Towards a Peer-Assisted Content Delivery Architecture

Bogdan Florescu, Mugurel Ionut Andreica

Towards a Peer-Assisted Content Delivery Architecture

Bogdan Florescu*, Mugurel Ionuț Andreica**

*Computer Science Department, Politehnica University of Bucharest
Romania (Tel: 0722 695 042; e-mail: bogdan.florescu@cti.pub.ro).
**Computer Science Department, Politehnica University of Bucharest,
Romania (e-mail: mugurel.andreica@cs.pub.ro)

Abstract: Currently, the most widespread model for data transfer uses a client-server paradigm (e.g. Content Distribution Network, or CDN). A totally different approach to content delivery is Peer-to-Peer (P2P) technology. While both technologies have proven their validity, a series of limitations affect each one: deploying CDN servers is expensive, and the overall transfer speed depends on the distributor's network bandwidth and processing power of the machine. Even though P2P architectures avoid these problems, for a successful data transfer they require a sufficient number of seeding clients, making them unreliable. In this paper, we propose a hybrid content delivery solution that tries to leverage the advantages of both approaches, while mitigating their weaknesses. This architecture aims at offering a viable data transfer model, with superior download speed, increased reliability and affordable resources.

1. INTRODUCTION

Over the last years, high-speed Internet connections became widely available, and, as a consequence, the quantity of transferred information has seen an exponential growth. Different models for data transfer have been proposed, of which two are highly used: client-server and peer-to-peer systems.

When speaking of the client-server paradigm, from now on we will refer to Content Distribution Network (CDN) technologies. Understanding this model is very straightforward: there are a number of CDN servers which store and forward content, and some clients that make data requests to these servers. Peer-to-peer (P2P) represents a newer architecture for delivering content: instead of an established server, every P2P node can be at the same time server and client, distributing already stored content, while downloading new content from other peers. These two very different models have their strengths, but also some important disadvantages. In this paper we will describe a hybrid approach, which offers better performance and scalability than either of these technologies alone, with the price of increased system complexity. First of all, let's analyze both of these paradigms to understand what advantages and disadvantages each one possesses.

1.1 Client-server architecture

As stated before, CDNs employ a client-server architecture: „a network architecture in which each computer or process on the network is either a client or a server. Servers are powerful computers or processes dedicated to managing disk drives (file servers), printers (print servers), or network traffic (network servers). Clients are PCs or workstations on which users run applications. Clients rely on servers for resources, such as files, devices, and even processing power" (Wikipedia, 2011a). Each CDN server distributes stored data to clients in its designated domain. A simple CDN architecture is shown in Fig. 1.

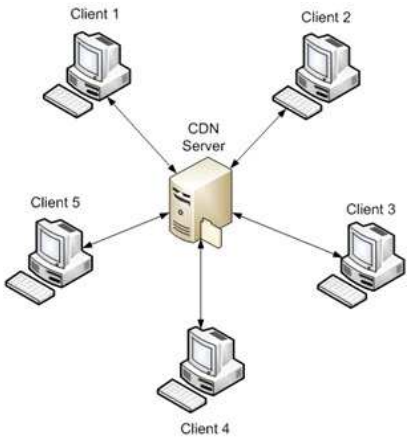

Fig. 1. CDN Architecture.

A CDN server has dedicated storage capacity and high-bandwidth Internet connection to address clients' data transfer requirements. Data management is simplified because the files are stored in one location. Ensuring information backup and security control is also an easy task. However, deploying and maintaining these servers is difficult and costly. Servers constitute a single point of failure, so in such cases the entire system can suffer from great performance loss or even become unable to deliver content to its clients.

1.2 Peer-to-Peer architecture

On the other hand, Peer-to-Peer technologies are based on a de-centralized architecture, which overcomes some of the

client-server disadvantages (note: in this paper, when referring to a P2P architecture, the reader should understand a pure P2P network). R. Schollmeier (Schollmeier, 2001) gives a formal definition: "A distributed network architecture may be called a Peer-to-Peer (P-to-P, P2P …) network, if the participants share a part of their own hardware resources (processing power, storage capacity, network link capacity, printers …). These shared resources are necessary to provide the Service and content offered by the network (e.g. file sharing or shared workspaces for collaboration). They are accessible by other peers directly, without passing intermediary entities. The participants of such a network are thus resource (service and content) providers as well as resource (service and content) requestors (servent – concept)". Figure 2 describes a simple P2P architecture.

