTP requests) other Perl modules, such as *HTTP::Daemon* and *Net::Server*. The former is a single-thread process which can handle one request at time, and is used for normal development. *Net::Server* is an advanced server which can fork (or even *pre-fork* in APACHE *fashion*) many listener process, and concurrently serve more requests.

4.3 Advanced features

4.3.1 Concurrent processing

As CDN nodes operation deals with a single resource, the Streaming Server itself, nodes do not gain significant advantages from concurrent processing. On the contrary, RRDM may serve the needs of many different contemporary *Programs*, and may control NODES located in very different places, for the needs of very different clients. For this reasons, RRDM must be able to accept and process more contemporary requests: as outlined in 4.2.4, this can be obtained by use of the *Net::Server* Perl module.

4.3.1.1 Shared state and IPC If concurrent processing is enabled, the different instances of RRDM must share a common view of the CDN node registration information, node status, and deployed distribution topology. All this data can be maintained by a separate process, to be queried for information retrieval, and to which submit changes.

²³<http://search.cpan.org/author/RJRAY/IPC-XML-0.53/>

Access to the shared state is performed by means of *Inter Process Communication* (IPC) mechanisms, which in *nix world are based on a well established set of primitives²⁴. Instead, it has been chosen to implement IPC by use of an additional XML-RPC method, executed by the process which holds the shared state data, and invoked by the other concurrent processes.

4.3.1.2 Locking While concurrently serving many different requests, there may happen conflicting circumstances²⁵, where newly arrived requests must not produce further RRDM activity: on the contrary, responses will be sent only after some other operations have completed. For this reason, an arbitration mechanism is needed, in order put a lock on a virtual resource²⁶, and suspend processing until the shared state is updated; after that, the lock is removed, and new processing for the same resource can be executed²⁷. The same locking mechanism can also be used in order to avoid shared state inconsistencies, which may arise if an update is performed during two related write operations: such cases must be carefully analyzed, in order to avoid potential race conditions.

4.3.2 Distributed health information

Maintenance of node health information by the RRDM may introduce scalability problems, as the RRDM should monitor all the CDN nodes. For this reason, a distributed health information status can be envisaged, by letting all the nodes which have downstream relays, to directly collect health information from these.

When RRDM invokes the *DoRelay* method for a *Setup Request*, it will communicate to the FH the *Downstream Candidate Relay List*: from then onward, the node knows which is *his* downstream set, and starts to monitor their health, as well it will do for any other newly communicated downstream relay. Health information will be used as usual, for selection of the *best* surrogate, and will permit to operate a form of load balancing.

In this way, health information stays local in the neighbors of each node, and not accumulates in a unique point of control. The only health information held at the RRDM will be that pertaining to FH nodes.

4.3.2.1 Aging of health information When a node hangs, health information about it becomes stale. For this reason, collected informations must be time-stamped, and requested again after some time has passed. Neighbors nodes can become aware of a node failure in advance of the scheduled polling, as for instance when a CDN request does timeout. When a node failure is detected, it must be reported to the RRDM. This could be accomplished by means of a third-party node de-registration message.

4.3.3 Adaptation to network conditions and node load

Node health measure may not well reflect actual network conditions, which may vary quickly and unexpectedly. As anticipated in 3.6.5 and 3.6.6, when more than a downstream candidate exists, the upstream node should be able to choose the one which answers first. This can be accomplished if nodes do implement an *echo service*, so that they can be probed by means of UDP packets: the upstream node will spawn a different thread of execution for each candidate downstream node, probe them in parallel, and assess the quickest in answering. Also, if some *a priori* preference do exists for some competing downstream candidate, probing of less preferred nodes can be delayed by some extent, in order to give a temporal advantage to more preferred nodes. Alternatively, several UDP probe packets may be sent toward nodes, and RTT averaged, in order to obtain a statistically more significant latency estimate.

²⁴Among *InterProcess Communication* mechanisms are semaphores, shared memory and message queues. In Perl, these tools do depend on the *libc* implementation of IPC itself, which can introduce portability problems.

²⁵For example, a user insists in clicking on the portal, before the CDN Setup initiated by his first request has been completed: subsequent requests should not be processed, until operations started by the first one have completed. Another example is a crowd of clients lying in the same footprint, requesting the service at the same time, for which the same consideration do applies.

