Programme ANR VERSO

Projet VIPEER

Ingénierie du trafic vidéo en intradomaine basée sur les paradigmes du Pair à Pair

Décision n° 2009 VERSO 014 01 à 06

du 22 décembre 2009

T0 administratif = 15 Novembre 2009

T0 technique = 1er Janvier 2010

Livrable 1.3
Update of the functional architecture

Auteurs :

F. Guillemin et Stéphanie Moteau (Orange Labs), Y. Hadjadj-Aoul (INRIA), Jérémie Garnier (NDS), Annie Gravey (Telecom Bretagne)

Compilé par :

F. Guillemin (Orange Labs)

Mars 2012

Telecom Bretagne ; Eurecom ; INRIA ; France Telecom ; NDS ; ENVIVIO

Résumé : *(15 lignes)*

The VIPEER project aims at developing a distributed CDN (dCDN) operated by an ISP in order to disseminate content at a national or regional scale while controlling the quality of the delivered content. VIPEER pays special attention to the distribution of video by means of adaptive streaming. Content is thus divided into chunks downloaded by end users with an adaptive streaming viewer.
This deliverable presents an updated version of the functional architecture presented in Deliverable D1.2. The VIPEER architecture relies on three basic building blocks: 1) a dTracker in charge of the interface with the surrogate CDN, of the control of the placement of content in the dCDN, of the control of the selection of the replica to be sent when a user requests a given content 2) a dCollector in charge of controlling the load of the dCDN and to balance traffic in coordination with the dTracker, 3) dCDN nodes in charge of storing chunks. This deliverable describes the basic building blocks and specifies how content is injected into and managed in the dCDN. Various possibilities are envisaged for the download of chunks, one is based upon successive redirections while another is based on a peer-to-peer approach. This document also specifies the metrics that have to be exported by the dCDN nodes and used by the dCollector ; it also addresses the management of data bases in VIPEER.

Key words : HTTP streaming, Digital Rights Management, Contend Distribution System, Quality of Service, Quality of Experience, Collector, Tracker

Table of content

Table of Figures

1 Introduction

The goal of the VIPEER project is to design a distributed content distribution network (dCDN) operated by a network operator to assist a global Content Distribution Network (CDN) in the delivery of content in the last miles. The objectives of a dCDN are at least twofold:

- Since the dCDN is operated by the network operator, which can control the QoS of delivered content and balance the load in his own network, the QoS perceived by the end user is significantly increased in comparison with the case when the content is delivered through the pure best effort channel, which can traverse congested links, notably peering links between transit networks.
- Since the network stores content, the load of peering links either with transit network or the bandwidth necessary to connect the server of the CDN is reduced. Since the cost of this latter bandwidth is general high, CDN tends to reduce it at the risk of degrading the global QoS of delivered content if the demand by end users is high. In addition, the upgrade of peering links is also costly for the network operator.

The tough questions that the VIPEER aims at addressing are the following:

- Is it possible to balance traffic inside the network by using a distributed CDN? The alternative is clearly to have a large centralized server. In that case, there may be some risk of link congestion in presence of unbalanced requests.
- What is the gain achievable by coordinating small or medium size CDN servers? If servers are not coordinated, content is replaced in these servers by taking local decisions only and content can be replicated at the detriment of other content, thus leading to retrieve more content from the content provider or the surrogate CDN.
- To distribute Over The Top streamed video services (e.g., Web TV), which are requested by a large number of users, the most efficient way is clearly to establish multicast trees in the network. Is it then possible to efficiently construct multicast trees by using a distributed CDN?

As a prerequisite in the VIPEER project, we assume that content is segmented into chunks. In the particular case of video content, several representations of the same content are available. Specifically, chunks corresponding at various resolutions are available and can be downloaded by end users. In other words, VIPEER assumes that video streaming is supported by HTTP Adaptive Streaming (HAS).

As described in Deliverable D1.2, the dCDN is composed of a number of functional elements that are respectively in charge of:

- The interface with the CDN, which is also responsible for duplicating content and for selecting the server when a user requests content. This element is referred to as dTracker.
- Network monitoring. This element, referred to as dCollector, aggregates information sent by the dCDN nodes .
- dCDN nodes which store the content and which enforce some rules in order to deliver content with a given quality level.

The global architecture of the VIPEER project is illustrated in Figure 1.

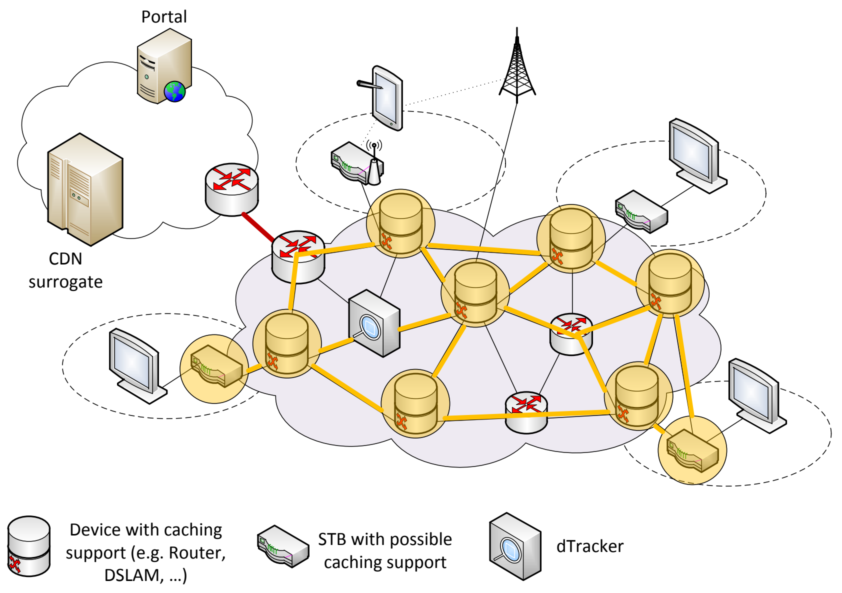

Portal
CDN surrogate
Device with caching support (e.g. Router, DSLAM, …)
STB with possible caching support
dTracker

Figure 1 : Global Architecture of a VIPEER system

The roles of the various actors when a dCDN is implemented (namely the content provider, the Content Delivery Network, the ISP, the transit networks, and the Internet Acces Provider) are described in Deliverable D1.1.