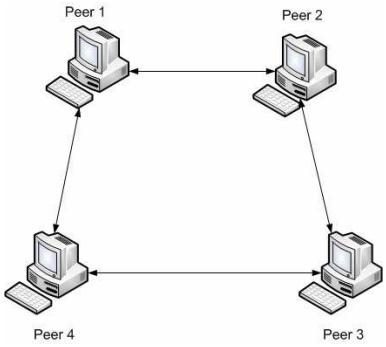

Fig. 2. Peer-to-peer architecture.

In this architecture, every client (called peer) is defined as: „a network node that can act as a client or a server, with or without centralized control, and with or without continuous connectivity" (Peer-to-Peer Working Group, 2011). After a peer receives a certain piece of data, it can provide that data to other peers. Two peers can establish a connection without intervention from a central server. A peer that is currently serving information is called a seeder, and a peer downloading is referred to as leecher. The essential advantage of this paradigm consists in the aggregated dissemination capacity of individual seeders (Xu et al., 2006). In contrast to client-server architecture, in P2P there is no single point of failure, which means that any peer can leave the network at any time, with no penalties on communication between the remaining peers. Peer-to-peer networks scale much better and avoid bottlenecks due to lack of centralization. Another important benefit is the opportunity to leverage clients' resources, such as computing power, storage capacity and network bandwidth. Unfortunately, P2P networks have a series of important disadvantages. A data transfer can begin only when a sufficient number of seeders are available. Usually, clients' connections are not designed for high throughput, compared to what a CDN server could offer. For sensitive content, peer-to-peer networks cannot enforce the required standard of security policies. Also, content availability depends directly on the seeder's availability. Instead of a centralized control, peers have to rely on other peers' will and resources to provide data. Regarding this problem, there needs to be developed a set of policies that ensure fairness and peer stimulation.

1.3 Hybrid architecture

This paper presents an architecture that retains the benefits from client-server and P2P models, but tries to eliminate most of their disadvantages. It can be viewed as a client-server architecture, with an additional peer-to-peer layer. At its core, the system relies on one or more content delivery servers to provide continuous access to information. First, new content will be available for transfer from these servers. Each client connects to one or more CDN servers, and starts the data transfer. When the client finishes the transfer of a piece of data, the server marks it as a possible seeder for that piece and adds it to an internal peer list. After this step, new peers can connect to seeders in order to transfer data. When a peer makes a data transfer request, the server decides who will be the source and establishes a connection. In this way, as the number of peers grows, the system can leverage their resources, while maintaining control over the information flow and network topology. Another issue needs to be addressed: as in classic P2P networks, some peer stimulation policies have to be implemented. In the absence of these policies, the system cannot always benefit from the P2P layer. A detailed system and policy description will be provided in the following sections.

The next section discusses related work in the field of hybrid content delivery networks. Section 3 describes a possible architecture design; section 4 presents some policies for stimulating peer participation in content distribution and section 5 quickly reviews chosen technologies for implementation. Finally, section 6 concludes this paper.

2. RELATED WORK

There are a lot of technologies available for transferring data over a network. However, the majority of them is either based on the client-server paradigm, or uses a peer-to-peer architecture. In the recent years, the shortcomings of these approaches became more and more evident. As a result, some development effort was spent for building hybrid solutions to address these disadvantages, while still benefiting from previous knowledge. However, there are currently no widely available and accessible hybrid solutions for data transfer over the network.

In their paper "Towards a Peer-to-Peer Extended Content Delivery Network" (Pakkala and Latvakoski, 2005), authors Daniel Pakkala and Juhani Latvakoski propose the concept of *P2P extension* for CDNs. The classic star topology CDN, consisting of distribution and edge servers, is extended with P2P overlay networks at three different domains. The architecture of the P2P extensions is hierarchical: the P2P-connected edge servers host and manage the extensions in the CDN service provider domain, that further host and manage the possible end user domain extensions. The authors' main focus in developing the P2P Extended CDN was to enhance the star topology CDN networks with a peer-to-peer layer, while still maintaing security. For a peer to enter the trusted domain of the P2P extension, it needs to be authenticated and authorized. Figure 3 presents this architecture, where red triangles and squares represent peers from the P2P CDN

extension. While the concept is very interesting, no real working prototype is provided. It addresses some security issues related to P2P networks, but this also adds increased system complexity, making it more difficult to implement and deploy. However, it offers a solid starting point for developing a hybrid CDN architecture.

