Deliverable D2.4.1

Design specification for the platform components
Start date of the project: January 2004
Duration: 48 months
Project Coordinator: Prof. Dr. math. Friedhelm Meyer auf der Heide
Heinz Nixdorf Institute, University of Paderborn, Germany
Due date of deliverable: December 2004
Actual submission date: February 2005
Dissemination level: PU – public

Work Package 2.4: Integrated Application Testbed (IAT)
Participants: Telecom Italia Learning Services S.p.A., Rome, Italy
University of Paderborn, Germany

Author of deliverable: Giovanni Cortese (g.cortese@computer.org)
Roberto Stomeo (UE006123@guest.teelcomitalia.it)
Federico Morabito (federico.morabito@telecomitalia.it)
Fabrizio Davide (fabrizio.davide@telecomitalia.it)
1 Introduction

End-system or application-level multicast has become an attractive alternative to IP multicast. Instead of relying on a multicast infrastructure in the network (which is not widely available), participating hosts route and distribute multicast messages using unicast network services. Our own group’s interests focus on large-scale applications environments, and in particular on application-level multicast in peer-to-peer (p2p) or cooperative environments. In particular, we have designed and implemented an application to distribute high volume multimedia content to large numbers of users. The application uses DHT-based middleware, developed by TILS in WP 2.3., as the basis for the routing substrate and the overlay network. The application enables multicast at the application level, by creating a forest of nodes to distribute content to participants. A resource reservation mechanism, shares the bandwidth available on each node. Message passing mechanisms are used to create an overlay network, thereby optimizing content distribution. A distributed testbed is used to evaluate the performance of these solutions.

2 Self-managing IT infrastructures

Scalable, highly decentralized systems require enormous resources for network management and maintenance. To continue growing, future systems will require higher automation and autonomous management. Research from the networking and P2P community provide insight on how to build mechanisms for self-management and self-stabilization into system infrastructure.

Overall, we believe that in the future, applications serving thousands of users (e.g. in large enterprise scenarios) or millions (as in the case of Internet scenarios, applications on mobile phones, sensor networks . . . ) will be based on middleware layers which are aware of the current computation context, which are fault-tolerant and which do not require human intervention to deal with the majority of system events (e.g. addition or removal of resources or functionality).

The discussion below, which is based on [5], describes the challenges we face as we move from today’s centralized, managed paradigm (in which services run on professionally managed servers in locked computer rooms), while providing necessary performance and reliability guarantees to users. Next generation systems, we believe, will be based on a massively distributed network of interacting peers, cooperating to provide service to users; many of these peers will be untrusted computers which may seek to exploit vulnerabilities of the system, damage the system, compromise privacy etc.. They will also be unreliable (i.e. built on systems which are not professionally managed, and which may crash or fail at any time).

In this kind of system, the only thing that can be trusted is the aggregate behavior of peers. To achieve reliable behavior, we need ‘thermodynamic design’. Systems made up of a large number of peers possess latent order. Entropy increases as the system operates, and new data or peers enter or leave the system. Self-organizing behavior reduces entropy by reorganizing and moving data or pointers. Active entropy reduction is performed by introspective systems i.e. systems that use spare computational resources to collect and analyze data, and adapt system behavior accordingly. Introspection - an idea first introduced by IBM [6] - is one of the key principles underlying the construction of autonomic systems.

Reliable behavior in a P2P system requires a combination of redundancy, cryptography, and thermodynamic stabilization, with active system repair. A growing research community is pointing to DOLR technology, coupled with replicated and cryptographic data management, as a way to design a system which conforms to these principles.

Future applications will be integrated in middleware with built-in management capabilities. In what follows, we will discuss content delivery networks and other solutions for content distribution: an applications family which can benefit from the approach just described.
3 Application-Level Overlays for Content Distribution

In this chapter, we introduce Content Delivery Networks (CDN), a new way of providing Internet services that makes it possible to distribute content to end users within an enterprise organization. Effective CDNs guarantee efficient use of network resources and significant reductions in user perceptions of response times.

3.1 Content Delivery Networks: an Introduction

User perceptions of the performance of a telecommunications-based service networks have become increasingly important. The traditional model for delivering Internet services, a centralized server responding to users’ requests, has proved inadequate to the task of providing services which are trustworthy, fast, and easy to access. The weakness of the traditional model has led to the development of a wide variety of solutions intended to deliver better performance, by optimizing the use of network resources.

Content Delivery Networks (CDNs) are now the most widely used solution, coming closer than other systems to meeting theoretical expectations. At present, CDNs are the most widely used and effective means of distributing content to large numbers of users in controlled settings, such as enterprise networks.