In the VIPEER architecture, end users request content from a content provider that forwards the request to the CDN, which eventually forwards the request to the dCDN. The dCDN is then responsible for content delivery to the end user but is not seen by the end user (only through http redirection).

The dCDN does not modify content (no transcoding) and can possibly modify the information passed by the CDN (for instance by modifying the manifest in the case of adaptive streaming).

Digital Rights Management policies are not modified by the dCDN as the content provider is still responsible for cyphering the content and applying access control.

The various elements involved in the delivery chain are recalled in the next section.

2 Functional blocks

Several functional blocks have been identified in Deliverable D1.2. These blocks are illustrated in Figure 2 that represents:

- A user requesting a content
- A CDN uploading content
- The VIPEER system optimizing its elements

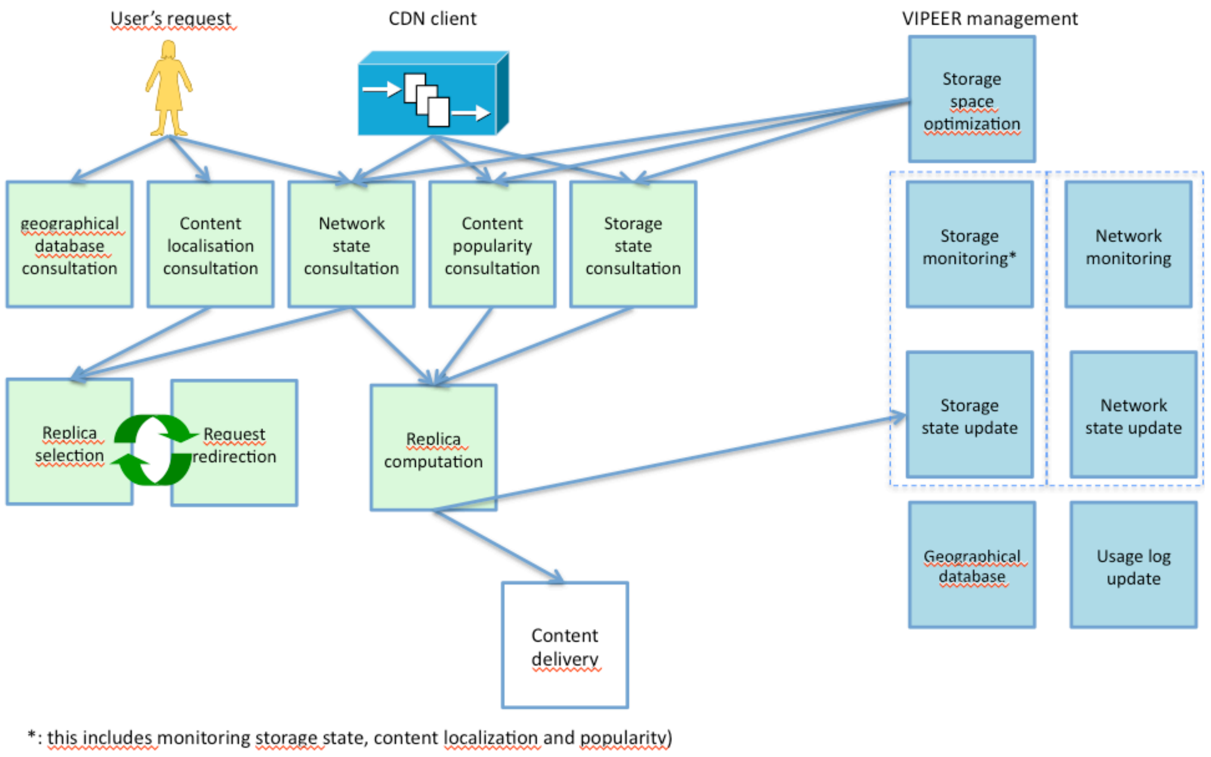

User's request
CDN client
VIPEER management
Storage space optimization
geographical database consultation
Content localisation consultation
Network state consultation
Content popularity consultation
Storage state consultation
Storage monitoring*
Network monitoring
Replica selection
Request redirection
Replica computation
Storage state update
Network state update
Geographical database
Usage log update
Content delivery
*: this includes monitoring storage state, content localization and popularity)

Figure 2 : Functional Blocks in Vipeer

VIPEER management relies on maintaining several databases that record

- Geographical data, that is used to select servers that are « close » to a user
- Storage data, which records in which nodes content is available, and available storage space
- Network data, which records available uplink and downlink bandwidth for each dCDN node
- Usage data, which can be used for charging purposes (this function is not further addressed in the VIPEER project).

Since each database is consulted whenever either content is uploaded or when users' requests are served, they should be provided by scalable methods.

The blocks represented in Figure 2 can be realized differently. In particular, the VIPEER project assesses two methods to implement a dCDN:

- A centralized method for replica placement, which is associated with a centralized method for serving users' requests (sections 4.1 and 4.3.1).
- A method inspired by P2P procedures in order to serve requests (section 4.3.2), and can be coupled either with the centralized placement method, or with a P2P placement method.

3 Functional elements

In this section, we recall the functional elements of the dCDN, which have been described in details in Deliverable 1.1. The complete delivery chain is composed by

- The content provider's server (we do not mention the server supporting the search engine, for instance Google, which redirects the user to the adequate content provider).

- The CDN's servers. As usual, the Content Provider pushes content into the CDN.
- The dCDN, which is operated by the ISP.

The delivery chain composed of the Content Provider and the CDN is the classical delivery chain, which is already implemented in real networks. The real innovation of the VIPEER project is to introduce a new actor (namely the dCDN) into the delivery chain in order to improve the quality of experience of the user by controlling the delivery of content in the last miles.