²⁶A virtual resource is tied to a physical one, such as a client address, or a footprint, but is represented by an abstraction on the machine where RRDM executes. A viable example of abstraction is a file with the name of the resource, to be locked by a *flock(2)* call, nicely wrapped by the *File::Flock* Perl module.

²⁷This new processing will produce a new lock, and will operate on the new shared state.

The proposed measure has the nice property of being directly related to both the network conditions, and the load average of the probed node. Also, if a node has gone down, and does not answers to the probe, no connection is attempted, thus avoiding the otherwise required await for a TCP timeout.

4.3.4 Proximity measures and *binning*

Despite the manual configuration approach of the proposed solution, real advantages may come if some amount of self-organizing CDN topology is allowed. In fact, it may happens that a candidate LH to be activated for the purposes of serving new clients, is much more near to an already active surrogate (whose footprint is completely disjoint from the one of the new candidate²⁸), than to the *Program Origin*, or to the *Transit Surrogate* which should be used, according to footprint informations.

Network efficiency can be gained if a proximity measure could be estimated, from active surrogate nodes, to candidate nodes. Although in principle this can be done by evaluating some network latency measure in between all the node pairs, like the one stated at the end of 4.3.3, this does not scale at all. A different (scalable) approach has been named *binning*²⁹, and is based on ranking of network latency measurements, made from each CDN node, to a small set (8-12) of fixed *landmarks* hosts scattered³⁰ along the Internet. Near nodes should report a similar ranking of landmark latencies, and as such, they can be classified as belonging to the same *bin*. If every node ranks *landmark* latencies at boot, and communicates the result to RRDM at registration time, then RRDM can estimate proximity of nodes on the basis of ranking similarities, and use proximity information in between nodes, during the best candidates selection phase.

4.3.5 Use of whois servers

As already stated in note (4) and sect. 3.2.1.2, approximate location of a client, and of a candidate surrogate, can be attempted by querying the *whois* database maintained by the RIR which is authoritative for the address block in which the IP client/relay addresses lie. Perl module *IP::Authority* may be used for locating the authoritative RIR for a given address, so that the *whois* query can be addressed to the proper one.

Data returned by the different RIRs do slightly differ, and different Perl modules (*Net::Whois::RIPE*, *Net::Whois::ARIN*), have been developed for such different queries, while other modules (*Net::Whois::IANA*, *Net::Whois::IP*) try to query all the RIRs and to manage differences.

Finally, Perl module *IP::Country* provides access to a database file, gathered from RIRs, for quick translation of an IP address, to the country code of the ISP to which the address block was assigned. The same service is offered by the module *Geo::IP*, which can also use a more accurate database, offered by MaxMind³¹ on a paid subscription basis, and which resolves Country, Region, City, Latitude, and Longitude of any IP address.

4.3.6 Nodes autoconfiguration

Direct *footprint* may be inferred by the node itself at boot, and used in the registration phase. In fact, the IP address and netmask length of the node is a good hint about direct LAN reachability, and they can be simply known by issuing an *ifconfig* command. In a very similar way, a call to *gethostbyaddr* will return the DNS name of the node, whose suffix can be quite safely be used as the DNS direct footprint.

4.3.7 Multicast support

At the time of writing, multicast issues have not been explored. But if multicast works well, there is no need for a CDN!

²⁸This can happen either as a cause of an IP peering in the neighborhoods of the node, or as a consequence of the geographical independence of IP groups and DNS suffixes assignments.

²⁹S. Ratnasamy, M. Handley, R. Karp, S. Shenker, "Topologically-Aware Overlay Construction and Server Selection", IEEE Infocom 2002, New York

³⁰As for example, landmark hosts can the set of root DNS, just passively answering to ping probes.

³¹<http://www.maxmind.com>

4.3.8 Authentication

A simple mechanism for trusting of the exchanged messages, could be based on the use of a Public Key Infrastructure, by selecting a *couple* of *key-pairs* (public and private) to be used at the client and server side of any node, for every method. The *key-pair couple* is read from a configuration file, distributed by other secure mean, among all the nodes taking part to the same CDN. Every request will be signed with the private key of the client side, and authenticated by the server side by use of the client public key. Responses given by the server side will be signed by using the server side private key, and authenticated at the client side, by using the server side public key.