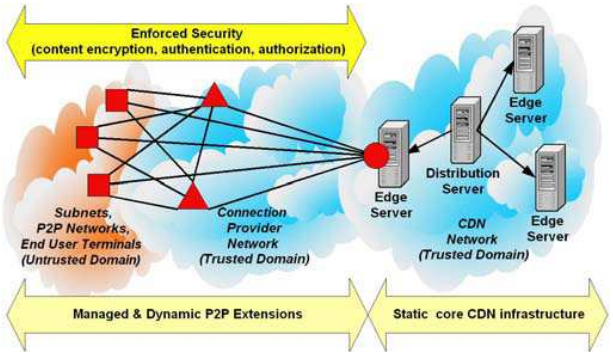

Fig. 3. P2P Extension (Pakkala and Latvakoski, 2005).

One successful commercial hybrid CDN implementation is Pando Networks (Pando Networks, 2011a). Pando Networks' technology is based on a modified version of the BitTorrent protocol. Its hybrid P2P and server-based network includes central control over file distribution, intelligent throttling between peers and servers, reporting/analytics and security. Pando leverages both P2P and HTTP protocols (peers and servers) to optimize content delivery for performance and cost efficiency (Figure 4).

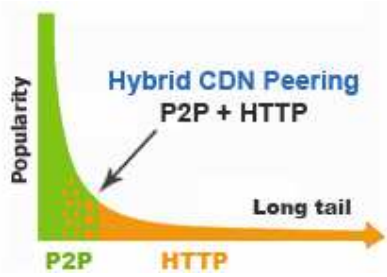

Fig. 4. Pando CDN peering (Pando Networks, 2011b).

Pando is a managed, hybrid P2P content delivery platform designed to maintain central control of content distribution. All networking communications are routed via Pando's trackers and web services. Consumers (peers) only supply bandwidth and storage to content they have proactively consumed. Moreover, Pando works as an add-on to an existing CDN, not as a replacement. The CDN is always considered the first and most reliable source of content. As peers demand more bandwidth, they also provide more of it. Demand and supply are directly proportional. As content becomes popular, more of it is served by consumers (referred to as "Peer Cloud"). Less popular content will continue to be served mainly by central CDN Servers, which act as "super-nodes" in a Peer Cloud. Pando's commercial success stands as proof of hybrid architectures' viability in current real-world networking context.

Another two interesting hybrid CDNs are described in (Ha, 2008) and (Xu et al., 2006). Both of them are aimed at offering media streaming services by incorporating P2P over a content distribution network. In (Ha, 2008), authors propose „a new hybrid solution based on the effective management of playing buffer at the peer-side to best equilibrate the bandwidth used between CDN side and P2P side". They achieve this by dividing the playing buffer into two parts with different priorities (CDN and P2P). During playback, missing packets in the CDN priority part will be received from CDN servers, and respectively, missing packets in the P2P priority part will be received from other peers (Figure 5). The authors hope to reduce the playback time by using CDN servers to get immediately some parts of the needed content during the playback process. Also, P2P technology helps reducing the cost for a Content Provider. The consumed bandwidth to deliver the content is provided by the CDN and all the consumers. This solution differentiates from other hybrid CDN-P2P systems by working at the application level.

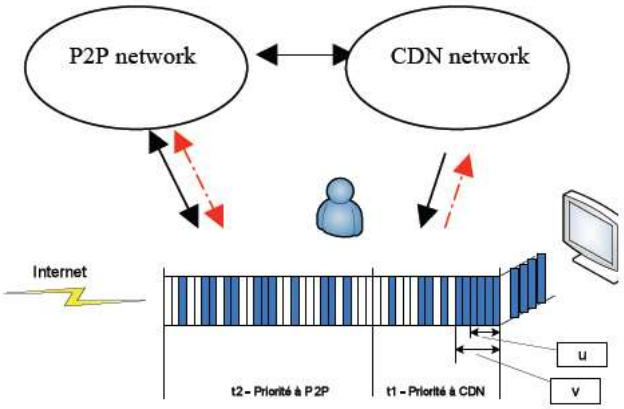

Fig. 5. Hybrid architecture for media streaming (Ha, 2008).