The dCDN is controlled and operated by the ISP who can use it to deliver content delegated by several independent CDNs.

3.1 dCDN nodes

The dCDN is composed of nodes that are interconnected by (virtual) links. The dCDN nodes form a virtual network which overlays the ISP's network, and which is fed by the client CDNs.

Nodes (represented in Figure 3) are composed of storage capacities and processing capacities implementing some functions:

- Control plane functions:
    - Bandwidth monitoring;
    - Online measurement reported to the dCollector (packet loss, delay estimation, load of the links with its neighbors, etc.);
    - Storage space monitoring (available storage space in each node, list of chunks available in each node)
    - Geographical information (geographical database locating users and nodes.)
- Data plane functions:
    - Packet/chunk multiplexing (possibly by using some traffic management functions such priority scheduling, bandwidth reservation, etc.)
    - Admission control (a node could deny the transmission of a chunk if it degrades too much the transmission rates of ongoing chunks);
    - Storage management via a given content replacement policy and chunk prefetching.
- The dCDN nodes can be hosted by end users (in their terminals, gateways, etc.) or by the network elements (routers, switches, etc.).

The virtual links of a dCDN are supported by the transmission links of the network. Those virtual links may be throttled in order to limit the bandwidth consumed by a dCDN, for instance when a dCDN is a specific server connected to the network (implementation of a rate limiter). This is especially the case when a dCDN node is implemented in the user terminal or gateway. A rate limit is implemented in order to limit the bandwidth consumed by the dCDN and prevent congestion of the user uplink since the uplink is no more used by the end user himself but by the dCDN. When a node of the dCDN is supported by a network element, then the bandwidth allocated to the dCDN when multiplexing traffic issued from the dCDN node and the rest of traffic traversing the network element may be limited (rate limit) or some priority queuing may be used in order to limit the impact of dCDN traffic on the rest of network traffic. Note that traffic inside the dCDN may be greatly unpredictable since it depends on request by end user and the balance functions enforced by the dCDN.

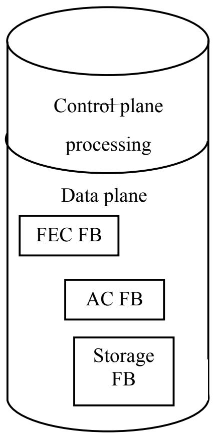


Figure 3 : Internal architecture of a dCDN node

3.2 dTracker

A dTracker is in charge of the interface with the CDN. New content is injected in the dCDN via a dTracker. When a CDN uploads content in the dCDN, it may provide popularity and identification of the previous requests regarding this content. A procedure is requested to select on which dCDN nodes new content is uploaded. Several procedures can be envisaged, depending on the policy implemented by the network operator as e.g.:

- if the ISP deploys a large storage space close to each regional POP, each new content can be copied in this large regional storage;
- if a content is likely to be requested in a single region, copies can be made preferably in this particular region;
- if the expected popularity of the new content is limited, or unknown, the dCDN can upload the content in a default storage space, while recording the actual popularity of this content in order to re-optimize the content upload in future.

Generally speaking, a dCDN could be composed of several dTracker. In the early development of the VIPEER demonstrator, only one dTracker will be in place, see Figure 4.

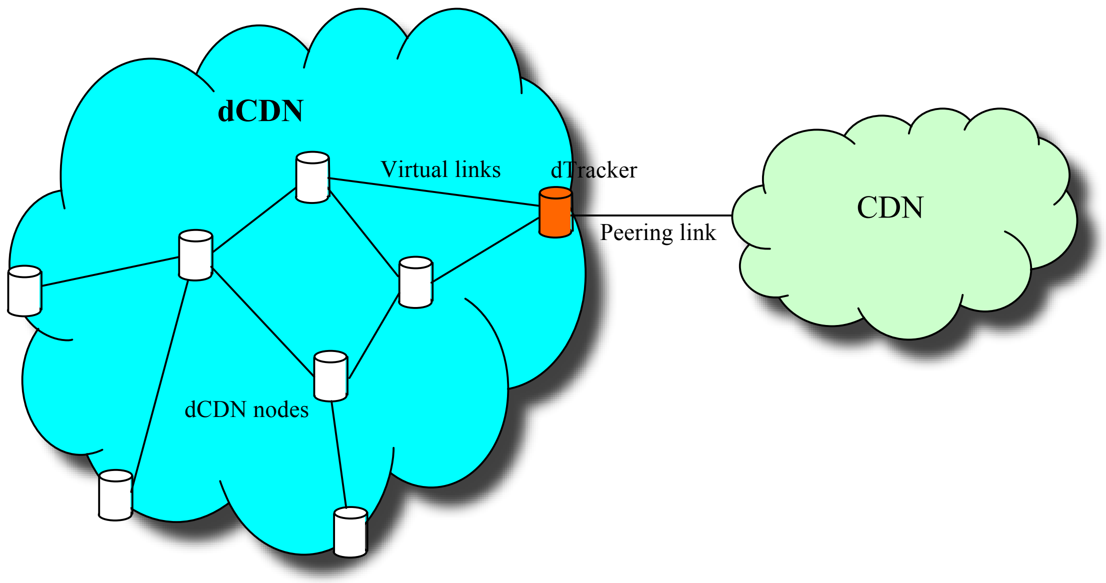


Figure 4: dCDN with a single dTracker

The dTracker also controls the chunk placement on the dCDN nodes. It maintains a list of those chunks that have been injected into the dCDN. In addition, the dTracker maintains the list of chunks stored by the different nodes of the dCDN. For maintaining content in the dCDN, the following features could be implemented:

- To maintain an up-to-date list of chunks in the dCDN, the various dCDN nodes could send the list of chunks they are storing on a periodic basis (for instance every minute, nodes could sent to the dTracker the chunk list). An alternative is for the dTracker to maintain the database, by recording its successive decisions regarding chunk duplication.
- A dTracker could send to a node a list of chunks to maintain, independently of the local replacement policy implemented by the node. For instance to construct some multicast tree in the network.