5 Development of the experimental setup

5.1 Requirements

An experimental set up has been made available at <http://labetl.ing.uniroma1.it/opencdn>, where interested users can download the latest development snapshot form CVS, and a *tarball*. Development is based on a REDHAT 9 Linux distribution, and runs on general purpose hardware. A common Perl 5.8 distribution is used, and extra packages from CPAN³² are also included in the repository.

Development is based on Darwin Streaming Server, which has to be downloaded³³ from the Apple website. Only ISMA compliant sources can thus be handled, and users interested in other streaming architectures, are encouraged in writing a new adaptation layer, as detailed in 5.5.

5.2 Implementation details and interfaces

The overall code has been put under an *OpenSource* license, and has been named *OpenCDN*. Distribution includes two *Perl scripts* to be executed for RRDM and NODE entity respectively, plus a reference *adaptation layer* which wraps the Darwin Streaming Server control interface. A *dummy* adaptation layer has also been included in the distribution, for the purpose of speeding up the development process, and it simply emulates the behavior of a streaming server, without actually performing any real operation.

During RRDM implementation, both the Service and Control interfaces defined in 4.2 have been mapped to the same TCP port. Moreover, implementation of concurrent processing and XML-IPC described in 4.3.1.1, has moved the *Setup* method to a different port, used only if *Net::Server* Perl module is used, and has defined a new method (IPC) for concurrent access to the RRDM shared data. Finally, no status information is used yet, as neither are other advanced features, as proximity measures and binning, whois, autoconfiguration, multicast, and authentication.

The following table reports about the proposed ports assignments, and identify as *core* the interface used by the non-forked RRDM methods, and *forked* the other one. The same table also indicates the port used by the NODE entity, evidencing the *probe* UDP socket used for testing network conditions and node load, as described in 4.3.3. We preferred to answer to probe packets directly from the node code, instead of relaying on an external echo service.

entity	interface	port	proto	methods/service
RRDM	<i>core</i>	4400	TCP	Register, Teardown, IPC
	<i>forked</i>	4401	TCP	Setup
NODE	<i>node</i>	4404	TCP	Dorelay, Norelay
	<i>probe</i>	4404	UDP	network_probe

5.3 Configuration of RRDM and NODE entities

Full installation and configuration instruction are given in the README file shipped with ocdn distribution. There are tree text configuration files, one for NODE parameters, one for RRDM, and one in common: if running only one of the entities, then only two configuration files must be filled. These files also allow to

³²<http://www.cpan.org>

³³<http://developer.apple.com/darwin/projects/streaming/>

specify address and port numbers for RRDM and NODE, overriding the defaults shown in table above. A fourth configuration file contains the parameters which are relevant to the streaming server actually used, and at the moment only used for stating the Apple Darwin *admin user* and *password*.

A special configuration variable for nodes has been named `$node_char`, which can hold one of the character values *'labor'*, *'fair'*, and *'lazy'* alternatively, and which do sort some effect only when the node will behave as a *Transit*. In the *labor* case, the node will try to contact the *more specific* LH node contained in the candidate list received from upstream, while in the *lazy* case it will try to contact the *less specific Transit* node of the list. While a *labor* behavior means that subsequent CDN requests will continue to be dealt by the same node, *lazy* behavior will limit node involvement for future calls, to only those footprints not already dealt by some of its downstreamers. Finally, in case of *fair* behavior, the node will switch to *lazy* or *labor*, depending if node load conditions are exceeding some threshold or not.

The distribution also contains some additional Perl CGI files, providing a *web interface* to RRDM and NODE status, and allowing to locally test the installation setup as detailed in 6.1. In order to make these CGI addressable, few lines must be added to the web server configuration files, and the README gives an example for the *Apache* web server.

5.4 CDN service invocation

The requester entity can send *CDN Setup* and *Teardown* requests, by forging a proper XML-RPC call, and address it toward the RRDM, at the port where it is listening, which either is the one listed in the table shown in 5.2, or the one stated in the config files. Also, please remember that if a forked RRDM is in use, *Setup* requests must be addressed to the same port number, plus one (but check distribution README for last minute changes). The format of *Setup* calls is given in the example given below, and the only supported *Transport* at the moment is *isma*, dealt by *Apple Darwin*.

```
<?xml version="1.0"?>
<methodCall>
  <methodName>Setup</methodName>
  <params><param>
    <value>
      <struct>
        <member><name>Client</name>
          <value><string>192.168.0.100</string></value>
        </member>
        <member><name>Program</name>
          <value><string>rtsp://192.168.200.200/netcast.sdp</string></value>
        </member>
        <member><name>Transport</name>
          <value><string>isma</string></value>
        </member>
      </struct>
    </value>
  </param></params>
</methodCall>
```