The paper (Xu et al., 2006) also presents a media streaming-oriented P2P-CDN system (Figure 6). Aside from describing the hybrid architecture with integrated capacity planning and runtime operations, the study includes a suite of limited contribution policies that advocate and reflect fairness toward peers. The authors proceed to analyze the impact of different policies and parameters on the progress, cost, and peer load of a media distribution process.

Although these two papers are concerned with media streaming problems, some ideas presented there can be successfully implemented in a more general content delivery system (e.g. prioritizing transfer buffer, contribution policies etc.).

3. ARCHITECTURE

3.1 Overview

As stated before, the proposed hybrid architecture will be based on one or more core servers, extended with peer-to-peer capabilities (Figure 7). Servers are the primary source

for data, making sure that content can be always provided even though there are no more peers seeding. In addition, servers will be responsible for initiating connections between peers, maintaining peer lists, enforcing policies and keeping a topology map. At any time, peers can enter or leave the network, without affecting communication between the remaining clients and servers. After a peer finished downloading a certain piece of data, it can choose to continue seeding it. A server will establish connections between peers only when its occupied bandwidth exceeds a defined threshold, in order to avoid bottlenecks.

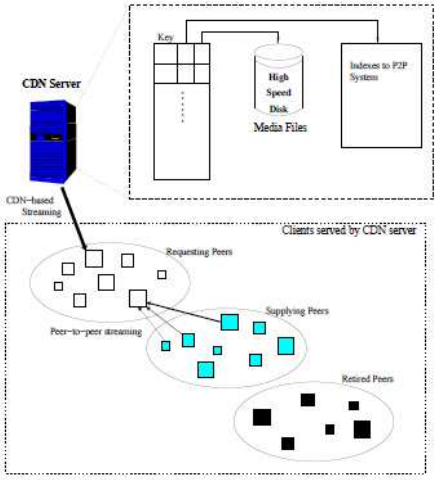

Fig. 6. Different hybrid architecture for streaming media distribution (Xu et al., 2006).

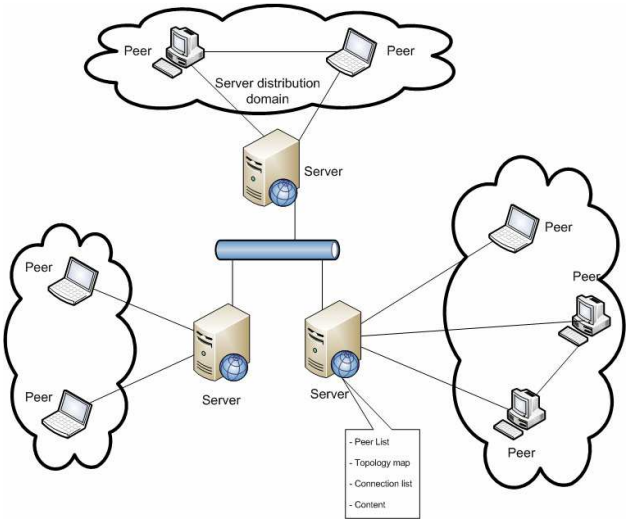

Fig. 7. Hybrid peer-assisted content delivery architecture.

In Figure 7, peers connected to a server are shown to belong to a server distribution domain, which means they can only communicate with peers and servers from the same domain.

Our architecture will be implemented as a service which can be accessed by clients. Generically speaking, clients submit requests to the service, which processes them. After processing a request, the service may send back a reply to the clients. The service maintains an interval service state (e.g. which, in this case, may consist of the files being served to the clients). The processing of requests may update the internal state (e.g. special clients may upload new files, modify or delete current files). Fig. 8 and 9 present the generic service view described in this paragraph.