In the functional architecture described in Deliverable D1.2, when a user requests content:

- The user sends a request to the content provider.
- The content provider knows if the desired content is stored in the CDN. Otherwise the content provider pushes the content to the CDN. If the content is already in the CDN, the content provider redirects the request to the CDN.
- In the case, when the content is already known by the CDN, the CDN knows whether the desired content is already stored in the dCDN. If this is the case, the CDN redirects the request to the dTracker, which then handles the request by redirecting the request to a dCDN node where the content is placed. The selection of the dCDN node can implement procedures to identify which dCDN nodes are the most suited to handle the request (availability of the content, minimal distance to the end user, biggest available

bandwidth, ect.). If not, the CDN pushes the content to the dCDN via the dTracker, which then redirect the content to the end user via the node, which is the most appropriate.

The above process is illustrated in Figure 5.

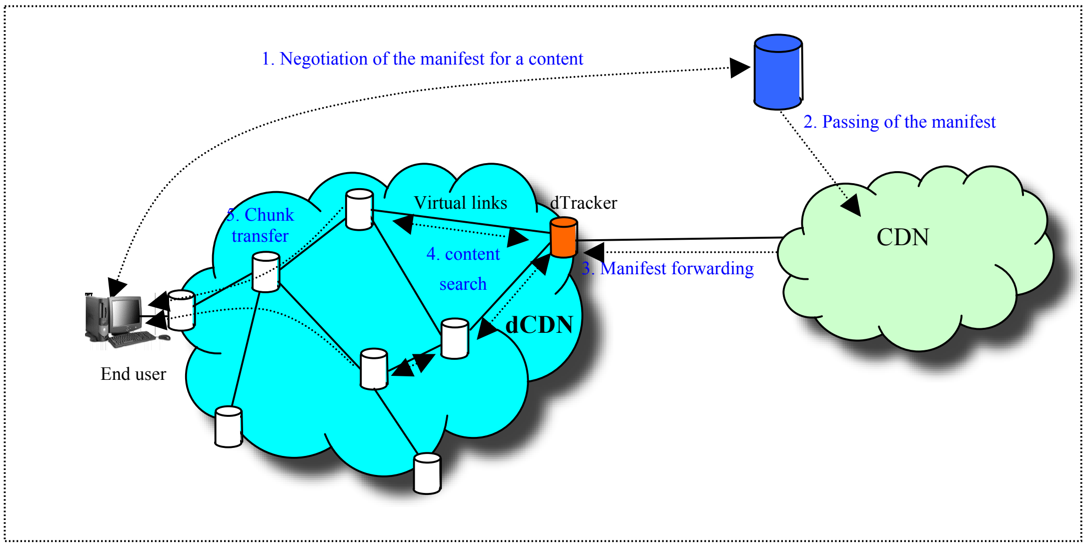


Figure 5 : Handling a user's request

Beyond the list of chunks stored in the nodes of the dCDN, the dTracker should maintain the logical topology of the dCDN, i.e., the map composed of the nodes and the logical links between them. This logical topology is then valuated by: the load of the link, the available bandwidth, the load of nodes, etc. The valuated logical map is computed by the dCollector described in the next section.

Prefetching is performed by the content provider, which is the only entity which can keep track of the request of the user and which can push content in advance in the CDN, which can then push content in the dCDN.

3.3 dCollector

The nodes sent information about their load, the bandwidth available to transmit chunks to the dCollector. End users could also send some reports about the quality of experience perceived when viewing content. This last kind of information can be used by the dCDN to report some QoE degradation due to packet loss, jitter, etc. up to the end user which can not be directly monitored by the dCDN.

Information sent by the nodes is used by the dCollector to construct the valuated topology map of the dCDN. For this purpose, the dCollector has to acquire the logical topology of the dCDN (i.e. the location of the nodes and the links between them). The dColletor could be part of the routing process (for instance a dead leaf) in order to acquire the topology and the weights of the links between the nodes. In an early deployment, this topology could be fixed by construction. While this latter possibility is rather rigid, the former could take into account routing fluctuations (rerouting, link outage, etc.) but is much more difficult to implement.

The relation between the dCollector and the dTracker is illustrated in Figure 6.

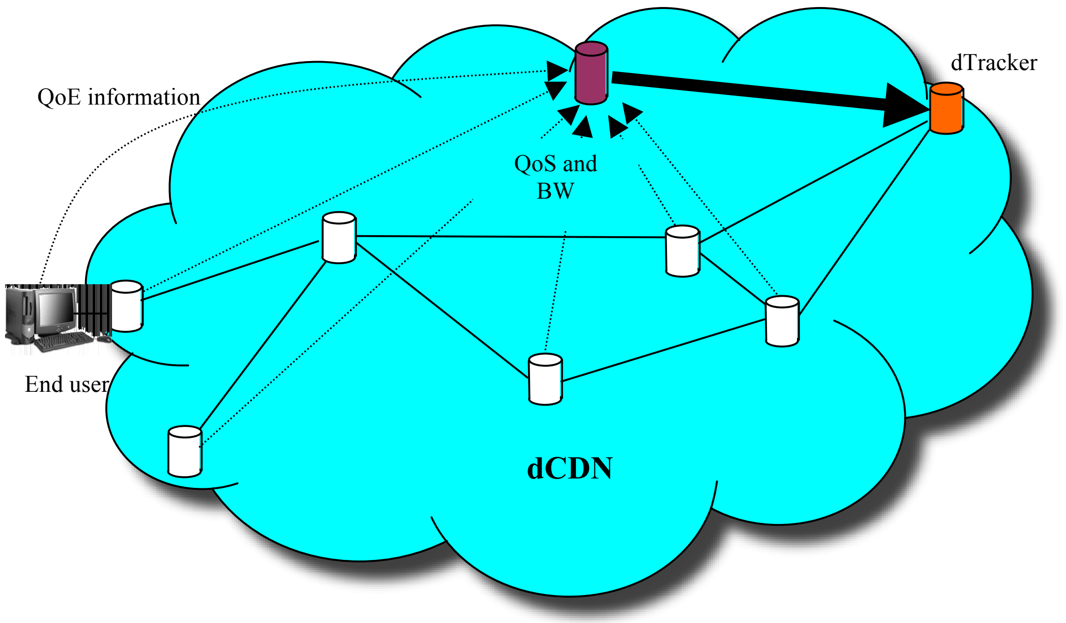


Figure 6: dCollector aggregating monitoring information

4 Running a dCDN

4.1 Offline upload of a given content

We assume here that the CDN has decided to delegate the distribution of content to the dCDN.