Similarly, the format of a *Teardown* call is given in the following example, where the parameter *Requester* has been inserted just for logging purposes, and in the set up detailed at 6.1, it will report the client who has requested the *Teardown*.

```
<?xml version="1.0"?>
<methodCall>
  <methodName>Teardown</methodName>
  <params><param>
    <value>
      <struct>
        <member><name>Program</name>
          <value><string>rtsp://192.168.200.200/netcast.sdp</string></value>
        </member>
        <member><name>Requester</name>
          <value><string>Portal at 192.168.123.321</string></value>
        </member>
      </struct>
    </value>
  </param></params>
</methodCall>
```

5.5 Porting of adaptation layer to other platforms

Porting requires new Perl module to be written, containing the implementation of the the subroutines already developed for the *Darwin* and *Dummy* Streaming Servers. This new module will come in use, if its name appears in the proper place within the NODE configuration file.

Appendix 7.5 reports about the synopsis of the API calls to be implemented. Basically, these provide a common interface for setting up a new pull relay, dismantle it, and gathering some information about the server functioning.

6 Experimentation results

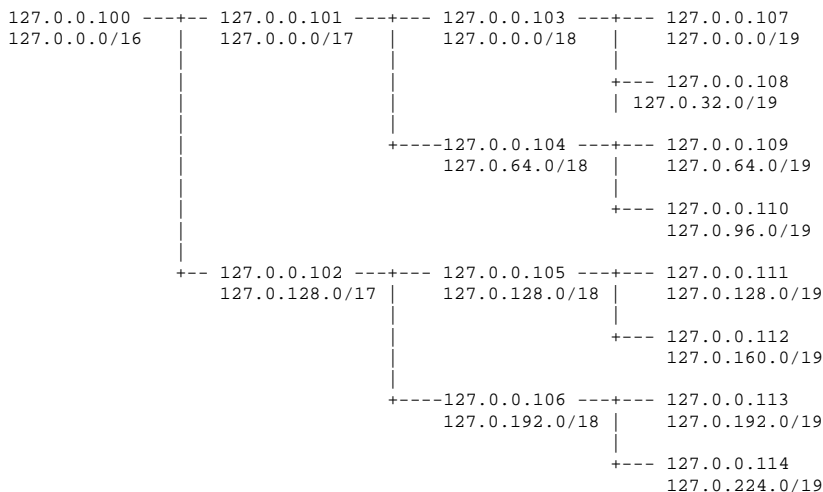
OpenCDN has been experimented at the Info-Com department, University of Roma "La Sapienza", by using as source a 230 kbps, QCIF sized MPEG4 feed, obtained by coding a terrestrial TV channel with the tools from the MPEG4IP³⁴ project. A second source was a 450 kbps, CIF sized MPEG4 file, turned to a live stream. More sources have been provided by TF-NETCAST members.

6.1 Demo page

At <http://labtel.ing.uniroma1.it/opencdn> an HTML page has been set up, from where it is possible to choose in between the available sources, and to ask for CDN *Setup/Teardown* service. The request is fed to a CGI Perl script, which forges the XML-RPC call for RRDM, waits for a response from it, and generates an HTML output to be shown to the requesting client. This latter response page contains the surrogate URI communicated by the RRDM, and which should be used by the client for reception of the requested source.

6.2 Test CDN

In order to test and evaluate CDN operations in presence of many registered nodes, with various announced footprints, and different node behavior, 15 different nodes have been launched on the same machine, each announcing a different footprint, and different client addresses can be simulated. Here follows a sketch of the test network:



For each of the 15 nodes, the IP address and advertised footprint are shown. Please note that lines are not actual links, but only represent how footprints encapsulate, as well as how the distribution chain of surrogates can be when all of them behave as *lazy* nodes. Rightmost nodes do represent LH nodes.

³⁴<http://mpeg4ip.net/>

7 Appendices

7.1 Technology survey about streaming devices

Here below follows a survey compiled by after web browsing in 2003 2Q, and as such, it must not be regarded as either accurate or exhaustive. Although, it can well represent the very scattered scenario offered to users and administrators.