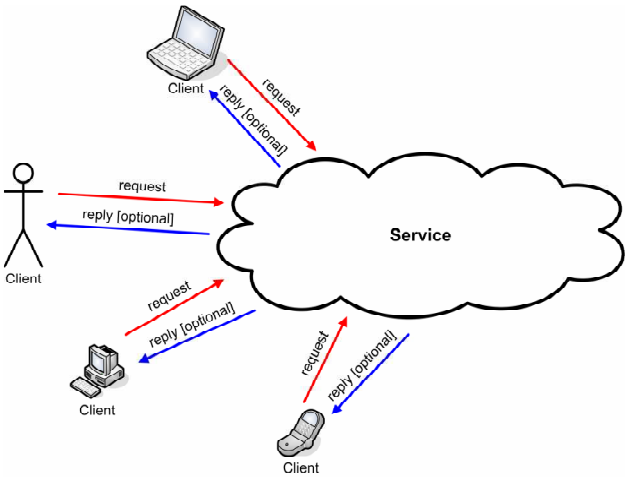

Fig. 8. Generic service model.

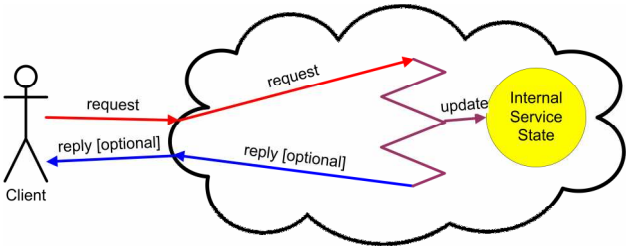

Fig. 9. Generic interactions between a client and the service (request-reply). Processing of requests may modify the internal state of the service.

3.2 Stakeholders

Peers/clients can be in one of these states:

- Downloading – consumer or leecher – it only downloads data from a server or from other peers. Either the client opted not to share any data, or the server decided not to initiate any connection to this peer.
- Uploading – seeder – it only serves previously downloaded content to other peers. In this case, the peer chose to share data and the server established connection(s) with other peer(s). The peer acts like a *lightweight distribution server*.
- Downloading and uploading – it provides content to other peers while downloading new content from peers/server.

CDN servers perform the following functions:

- Continuously provides content. Configured network parameters define how much bandwidth can be reserved

for upload, until the server decides to redirect new connections to seeders.

- Maintains active lists of connected peers. Periodically, the server sends keep-alive messages to refresh peer lists. It is useful to store additional information about each client, such as connection quality (based on round trip time of keep – alive messages). For every file stored, the server associates a seeder list.
- Maintains a topology map, which contains information about peers and cost associated with the network connection between them.
- Maintains a list of active connections, necessary for making decisions like moving transfers from a seeder to another seeder. These decisions will be taken in case a seeder quits the network, when a seeder with better connection joins the network etc.
- Enforces policies for motivating peers to continue seeding content even after they finish downloading. Also, policies have the role to ensure fair contribution between peers.

3.3 Network events

We will analyze, for now, two events that can happen across the network:

a) a peer/client enters the network;

b) a peer/client leaves the network.

A. Peer/client connects to network

Client

1. request server connection
2. request file download
3. receive connection details
4. if necessary, connect to remote peer
5. start transfer

Server

1. establish connection
2. add client to active peers list
3. store / update client preferences (file(s) requested, file(s) seeding, willingness to share content etc.)
4. if *server_used_bandwidth* is lower than *threshold*
    1. search file in available files list
    2. if found
        1. set transfer parameters
        2. serve content
5. else
    1. loop through connected seeders list
        1. search file in currently seeding list
        2. if found
            1. add seeder to temp list
    2. sort temp list descending by upload speed to client
    3. select first seeder with load lower than a threshold value (known as designated seeder)
    4. send transfer parameters to designated seeder
    5. send designated seeder's connection details to client
6. update connections list
7. update topology map
8. monitor client's activity within the network and periodically update its details

For optimization purposes, if a client had previously connected to the network, the server can first check its last peer connections, and select the one offering the best bandwidth. If security is enforced, before initiating content transfer, the server must validate security policies against client's security level.

Remote seeder

1. receive connection request from client
2. receive transfer parameters from server
3. establish connection
4. serve content

B. Peer/client leaves the network

Server

1. update active peers list
2. update topology map
3. update peer details (total transfer size, files transferred list, total and per-file ratios etc.)