1. The server of the CDN pushes the content to the dCDN by sending it to the dTracker. The CDN may provide popularity measures for this content, together with a log of previous requests.
2. If popularity and usage logs have been provided, the dTracker uses its geographical database in order to localize past requests. The dTracker uses this information together with storage monitoring and network monitoring data to compute the placement of the content on the dCDN nodes.
3. The dTracker pushes the content to the dCDN nodes and updates the storage-monitoring database.

The aim of each placement is to minimize the storage cost on the one hand, and to optimize the expected QoE in terms of serving latency.

Step 2 is detailed below:

- The popularity of content is derived from the initial usage information provided by the CDN, and from the usage logs recorded during the dCDN operation. Each content should be present at least once in the dCDN system. Additionally, more popular content should have more replicas, and the location of the replicas can be based on the localization of the requests (e.g. if a content is requested more often in a given area, it is preferable to locate the copies close to this area).
- The capability of dCDN nodes is reported by the storage monitoring block which may be provided by the dCollector. The capability of each dCDN node includes its bandwidth capacity (how many clients it can serve) and the storage capacity (how

many videos it can store). The decision of video placement should not overpass the capability of each server.

- The network environment reported by the network monitoring block which is also provided by the dCollector. The network environment indicates the cost for pushing videos into each dCDN server and the latency for the clients to retrieve their recommended videos from dCDN servers.

The solution of the problem provides explicitly the placement of videos and their replicas in each dCDN node.

All the above factors can be taken into account in the replication process. Deliverable 4.3 reports on a computation based on a LP model of a special version of Facility Location Problem. A genetic algorithm parallelized by MapReduce is designed to quickly solve the problem even if the instance is large (millions of clients and thousands of servers).

The placement process implicitly pre-selects the server to which a client should be redirected to retrieve a given content.

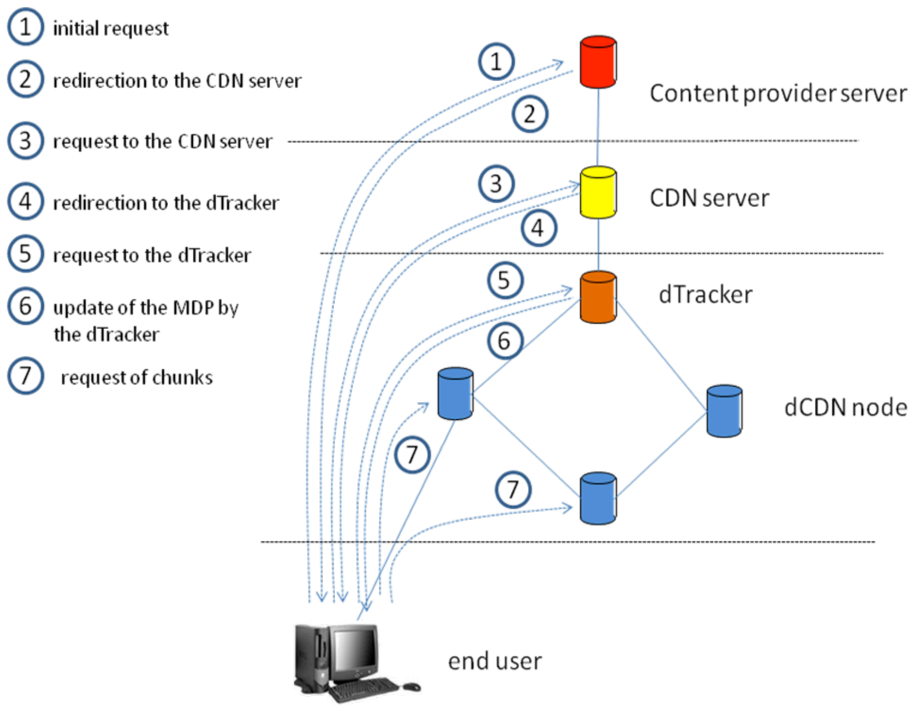
The placement of videos should be updated regularly according to the variation of the content popularity.

4.2 Requesting a given content

In this section, we assume that content can be uploaded online, as part of serving a user's request. The following describes the process of serving a user's request.

- The user sends a request for a desired content to the content provider (after having been redirected by a search engine).
- The content provider redirects the request to CDN, which in turns redirects it to a server which is considered as the most appropriate for the request (this is a standard CDN policy).
- The CDN's server pushes the content to the dCDN. The CDN server redirects the request to the dTracker.
- The dTracker pushes the content to the dCDN nodes that are e.g. closest to the end user (this is easy for a network operator by using routing information) and redirects the request to the appropriate dCDN nodes.