NetCache (Network Appliance) is compatible with Real, MS and QT formats and protocols, and offers three rackable product lines, with huge disk and bandwidth capacity. Scalable delivery is performed by replication of live streams to remote NetCaches. A *Global Request Manager*³⁵ solves the problem of request routing, by deployment of a server/agent DNS architecture. Moreover, NetCache may act as a transparent proxy for live streaming content, by interception of the client requests. In any case, it requests the stream from the original source, only after it has been requested by a client.

ISMA All the constituents of the corporation offer solutions, and ones from APPLE, ENVIVIO, FRAUNHOFER IIS, IVAST, PHILIPS, SUN, KASENNA, can be find on the ISMA site.

IBM Video Charger Is offered on Windows NT and AIX, supports RTSP control and multicast distribution, Quicktime and MPEG-1, 2 and 4 formats, although reception of the latter seems to require a java player downloaded from the server. Compatibility with ISMA specifications is claimed.

Envivio offers both software and hw-rackable solutions, compliant to ISMA specifications, and sells also an encoder, a producer, and a free player.

MediaBase XMP (Kasenna) complies to MPEG and ISMA specifications, and runs on SUN, SGI and INTEL/LINUX, over IP and ATM. It can deliver video via unicast and scheduled multicast, can record while streaming, and provides full VCR controls (FF/RW, pause, stop, play) for VOD stream, by using of its windows client. It supports push and pull on-demand content distribution, and performs *Prefix Caching* of content for browsing purposes, and for reducing streaming latency. An integration framework is provided as an API, based on Web services programming paradigm (SOAP, XML, Java, etc). Distribution of Live content is performed by manual setup of splitting functions, and automation of the process can be integrated in the platform.

iVAST has an ISMA-compliant MPEG-4 hardware encoder, and an RTSP server running on Windows 2000 server. It provides also mpeg-4 authoring tools.

MediaBox by iTuner Networks is an embebbed Linux server, supporting encoding and streaming, and supporting MPEG4, Real, MP3, Vorbis and QTSS.

Below are reported informations about systems which appear to be different from those previously reviewed, as based on different assumptions, oriented to a different market, or whose features are poorly publicly documented.

VideoLan is an Open Source project for streaming of MPEG-1, 2 and 4, DivX files, DVD, digital satellite channels, digital terrestrial television channels and live videos on IPv4 or IPv6 network, in unicast or multicast under many OSes. It also features a cross-plaform multimedia player. Does not offer RTSP control, and can announce transmissions via SAP/SDP.

Cisco IP/TV is a multicast streaming system, delivering MPEG-1 and 2, ISMA MPEG-4, and ASF media, and executes on routers or Win NT. A Broadcast Server supports one capture card for encoding from devices such as video cameras, VCRs, DVDs, satellite and cable feeds, or prerecorded Windows Media, AVI, MP3, and MPEG files. A Content Manager directs the operations of the Broadcast Servers, by communicating scheduling information, available video formats, and bandwidth considerations, as well as publishing programming information for the IP/TV client viewer. Media is distributed by multicast, and announcements interoperate with SDP metadata used in MBONE. Non-multicast

³⁵http://www.netapp.com/tech_library/3152.html

enabled network sections are tunneled by *smallcast*, allowing a server to make up to 20 unicast connections toward users, or other servers, which may act in turn as unicast to multicast relays.

Streaming21 offers services comparable to Cisco IP/TV. Is it a reseller ?

Inktomi has a *Traffic Edge* product that caches and delivers rich-media and traditional-media content, including live streaming audio and video, and which can be operated together with a *Content Delivery Suite* (CDS), for automating content distribution, synchronization and replication from source to target servers or caches. The same product is integrated in solutions sold by HP, FUJITSU, DELL, COMPAQ and 3COM. Actually the company has been acquired by Yahoo, and is mainly devoted to Web search and paid insertions.