3.4 Interactions between core servers

So far, we implicitly assumed that the core servers provide a *fully replicated service* to the clients, i.e. that each server provides the same functionality and has access to the same data (e.g. the files which are delivered to the clients). In order to achieve this goal (that of a fully replicated service), however, the core servers (CDN servers) need to communicate with each other and synchronize their data.

There are multiple possibilities for achieving this goal. The simplest one is to use a *data storage and retrieval service* for storing and retrieving the files. Then, the CDN servers are simply front-end servers which rely on other servers for storing and retrieving the files for them.

If a separate data storage and retrieval service cannot (or should not) be employed, then the files need to be stored on the same machines on which the CDN servers are running. In this case, the simplest possibility (conceptually speaking) is to have the files fully replicated on each CDN server. Then, when a new file is added (or is updated), it is propagated to all the servers.

The most general case consists of partial replication of the files. In this case, each file has several instances across several CDN servers. When a server *S* needs to send a file *F* to one of its clients, it first searches for *F* among its local files. If *F* is not found, then it will search for *F* among the other CDN servers – the more replicated *F* is, the fewer other servers need to be queried. After *F* is found at a server *P*, the server *S* may either transfer *F* from *P* (and then cache it locally) or may redirect the client to the server *P*.

In order to efficiently implement the behaviors described above, the CDN servers need to interconnect into a peer-to-peer overlay (also called a *service overlay network* (Duan et al., 2003)). In such an overlay, a server interacts directly only with its neighbors. It may also interact with non-neighboring servers, by routing messages through the overlay. A framework for developing peer-to-peer applications and services which can also be used for implementing a CDN service overlay network was presented in (Andreica et al., 2011).

Another aspect worth studying in the presented architecture is the fault tolerance of the CDN service. Note that clients (peers) may join and leave the system at any time, and even if none of them is willing to help others download a file faster, another client may always resort to a CDN server. Thus, the fault tolerance of the *normal peer overlay* is not that important, because a CDN server is always expected to be available. Thus, the CDN service has strong high availability constraints. Two important mechanisms for ensuring high availability are *checkpoint-restart(-replay)* and *dynamically adaptive replication*.

Finally, another important issue regarding CDN server interactions is data transfer performance. In order to synchronize their data, the servers may need to transfer large files between them in a short amount of time. Thus, data transfer speed may be an important factor (especially if the data transfer is performed as a consequence of a pending client request). Data transfer performance can be addressed at least at the following two levels: data transport protocol level and overlay level. On a protocol level there have been many attempts (e.g. (Kelly, 2003), (Gu and Grossman, 2007), (Iyengar et al., 2006)) to design data transport protocols with various characteristics (e.g. reliable delivery, high throughput, etc.). On an overlay level, communication overlays for multi-path data transfers have been designed and implemented (see, for instance, (Andreica et al., 2009)).

4. PEER STIMULATION POLICIES

This type of hybrid architecture relies on the P2P layer to achieve superior download speeds compared to a simple client-server model. In contrast to Peer-to-Peer, when seeders start quitting the network, the content remains available, but the advantages of the hybrid design fade away. The network starts behaving like a regular CDN system, with the added cost of managing the unproductive P2P extension. In order to avoid these situations, certain policies for stimulating peer contribution need to be developed.

4.1 Existing techniques

The BitTorrent protocol optimizes download speed using a *tit-for-tat* strategy (meaning "equivalent retaliation") (Wikipedia, 2011b). Using this incentive policy, a BitTorrent peer closes the connection with other peers that do not provide upload in return to the peer's own upload (Cohen, 2003). The upload slot is then allocated to a more cooperating peer. To allow finding more cooperative peers and also offer a chance to previously non-cooperating peers, a peer will choose periodically a random non-cooperating peer and allocate it an upload slot. In other words, BitTorrent peers give upload priority to other peers that provide the highest upload rate to them. While this method achieves a high level of efficiency, it doesn't offer any motivation for peers to provide content after their download is finished.

In their paper (Carlsson and Eager, 2008), authors Niklas Carlsson and Derek L. Eager suggest a new policy to address the shortcomings of BitTorrent's tit-for-tat approach. They present an analytic model of a „priority-based incentive mechanism which provides peers with strong incentive to contribute upload bandwidth beyond their own download completion". Their solution extends the tit-for-tat policy by enabling a peer to give a fraction of the upload bandwidth to high-priority class peers. The system classifies peers into two categories, high priority and low priority, based on their prior contribution. Usage scenarios show proof of achieving better transfer speed than a simple tit-for-tat policy, for both low and high priority peers.