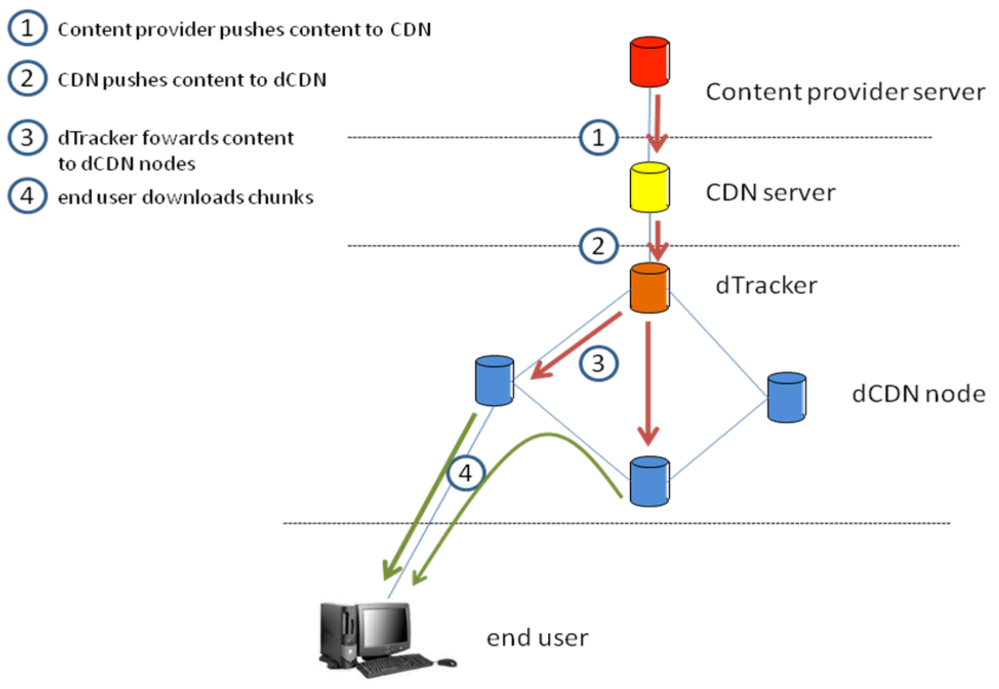
Figure 7 illustrates the redirection of requests so that the end user downloads the chunks from the appropriate dCDN nodes.



1 initial request
2 redirection to the CDN server
3 request to the CDN server
4 redirection to the dTracker
5 request to the dTracker
6 update of the MDP by the dTracker
7 request of chunks
Content provider server
CDN server
dTracker
dCDN node
end user

Figure 7: redirection of the request

The flow of data is illustrated in Figure 8. Content is pushed by the Content Provider into the CDN, which itself pushes the content into the dCDN via the dTracker. This latter then forwards the content to the dCDN nodes, which store the content, by enforcing some content replacement policy if necessary.



1 Content provider pushes content to CDN
2 CDN pushes content to dCDN
3 dTracker fowards content to dCDN nodes
4 end user downloads chunks
Content provider server
CDN server
dTracker
dCDN node
end user

Figure 8 : data flow for the content being pushed from content provider to dCDN and to the user

4.3 Normal operation

In this section we assume that the content has already been pushed into the dCDN. For the download of the chunks, two possibilities are available, either a centralized process or a P2P process.

4.3.1 Centralized serving process

The end user has received from a dCDN a manifest (or MPD) that points to virtual URLs. A virtual URL corresponds to a chunk in a given content. It is independent from both the localization of the chunks and the localization of the user.

The user requests these URLs to the dTracker, which then correlates the localization of the user with the localization of the chunk's replicas. The dTracker redirects the request to the appropriate replica.

4.3.2 Second scenario based on peer to peer principles

Another possibility for the download of chunks is to adopt a peer-to-peer approach. When the manifest is updated by the dTracker, a list of chunk request is sent to the selected dCDN nodes, which supposedly have the desired content, which are edge nodes. This leads to a slight modification of the global architecture via the introduction of different kinds of nodes: the VECs (ViPeer Edge Caching), the VCCs (ViPeer Core Caching), the VPPs (ViPeer Point of Presence) –see Figure 9.

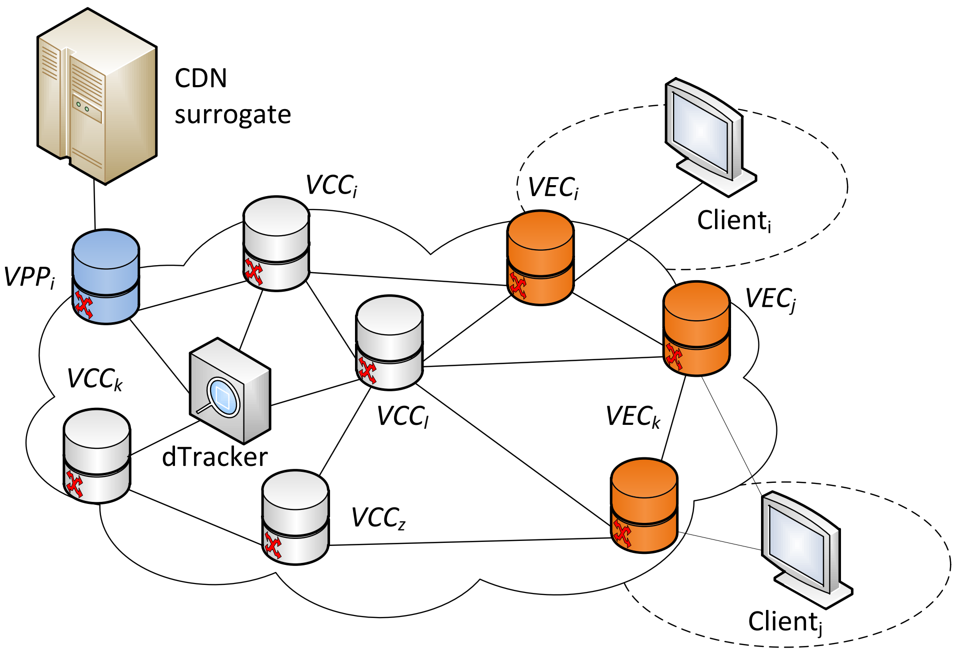


Figure 9 : Modified architecture for the P2P based solution

4.3.2.1 VECs/VCCs/VPPs

The VECs, VPPs and the VCCs are all elements of the intra-domain architecture, which allow retrieving the requested chunks. The only difference existing between the VECs and the VCCs is that the VECs in charge of receiving the requests from the end-users, thanks to the dTracker action. When the content is not present in the intra-domain, the VPPs are the network elements in charge of retrieving the content directly from the CDN. The content is then forwarded to the VECs following the request path.