VBrick VBricks are a family of "Video Bricks", sold as turnkey solutions, which allow delivery of Unicast and Multicast streams, broadcasted over IP and ATM. *VBrick* 7000 devices do encode/decode MPEG-1 and MPEG-2 formats; *VBXcast* performs ISMA-compliant MPEG-4 encoding, and conversion from MPEG-2 to MPEG-4; *VBXcoder* converts from MPEG-1 and MPEG-2 to MS .wmv format. An SDK is provided, allowing to develop applications which make use of the VBricks. Very few is said about compliance to open standards, and even the player seems to be proprietary.

nCUBE is appointed in VoD services, and offers a system architecture based on *n4*, a Streaming Media Appliance Hub, configured to handle ATM/OC-3C, ATM/OC-12, 64-QAM, 256-QAM, DVB-ASI, ethernet, gigabit ethernet, and supporting MPEG-2 TS. Scalability is accomplished by meshing of *n4* elements in *hypercubes*, which can be grouped in hierarchical fashion. Their market is mainly oriented to Cable Television operators.

Akamai offer a DNS-based redirection service, aiming to distribute the load of highly requested contents among several caches, hopefully located near to the requester. But a recent analysis³⁶ has shown that DNS redirection do not picks up the *best* cache.

Cable & Wireless (formerly *Exodus - Digital Island*) provides a distributed network to customers for on-demand streaming of stored media, as well as end-to-end streaming solutions for live Web events, and supports MS WM, Real, and QT formats. The network is made by streaming servers placed at the edge of the Internet, which provide requesters with local access to content, by means of their patented *Best Distributor Selection* technology, which topologically allows end-viewers' requests to be directed to the most optimal content distributor. Application-level multicasting is performed, with enhancements such as packet loss network correction and traffic redirection around failures.

Globix has deployed an *EarthCache* CDN, allowing for peak-load demand of unexpected crowds; it is proposed for streaming of events, lectures internal communications and more, worldwide. Contents are hosted at the edge, leveraging more than 1,200 peering agreements worldwide.

Speedera offers content delivery services, using a global CDN. Users may set up live streaming events by using the *SpeedEye* browser-based interface. Strategically located traffic managers and network probes, allow for intelligent content routing decisions based on real-time network, server and application health criteria. Speedera also provides a complete set of logs, and supports MS WM, Real, QT, and MPEG formats.

Cidera operates a satellite broadcast network, allowing to simultaneously deliver live streaming video and audio content to servers located at numerous access points, avoiding congestion on the Internet, and delivering content directly to the edge. It is compatible with MS WM, Real, and QT formats.

StreamOS by *Nine Systems Corporation* is a unified browser-based environment, where content developers can manage the diverse activities needed to deliver streaming media content to web users. A *Broadcast Relay Manager* enables producers to manage ingress locations, featuring a *Multi-CDN Routing Engine*, and allows for scheduling of back up encoders and for creating custom "Off Air" clips. It supports .SMI, .RAM, .WMV and QT.

³⁶<http://www.terena.nl/conferences/wcw/Proceedings/S4/S4-1.pdf>

Burstware by Burst.com, proposes an architecture in which a *Conductor* distributes requests over multiple servers, balancing the load and providing complete fail-over protection. The server and the conductor run on Win NT, Linux and Solaris. Supported media types are: MPEG, ASF, AVI, DIVX, MOV, H.263, Sorenson and MP3; reception is possible only with Windows, by using WMP or QT, after installation of a Burstware plugin.

IXJet by 3CX offers a Live server, running on Win NT, which streams MPEG-1 content to DirectShow clients, and a VoD server, supporting MPEG-1, -2, QT and ASF formats.

Amino Communications offers products for traditional broadcast or new on-line services. Combining in-house IP with open applications such as Linux, they created a range of Set Top Boxes and Gateway products and licensable designs. Connected to the Internet and to a TV set, the STB can act as RTSP (VoD) client, multicast receiver, MPEG 1&2 decoder; enhanced models do include a DVB-T receiver, an HD, and a streaming server.

Internet streaming is a very dynamic contest, and after one or two years enterprises changes market, and products disappear. Here below you can find a report about the actual status of previously (in other reports) referred items.

FVC.COM I-studio First Virtual Communications³⁷ is now oriented in videoconferencing, and seems not supporting anymore streaming solutions.

Entera has been acquired by cacheflow, which has been acquired by BlueCoat.

7.2 DNS redirection mechanisms and motivations for not to use it

DNS redirection works by returning a different IP address for a given URI, on the basis of the IP address from which the request comes from. Translation is done (see fig. 2 at pag. 9) by a (modified) authoritative DNS for that URI, which is aware about the IP address of the requesting DNS, hopefully co-located with the requesting client.