4.2 Proposed policies

An interesting and well-proven idea could be borrowed from BitTorrent sharing communities, namely the concept of "ratio". Whenever a peer transfers a file, a ratio is calculated between how much data was uploaded and how much data downloaded:

$$R = U / D,$$

where *R* characterizes a file. There can be two types of ratios: a file-specific ratio and an overall ratio (calculated using total data size transferred). The goal is to ensure that peers are actively stimulated to contribute content even after their transfer is finished. This computed ratio will further be used to gain credits for a peer: for every ratio $R \geq 1$, the peer receives $\lfloor R \rfloor$ credits. These credits will provide a peer with some benefits:

- better download speed;
- higher priority for future data transfers;
- early access to specific content, etc.

Each new download consumes a credit, but a ratio greater than 1 earns new credits. The system can be enhanced for peers that suffer from a low-bandwidth connection: R can be calculated using time metrics instead of download/upload size. Thus, the new formula will be:

$$R = T_u / T_d, \text{ where}$$

T_u = upload time (seed time) and

T_d = download time (leech time)

The server performs tests for every peer connection, learning different network parameter values, and then decides which method is appropriate.

Using a different approach, the server can alter network parameters to limit transfer speed for each peer. The limit is dynamically calculated considering connection speed and file size being transferred, to ensure the download of a specific file ends only after a ratio $R \geq 1$ was attained. The goal is to guarantee that every peer contributes fairly to the network.

These two types of policies can be used together or only one at a time. A scenario when they can coexist is the following: for peers with a number of credits greater than *0*, the first policy will be applied, while for the rest the connection speed limiting policy will be used.

5. TECHNOLOGIES

5.1 Programming language

The peer-assisted content delivery solution proposed in this paper will be developed using Java SE 6 and additional software libraries/packages. Java was chosen due to the straightforward approach to using sockets and the general availability of resources (documentation, tutorials, libraries etc.).

One solution for developing applications with sockets is the java.net package. It provides a Socket class, that implements one side of a two-way connection between two Java programs on the network. The Socket class sits on top of a platform-dependent implementation, hiding the details of any particular system. Additionally, the ServerSocket class implements a socket that servers can use to listen for and accept connections to clients.

5.2 Additional libraries

Another solution is represented by the *P2P Sockets* (P2PSockets, 2011) package, a reimplementation of standard Java sockets on top of JXTA (JXTA, 2011). P2P Sockets allows us to gain much of the power of JXTA, such as NAT and firewall traversal, without being exposed to its complexity. It does this through ports of popular software projects, such as a web server and web services stack, to work on the JXTA peer-to-peer network. P2P Sockets also introduces implementations of java.net.Socket and java.net.ServerSocket that can work on the JXTA network, as well as a simple, light-weight, distributed and non-secure DNS system.

The JXTA platform is an open network computing platform designed for peer-to-peer computing. It employs a standardized manner in which peers advertise and discover resources, communicate and collaborate with each other. The JXTA platform is defined by six protocols (e.g. Peer Resolver Protocol (PRP), Peer Discovery Protocol (PDP), Endpoint Router Protocol (ERP) and others). ERP is the only required protocol. This offers great flexibility, by letting a programmer to selectively implement a subset of other protocols.

Flexibility is also provided in choosing the transport protocol, as JXTA supports TCP/IP, HTTP, Bluetooth etc. The layer of abstractization offered by JXTA makes it ideal for implementing a working prototipe for the hybrid architecture proposed in this article. Also, the wide array of supported protocols and services provide the possibility for testing different approaches and comparing results.

6. CONCLUSIONS AND FUTURE WORK

Today, there is more need than ever for an efficient data transfer mechanism. Although many system architectures manage to handle the task, no single model can ensure performance, scalability and reduced costs at the same time. In this paper, we propose a hybrid design that combines two well-known architectures (peer-to-peer and client-server) to achieve that goal. Related work in the area is discussed, along with some peer stimulation policies that aim to enhance the P2P layer.