End-to-end performance in the Web environment depends on a wide range of factors, including client-server connectivity, network lossage and latency, the power of the server, and delays in name resolution. Content-serving architectures have a significant impact on some of these factors, and on various parameters not strictly related to performance, such as cost, reliability, and ease of management. In a traditional content-serving architecture, such as the architecture in Fig. 1, users request content from a single location. In this case, performance and scalability can be improved by increasing the number of servers, but it is not possible to improve performance when the network is congested. This approach may also be expensive, as the server must be large enough to handle sudden fluctuations in demand.

![Figure 1: Centralized architecture](image)

It is possible to overcome poor performance, due to network congestion or peaks in demand, by placing the most requested content in server caches located at the edge of the network. Such a distributed network of servers is a Content Distribution Network (CDN), as shown in Fig. 2.
A CDN is basically a distributed system of data storage making it possible to improve the quality of traditional services and to efficiently distribute high volume audio and video streams.

The main components of the architecture in Figure 3, are:

- The origin server, that is, the site where the data to be distributed is held;
- The surrogate server (cache engine), that is, the server in the access POP, authorized to distribute content on behalf of the origin server;
- The content distributor, which distributes content to the caches;
- The Global Load balancer, which routes users’ content requests to the surrogate site offering the best performance;
- End users, who access content from different network access points, offering guaranteed quality of service (QOS).

A CDN distributes content to clients from one of the CDN servers, with the result that perceived performance, in terms of response time, is better than for content distributed directly by the origin server. CDN architectures improve on traditional architectures by:

- distributing content from the edges of the network, close to the user, thereby improving user perceived response times;
- using several CDN servers to distribute images and multimedia files, thereby reducing the total load on origin servers;
- replicating data at several distributed locations thereby improving its availability.

CDNs play a central role in content-serving strategies. However, an effective increase in performance is only possible if the edge-nodes designated to serve client requests make it highly probable that the content requested is available locally; that is, if the edge-nodes have high hit rates. In this case, service requests will be fulfilled directly by one of the surrogate servers, making it unnecessary to
obtain the content from the origin server, i.e., by using a separate connection that would necessarily involve congested network links.

The need to guarantee high edge-node hit rates means that the replication and storing of content on surrogate servers (i.e., content replication and content caching) has a fundamental role. The basic structure of a surrogate server is a content cache that acquires and memorizes the content transferred to it, by replication, from content providers in the CDN. The distinctive functionality of a network of this type is, therefore, the distribution of content in multiple caches located “near” end-users, with the aim of reducing service latency and response times, perceived by users.

Content caching — in conjunction with Request Routing and Redirection mechanisms — make it possible to increase the typical performance of CDNs, and to reduce the load on the origin server. However, the effectiveness of caching strategies is strongly dependent on content replication and content caching policies. These are decided in a centralized and static way, on the basis of the types of clients to be served, their location, and their available resources. In cases in which the demand for services is dynamic and not known a priori, this approach may be inadequate. New problems arise when it is necessary to scale up the network. In practice, this is only possible when the size and characteristics of the user population are well known.

3.2 P2P infrastructures based on the CDN model

CDNs are dedicated collections of servers located strategically across the Internet, or in an enterprise Intranet. For example, the Telecom Italia unit involved in DELIS owns a large CDN serving the Telecom Italia group. Installing and managing a CDN requires significant investment and advance planning.

In the research community, there is a growing interest in exploiting application layer overlays for replica management. DHT-based middleware provides the basic mechanisms (Internet scale routing, directories, publish-subscribe, replica management) needed to build a P2P-based replication infrastructure, and scalable overlay-based multicast and parallel streaming. The DHT-based approach makes it possible to manage a large number of users without expensive over-sizing of network infrastructures.
In our work, we aim to exploit this work as the basis for a utility infrastructure for content distribution, bridging together the P2P model and commercial, CDNs. Such a utility infrastructure could include different kinds of computing nodes, which could belong either to a commercial CDN provider, or to individual users, as in conventional P2P models.

The proposed content delivery infrastructure could provide a broad range of different services including collaborative web-conferences, Web-TV, and distribution of sensor data. The application we have developed allows the distribution of content via a collaborative mechanism that enables every node to share its available bandwidth. This makes it easy to implement services. To date, the project team has implemented a content distribution service offering generic content, an on-demand Web-TV service, and a live Web-TV service (including collaborative web-conferencing).