DNS redirection is actually used by some CDN implementations (e.g. Akamai³⁸). Unsuitability reasons for this approach are given in the IETF drafts, namely:

- DNS only allows resolution at the domain level, but it should be done at the object level.
- DNS Request-Routing is based only on knowledge of the client DNS server, as client addresses are not relayed within DNS requests. This limits the ability of the Request-Routing system to determine a client's proximity to the surrogate.
- DNS servers can request and allow recursive resolution of DNS names. In that case, a Request-Routing DNS server will not be exposed to the IP address of the client's site DNS server, but to the address of the DNS server that is recursively requesting the information on behalf of the client's site DNS server.
- Users that share a single client site DNS server will be redirected to the same set of IP addresses during the TTL interval. This might lead to overloading of the surrogate during a flash crowd.

7.3 Address sets, footprints, and metric

A footprint is an address set, which can be expressed either in terms of DNS domain addresses suffixes, or in the form of IP addresses prefixes. In both cases, a metric can be defined, so that different footprints can be ordered, from the more specific to less specific, with respect to a given destination address, which is contained within the footprints. Some example will clarify the method.

³⁷<http://www.fvc.com>

³⁸<http://www.akamai.com>

7.3.1 DNS metric

In this case, relay candidates express their ability to be last-hop or transit relays for DNS addresses, as shown in the following table.

Relay	Supported Domains	Transit	Last Hop
A	.it	x	
B	.uniroma1.it	x	x
C	.ing.uniroma1.it	x	x
D	.it cineca.it	x	x
E	.nl	x	
F	.surfnet.nl	x	x
G	.funet.fi	x	x

When a client with a given DNS address arrives, a match is performed in between the ending of this address, and the endings supported by the relay candidates. Two different subsets of relays can be found, namely, those accepting to behave as transit relay for that client, and those accepting to be the last hop surrogate. For each of the two sets, an ordering can be found, going from a more specific to a less specific match, by counting the number of matched DNS name sub-components. Generally, more matched components indicates a closer proximity of the relay to the client.

Client <i>genni.ing.uniroma1.it</i>	Relay subset		
	Preference	Transit	Last Hop
	<i>more specific</i>	C	C
	↓	B	B
	<i>less specific</i>	A, D	

7.3.2 IP metric

In this case, relay candidates express their ability to be last-hop or transit relays for IP address prefixes, as shown in the following table.

Relay	Supported Prefixes	Transit	Last Hop
A	130.186.0.0/16	x	
B	130.186.1.0/24		x
C	151.100.0.0/16	x	
D	151.100.112.0/20	x	x
E	151.100.122.0/24	x	x
	151.100.120.0/21	x	
F	192.87.0.0/16	x	
G	192.87.5.0/24	x	x
H	193.166.0.0/16	x	x

When a client with a given IP address arrives, a comparison is performed in between the 32-bits binary representation of the client IP address, ANDed with a *netmask* given by a left-aligned sequence of ones, whose length is given by the */length* field of each footprint, and the 32-bits binary representation of the IP addresses footprint field, ANDed with the same *netmask*. When a match is found, the client belongs to the subnet specified by that footprint. Again, two different subsets of relays can be found, Transit and LastHop, and for both of them, the ordering is given by regarding as a more specific match, the one which is related to a *longer* netmask.

Client <i>151.100.122.85</i>	Relay subset		
	Preference	Transit	Last Hop
	<i>more specific</i>	E	E
	↓	D	D
	<i>less specific</i>	C	

7.3.3 Comparison of the two techniques

In both cases, an ordered set of relays is found, for the purposes of distribute the media stream toward the client, and find suitable transit and last hop nodes.

The IP prefix approach seems to be more fine-grained and well tailored, as it emulates the way normal Internet routing is performed. But it requires specific knowledge about addresses assignments, and proper configuration for less specific nodes (i.e. those located near the NREN backbone) must be done by a person which is well aware about all the addresses ranges allocated.

The DNS approach is simpler to configure, but it is more exposed to problems related to the scattering of nodes of the same organization in different places.

Neither of the proposed metrics seems to be particularly well suited for the problem under analysis. However, they represent two equally valid approaches for initiating experimentation.

7.4 Short overview of XML-RPC

The following introduction is an excerpt taken from <http://www.xmlrpc.com/spec>.