In the control plan, these elements are based on a distributed P2P paradigm, just like the gnutella protocol (more details are given in the following).

After receiving a request from an end-user, the VEC, which is the entry point of the Intra-domain, checks whether the requested chunk is locally cached. If it is locally cached, the content is delivered to the end-user. Otherwise, a request for the chunk is forwarded to the direct neighbors. The same operation is performed until finding the requested chunks.

In the proposed system, each request is identified with a random number generated by the entry points to avoid duplicate requests and duplicate delivery of the same chunk. Moreover, if the request reaches the VPPs through all its neighbors, this last is in charge of requesting the chunk directly from the CDN.

We present in the following the detailed architecture by presenting the role of each element of the intra-domain of the proposed architecture, which comprises three major elements: the VECs "ViPeer Edge Caching", the VCCs "ViPeer Core Caching", the VPPs "ViPeer Point of Presence" and the dTracket "Distributed Tracker".

4.3.2.2 Role of the dTracker

Originally, the MPD describes a predefined content, including URL addresses of the pieces (i.e. chunks) of the video, which point-out initially to the CDN elements. As explained above, the dTracker is in charge of the modification of the Manifest to be ultimately sent to the end-user. The modification consists in updating the URLs of the chunks in order to point-out to one or more caching elements of the dCDN, or more precisely to the VECs.

Thus, the dTracker, first, selects the optimal[1] entry point(s) (i.e. VECs) for a particular client. The ***chunks*** to be streamed are, then, ***virtually shared between the selected entry points*** in a way to optimize resources' usage by enforcing natively load balancing without any support from traffic engineering or other network level mechanisms. Therefore, the selected VECs do not necessarily have to contain the requested chunks. This allows being ***completely independent from the caching strategies***, while supporting ***any type of clients*** with only requirement that the manifest is modified by the dTracker.

Just after sending the modified manifest (step 8, in Figure 10), a list LCR (i.e. List of Chunks to be Requested) of the chunks to be downloaded is sent to each concerned VEC, by the dTracker (step 9). As soon as the quality of the chunks to be requested is known, the ***VECs start proactively the download***. At late, the quality of the chunks is known after receiving the first request.

[1] It is meant by optimal the nearest VECs geographically or the VECs optimizing resources' use.

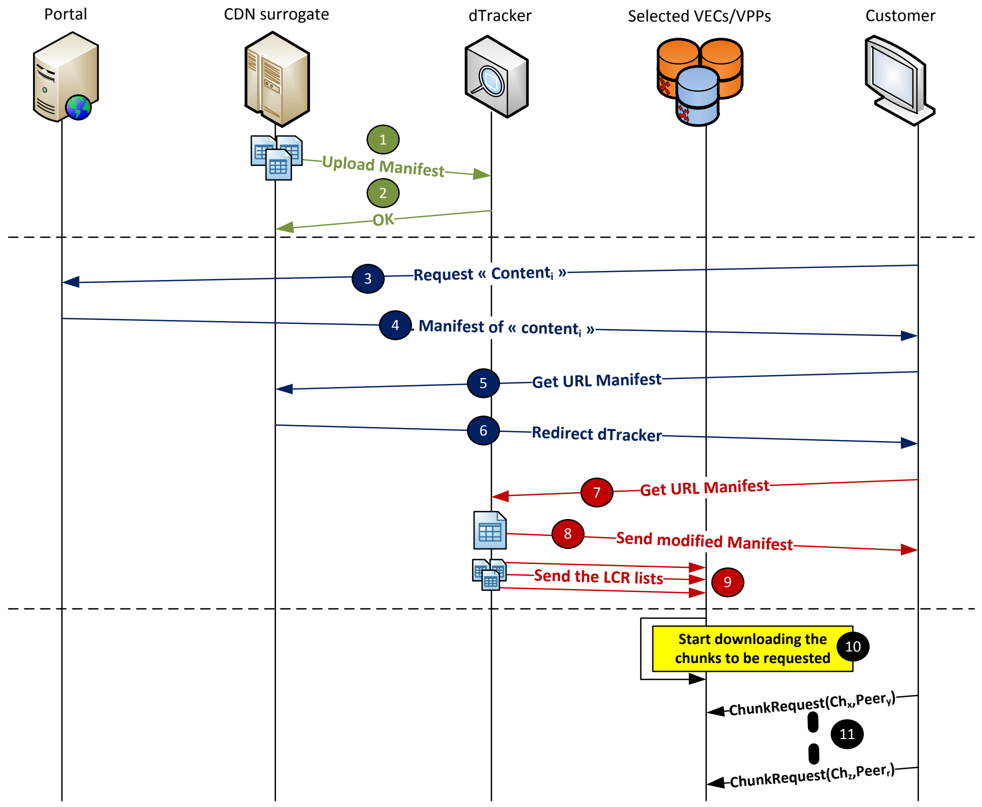


Figure 10 : Detailed scenario of communication for the P2P based solution

Note that the modification of the chunks' quality leads the VECs to download reactively the remaining chunks with the new requested quality. This may induce an increased delay. It remains, however, smaller than the delay incurred from directly downloading the chunks from the CDN.

5 Additional features of the delivery chain

5.1 Metrics reported by dCDN nodes to the Collector

In VIPEER, delay and available bandwidth were not identified at the beginning of the project, further studies and design choices of the system showed that they are needed.

5.1.1 Delay between nodes

The knowledge of the characteristics of the physical links between nodes is not enough to estimate in priori values of the delay as the dynamics of the network (routing decisions, congestion, etc) have a great impact on their values. To take into account this aspect, the dCollector must have an up-to-date knowledge of delays between nodes. Obviously, as this metric change continuously during time, that raises some information collection and measures orchestration matters to be discussed.

5.1.2 Available bandwidth

Network bandwidth continues to be a critical resource in the Internet because of heterogeneous capacities of access technologies and big sizes of exchanged contents. As the previous metric, the available bandwidth is changing during time. In Vipeer, the critical point is about the nodes, so what it is needed is to measure the available bandwidth between those nodes constituting the dCDN.