The application is based on a CDN architectural model. The main novelty is that the architecture incorporates the DHT-based middleware developed in WP 2.3. This makes it possible to create an overlay network, in which each node produces a multicast service at the application level [4]. Hosts send and distribute multicast messages via unicast network services. This approach avoids the need for a multicast infrastructure on the network.

3.2.1 General Characteristics

The hardest problem for current distribution systems is the transmission of high volumes of content, such as video or very large files. The application we have developed resolves this problem by using the cooperative properties and the message-passing algorithms of peer-to-peer systems.

In particular we have implemented Splitstream [1], a multicast infrastructure for collaborative settings based on Pastry. Its qualities of reliability (the ability to restore normal functioning of the network after catastrophic events), self-organization (the ability to maintain the network in a condition of distributed knowledge), and locality (efficient routing), are inherited from the general properties of the DHT implementation.

The key feature of the application is the generation of a “forest” of multicast trees for content distribution. The content to be distributed is split into K stripes. Each stripe has a multicast tree address that is different and separate from those of other stripes. As a result, every node in a multicast tree is also a leaf node for other multicast trees in the forest.

The separate generation of multicast trees ensures that the system load is balanced/shared between all the nodes of the multicast forest. For example, if one node wishes to receive K stripes and the other nodes are prepared to transmit K stripes, the application builds a forest of multicast trees so as to balance the load evenly between all nodes. This involves only a slight delay in transmission and a minimal use of system links. Figure 4 illustrates the construction of a multicast forest.

In this example, the original content is divided into two stripes, each of which follows a different multicast tree in the forest of multicast trees. The load is automatically balanced between all the nodes involved in the distribution. Figure 4 illustrates how each node is both an internal node for one multicast tree and a leaf node for the other multicast trees.

3.2.2 Managing Content by resource reservation

Let us assume that the original content to be transmitted requires bandwidth $B$, and that each stripe has a bandwidth that is half the amount of bandwidth required ($B/2$). Each node will receive both stripes. The maximum output and input bandwidth of the system is thus $B$. The system is thus able to speed up transmission of the content, by enabling each node to receive all stripes. As a result, the entire content is transmitted from the source node in multicast mode via several distributive trees.

In general, content is divided into $K$ stripes. Participating nodes receive a subset of stripes. To manage their input bandwidth, nodes can request an increase in bandwidth of $B/K$. In a similar way, nodes can control their own output bandwidth, and limit the number of “node sons/children” they
adopt. This allows the application to coordinate nodes with different bandwidths, both for input and for output. Nodes which make their bandwidth available to the application can autonomously manage the optimal allocation of this bandwidth.

### 3.2.3 Implementation

As is usual with P2P systems, each node can function both as a source and as a client node. If a node wishes to distribute content, it creates a topic of interest, with a topicID. The node nearest to the topicID becomes the service’s node of reference (rendezvous point). All nodes belonging to the network can subscribe to this topic by using the topicID, thus becoming subscribers to that topic. A subscriber node will automatically receive every piece of content generated by any node that decides to become a source of content on that topic.

The source node generates the distribution channel. This consists of a number of stripes that satisfy the bandwidth criteria for every node, and the criteria for the maximum size of file content. Every stripe is identified by a stripeID. In this way, the system creates as many multicast trees as there are stripes. The result is the self-organized creation of a multicast distribution forest.

After the set up of the distribution network, the P2P network begins to transmit content. All application services use the same method of content distribution, applying differentiated encoding and decoding techniques according to the size and nature of the files to be transmitted and the quality of service required.

### 3.3 Services

The application we have developed makes it possible to implement a range of different services. These include:

- Content Distribution
- Streaming Video on demand
- Virtual Classrooms
3.3.1 Content Distribution

The “Content Distribution” service allows the distribution of files of any type (e.g. .pdf .ppt .pps .txt .mpeg .avi, . . .) to nodes that have subscribed to the service. There are many possible applications. One example might be an application to update software (for example, an anti-virus application) on any machine that the end user decides to connect to, thereby guaranteeing that the user remains secure and protected, even from viruses that that machine itself is/was not properly protected against.

To access the service, users use the middleware developed in WP 2.3. This sets up the overlay network, data management functions and the communications channels required. When the network recognizes a node, it is assigned a nodeId and a list of available “topics”. Topics represent subject areas where new content is available for distribution. After the node decides which topic to subscribe to, it receives all new content that has been published on that topic. Any node with the appropriate credentials can be the source of new content.

Content distribution takes place via the transmission of byte buffers at the overlay network level. After the network has been generated, the source node for content distribution uses an appropriate algorithm to carry out the following sequence of operations:

- It encodes the file to be transmitted into individual bytes;
- It generates an array of bytes, which contains the complete encoded file;
- It splits the array into stripes;
- It transmits stripes as streams of bits.