7.4.1 Definition

XML-RPC is a Remote Procedure Calling protocol that works over the Internet. An XML-RPC message is an HTTP-POST request. The body of the request is in XML. A procedure executes on the server and the value it returns is also formatted in XML. Procedure parameters can be scalars, numbers, strings, dates, etc.; and can also be complex record and list structures.

7.4.2 Request example

Here's an example of an XML-RPC request:

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml Content-length: 181

<?xml version="1.0"?>
  <methodCall>
    <methodName>examples.getStateName</methodName>
    <params>
      <param>
        <value><i4>41</i4></value>
      </param>
    </params>
  </methodCall>
```

7.4.3 Response example

Here's an example of a response to an XML-RPC request:

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:08 GMT
Server: UserLand Frontier/5.1.2-WinNT

<?xml version="1.0"?>
  <methodResponse>
    <params>
      <param>
        <value><string>South Dakota</string></value>
      </param>
    </params>
  </methodResponse>
```

7.4.4 Payload format

The payload is in XML, a single `<methodCall>` structure. The `<methodCall>` must contain a `<methodName>` sub-item, and may contain a `<params>` sub-item. In this case, the `<params>` sub-item can contain any number of `<param>` sub-items, each of which containing in turn a `<value>` sub-item. Finally, the `<value>` sub-item contains either a scalar value (`<int>`, `<boolean>`, `<string>`, `<double>`, `<dateTime.iso8601>` or `<base64>`), or a `<struct>` sub-item (which in turn contains one or more `<members>`, each containing a `<name>` and a `<value>`), or an `<array>` (which contains a single `<data>` element, which can contain any number of `<value>`s).

7.4.5 Response format

Unless there's a lower-level error, always return 200 OK. The *Content-Type* is `text/xml`. *Content-Length* must be present and correct.

The body of the response is a single XML structure, a `<methodResponse>`, which can contain a single `<params>` which contains a single `<param>` which contains a single `<value>`.

The `<methodResponse>` could also contain a `<fault>` which contains a `<value>` which is a `<struct>` containing two elements, one named `<faultCode>`, an `<int>`, and one named `<faultString>`, a `<string>`.

A `<methodResponse>` can not contain both a `<fault>` and a `<params>`.

7.4.6 Fault example

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 426
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:02 GMT
Server: UserLand Frontier/5.1.2-WinNT

<?xml version="1.0"?>
  <methodResponse>
    <fault>
      <value>
        <struct>
          <member>
            <name>faultCode</name>
            <value><int>4</int></value>
          </member>
          <member>
            <name>faultString</name>
            <value><string>Too many parameters.</string></value>
          </member>
        </struct>
      </value>
    </fault>
  </methodResponse>
```

7.5 Adaptation layer synopsis

Here are reported the subroutines implemented in the *Adaptation Layer*, the arguments used for the calls, and values returned.

sub name do_relay

description Pulls media from an *Origin* server, located at a given mount point, and makes it available for clients, at a local mount point. Tags this relay with a given name.

arguments	<code>\$address</code>	<i>IP address of the origin</i>
	<code>\$mountpoint</code>	<i>mount point at the origin</i>
	<code>\$destmount</code>	<i>local mount point</i>
	<code>\$name</code>	<i>name tag for the relay</i>

returns	\$ret_code	200 or 220 on success 550 on failure
	\$ret_val	<i>reason phrase</i>

sub name no_relay

description Dismantles the relay indicated by the name tag.

arguments \$relay_name | *name tag* of the relay

returns	\$ret_code	200 on success 560 on failure
	\$ret_val	<i>reason phrase</i>

sub name query_band

description Gathers total throughput from the streaming server

arguments none

returns \$current_bandwidth | *total throughput* in bit/s

sub name server_status

description Checks if the streaming server is running and if everything is ok with it. For instance, in the case of Dawin SS, a check of user and password values in Config file is performed

arguments none

returns	\$ret_code	ok on success <i>reason phrase</i> on failure
---------	------------	--

sub name relays_stats

description Returns informations about active relays

arguments none

returns	\$ret_code	<i>reference to a data hash</i> on success (see code) <i>reason phrase</i> on failure
---------	------------	--

sub name clients_stats

description Returns informations about connected clients

arguments none

returns	\$ret_code	<i>reference to a data hash</i> on success (see code) <i>reason phrase</i> on failure
---------	------------	--