5.1.3 Available storage capacity of nodes in the dCDN

Once delay and available bandwidth are evaluated, another metric is needed: the available storage capacity of nodes in the dCDN.

As explained before, a movie is cut into small pieces named chunks. These chunks are then distributed in a subset of nodes. The dTracker has to know the free space in nodes to make the best decision about where to push content.

The available memory in nodes could be separate in two parts:

- a first one "reserved" for chunks that the dTracker has decided to push (for prefetchng reason) or to maintain (for redundancy).
- a second one with its local replacement policy

These two kings of memory are needed so nodes have to periodically send them to the dCollector. This periodicity is a real concern that it is address later.

5.1.4 Globals considerations

The metrics involved in VIPEER are dynamic and expensive (due to active measurements). So two essential questions are:

1. Where measurements have to be done?
2. How often measurements have to be peformed?

As the network operator operates the dCDN nodes, it is easy to install software in them in order to make the active measurements. Furthermore, as the dCDN is made to cover the last mile, it is accurate to have the metrics at this level: a node could be a representative of a local area.

For the frequency of measurements, we have to find a tradeoff between the overload and the scale time. In one hand, as the measurements are active, packets are sent and that could cost in bandwidth if they are made too often. In another hand, the dTracker has to know the real condition to take accurate decision. Typically the periodic time scale could be a minute.

The last thing is how to keep this metrics? The dCollector load dedicated databases for the delay, the available bandwidth and the available strorage capacity of nodes. The specific topic of database will be described in the next section.

5.2 Data base management in the dCDN

To maintain list of chunks in the VIPEER project, and all the other records (geographical information, usage information, metrology information), it is necessary to select database management systems that can support very large numbers of entries (e.g. each content may result in thousands of chunks), which may be read very frequently (each user may generate as many reads as there are chunks in a content) although they are written less often (when pushing new content or reorganizing the storage space e.g. according to modified popularity levels).

Although the project has not yet taken any final decision regarding the data base management selection, we have started at looking at the so-called "no-SQL" data base management systems. If we refer to the wikipedia definition [1]:

In computing, ***NoSQL*** *is a broad class of database management systems that differ from the classic model of the relational database management system (RDBMS) in some significant* *ways, most important being they do not use SQL as their primary query language. These data stores may not require fixed table schemas, usually do not support join operations, may not give full ACID (atomicity, consistency, isolation, durability) guarantees, and typically scale horizontally. Academic researchers typically refer to these databases as structured storage, a term that includes classic relational databases as a subset.*

Although Cassandra is probably the most publicized NoSQL database management systems, it is not the only one. A detailed taxonomy and analysis of these is given in [2].

In the context of VIPEER, we need a way for storing lot of data's and the data structure can change at anytime. And in the same time, we need to consider the performances in case of writing/reading data. Previous benchmarks [3] seem to indicate that Cassandra could be a good choice.

WP4 shall further study the data storage issued in order to identify the best approach for VIPEER.

5.3 Content adaptation and Digital Rights Management issues

WP3 has identified the usefulness of Adaptation Engines (AE) within the dCDN architecture. Relying on AE within the dCDN allows the reduction of the amount of storage by caching only those chunks that are actually requested by customers, instead of all available chunks referenced within the manifest. However, this requires a trust relationship between the Content Provider and the ISP, which may not exist. This problem is irrelevant for many DRM-free video contents, and AE can be co-located with storage space in dCDN nodes in order to produce on-line the requested chunk from a differently coded version of the same chunk.

However, content protected by DRM can be transcoded only if the necessary keys are distributed to the AE. As the contractual relationship between the CDN and the content provider may not permit the delegation of content access control, we have assumed in the VIPEER project that the access to DRM protected content is still controlled by the CDN which distributes the appropriate keys, while the content itself is distributed by the dCDN.

6 Conclusion

VIPEER assumes that video is distributed using HTTP adaptive streaming. Public CDNs delegate content distribution to an ISP operated dCDN.

When a CDN delegates the distribution of a given content to the dCDN, the latter has to copy this content on its set of servers (the dCDN nodes). The dTracker is responsible for computing the location of the various replicas. The placing process can take into account various factors such as

- The expected popularity of the content
- The state of storage nodes
- The state of network links

The ISP constantly monitors storage spaces and network states, and makes the monitoring measures available to the dTracker by the dMonitor.

The dTracker also keeps track of the copied replica and their location, in order to control the redirection process that is used to point customers to a replica of the requested content.

Keeping track of monitoring, log and storage data implies controlling large and possibly distributed database, which must be very efficiently read.

The present deliverable has identified two major variants for operating the dCDN:

1. A centralized variant pushes new content and constantly controls the replication process in order to optimize storage utilization and server latency. Customers requests are redirected to a dCDN node according to the knowledge of the location of the various replicas of the requested content
2. A distributed variant operates like a peer-assisted server. The set of storage servers are organized as a P2P content repository network, which can also dynamically request new uploads from the CDN in order to serve customers' requests.

Both variants present the following set of building blocks:

(1) an interface with a client CDN,

(2) a policy for caching and duplicating chunks in the dCDN nodes,

(3) mechanisms for monitoring available storage space, link bandwidth and delivered QoE,

(4) protocols for serving customers by selecting for each demand and each customer the appropriate dCDN node where the requested content is cached.

The present deliverable has identified the functions required to implement these building blocks, and their relationships (see Figure 2), however, it does not address the actual implementation of the presented functional blocks.

In the remainder of the project,

- WP5 shall develop a platform to provide such an experimental implementation;
- WP4 shall assess the efficiency of various content replication and distribution policies
- WP1 shall provide a constantly updated state of the art review of proposed commercial systems that present similarities with VIPEER (e.g. cloud based CDNs, HAS, peer-assisted CDNs, etc.)

7 References

[1] http://en.wikipedia.org/wiki/Nosql

[2] http://dbpedias.com/wiki/NoSQL:Survey_of_Distributed_Databases

[3] http://techblog.netflix.com/2011/11/benchmarking-cassandra-scalability-on.html