Content transmission thus uses the bandwidth made available by all nodes in the forest for the topic. Nodes that have not subscribed to this topic forward content; nodes which have subscribed, receive, encode and use the transmitted content. Having received the content, subscriber nodes use an appropriate algorithm to carry out the following sequence of operations:

- Reconstruct the array of bytes, in the correct order, from the stripes received;
- Encode the file, using the appropriate extension;
- Store the file received.

3.3.2 Streaming Video

We have developed a streaming video service that enables video to be transmitted and viewed at the same time on different machines. Each node in the network transmits video content, after creating an appropriate topic related to the video to be transmitted. Every node that subscribes to the topic receives the video content. Each node can decide, to transmit video content to other subscriber nodes. The communication mechanism is topic-based publish/subscribe, where topics represent a group of nodes interested in receiving topic-related content.

We have implemented two types of streaming video, **Streaming Video on-demand** and **Streaming Live Video** (for Virtual Classroom applications).

We used Java Media Framework for the encoding and real time viewing of video film, and integrated it above the ALO-CDN infrastructure for the distribution of video content.

3.3.3 Streaming video on demand

The **streaming video on demand** service can be distributed by a generic node that functions as the source of the service. Video content is pre-recorded or stored on the file system of the source node. All subscriber nodes receive the video, which can be viewed without using local storage.
The source node is equipped with video-decoding software that is integrated in the ALO-CDN infrastructure. The infrastructure transmits the video in single video stripes.

The decoding software carries out the following sequence of operations:

- It decodes the video and converts it into a stream of bytes;
- It generates the streams of bytes to be distributed;
- It splits the streams into stripes;
- It transmits the stripes.

The subscriber node is equipped with encoding software, which enables it to reconstruct the distributed streams and view the video.

The encoding software on the receiving node carries out the following sequence of operations:

- It reconstructs and orders the streams received by every stripe;
- It encodes the video for viewing, using Java Media Framework;
- It manages the buffer for the incoming stream, avoiding any interruption/discontinuity in viewing the video.

### 3.3.4 Live Streaming video

The live streaming video service makes it possible to view a video made by a recording device (webcam, video camera...) in real time. The live streaming video service can be distributed by a generic node, which functions as the source of the service, and is equipped with an appropriate recording device.

The decoding software on the source node carries out the following sequence of operations:

- It acquires video from the appropriate dedicated device;
- It decodes the video and converts it into a stream of bytes;
- It splits the streams created into the stripes that have been generated;
- It transmits the video stream.

Nodes subscribed to the service receive the streams sent by the source node, and view the video using the encoding software.

The encoding software carries out the following sequence of operations:

- It reconstructs and puts in order the streams received by every stripe;
- It encodes the video for viewing, using Java Media Framework;
- It manages the buffer for the incoming stream, avoiding any interruption/discontinuity in viewing the video.

The video is viewed as it arrives, in a stream, and it is not stored locally. In this way it remains streaming video, as defined.

This service makes it possible to simultaneously transmit and view live videos, in a multicast mode that provides efficient service to many users.

It should be emphasized that there is a slight delay between the time when live video being shot and the time when it can be viewed at client nodes. After a node has received a video stream, it can retransmit it, using the same topic (which in this case is associated with the Virtual Classroom).

The forest can be used for streaming to all subscribers to the service.
4 Testbed

The project team has designed and defined a distributed testbed, to analyze the performance of the architecture developed in WP 2.3 and the distributed services proposed by WP 2.4, and other distributed applications developed by the DELIS partners. To this end, the team has set up a node in the PlanetLab network [7,8]. The node is publicly accessible via the Internet.

PlanetLab is a geographically distributed overlay network designed to support the deployment and evaluation of planetary-scale network services, by providing an overlay-based testbed. PlanetLab is based on the following design principles: services should be able to run continuously and access a slice of the overlay’s resources; control over resources should be distributed; overlay management services should be unbundled and run in their own slices; APIs should be designed to promote application development.

PlanetLab will be used to evaluate the performance of the network management architecture of WP 2.3 and the application developed in WP 2.4. This requires the definition of the following parameters:

- The frequency with which information collected by agents is published;
- The size of the data to be published;
- Types of subscriber: topic-based subscriptions and subscription patterns (or long-running queries);
- Types of publisher: passive or active. A passive publisher simply collects information periodically and publishes it. An active publisher publishes information after modifying it locally, or only after certain kinds of events.

References

[8] www.planet-lab.org