Future work includes, most importantly, a working prototype, in order to validate the proposed solution. It must be designed in a manner that quickly allows changes at implementation layer, with minimum impact on other components. A network simulator also has to be used (or maybe developed from scratch), in order to test different scenarios and compare results with the simple P2P or client-server approaches. Finally, a comprehensive test suite will be developed.

Except for the implementation aspect, we will also focus on developing new and efficient techniques for the following possible cases:

- creation and maintenance of the P2P overlay by a single server;
- communication and coordination between servers (e.g. data synchronization or transfer of missing data from one server to another);
- creation and maintenance of a global P2P overlay including clients connected to all the servers (unlike the domain separation case which we considered currently);
- dynamic addition of servers
- peer stimulation policies.

ACKNOWLEDGEMENT

The work presented in this paper has been partially funded by CNCSIS-UEFISCDI under research grant PD_240/2010 (contract no. 33/28.07.2010), PN II - RU program.

REFERENCES

Andreica, M. I., E.-D. Tirsa, and N. Tapus (2009). A Peer-to-Peer Architecture for Multi-Path Data Transfer Optimization using Local Decisions. In *Proceedings of the 3rd Workshop on Dependable Distributed Data Management (WDDDM),* pp. 2-5.

Andreica, M. I., E.-D. Tirsa, and N. Tapus (2011). A Modular Framework for the Development of Peer-to-Peer Applications and Services. In *International Journal of Grid and Utility Computing*, special issue on „Advances in P2P Computing and Applications", Inderscience Publishers. In Press.

Carlsson, N. and D. L. Eager (2008). Modeling Priority-based Incentive Policies for Peer Assisted Content Delivery Systems. In: *Proceedings of the 7th International IFIP-TC6 Networking Conference on AdHoc and Sensor Networks, Wireless Networks, Next Generation Internet*, pp. 421-432.

Cohen, B. (2003). Incentives Build Robustness in BitTorrent. bittorrent.org.

Duan, Z., Z.-L. Zhang, and Y. T. Hou (2003). Service Overlay Networks: SLAs, QoS, and Bandwidth Provisioning. In IEEE/ACM Transactions on Networking, Vol. 11 (6), pp. 870-883.

Gu, Y. and R. L. Grossman (2007). UDT: UDP-based Data Transfer for High-speed Wide Area Networks. In *Computer Networks: The International Journal of Computer and Telecommunications Networking*, Vol. 51 (7), pp. 1777-1799.

Iyengar, J. R., P. D. Amer, and R. Stewart (2006). Concurrent Multipath Transfer using SCTP Multihoming over Independent End-to-End Paths. In *IEEE/ACM Transactions on Networking*, Vol. 14 (5), pp. 951-964.

JXTA (2011). http://www.jxta.org/

Kelly, T. (2003). Scalable TCP: Improving Performance in Highspeed Wide Area Networks. In *ACM SIGCOMM Computer Communication Review*, Vol. 33, pp. 83-91.

Ha, D. (2008). A Novel Hybrid CDN-P2P Mechanism for Effective Real-Time Media Streaming. M.Sc. Thesis.

P2PSockets (2011). https://p2psockets.dev.java.net/

Pakkala D. and J. Latvakoski (2005). Towards a Peer-to-Peer Extended Content Delivery Network. In: *Proceedings of the 14th IST Mobile & Wireless Communications Summit.*

Pando Networks (2011a). http://www.pandonetworks.com/

Pando Networks (2011b). CDN Peering. http://www.pandonetworks.com/cdn-peering

Peer-to-Peer Working Group (2011). Glossary for Peer-to-Peer. http://www.peer-topeerwg.org/tech/glossary.html

Schollmeier, R. (2001). A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. In: *Proceedings of the First International Conference on Peer-to-Peer Computing*.

Wikipedia (2011a). Client-Server Architecture. http://www.webopedia.com/TERM/C/client_server_architecture.html

Wikipedia (2011b). Tit for tat. http://en.wikipedia.org/wiki/Tit_for_tat

Xu, D., S. S. Kulkarniz, C. Rosenbergz, H.-K. Chaiz (2006). A CDN-P2P Hybrid Architecture for Cost-Effective Streaming Media Distribution. In: *Multimedia Systems*, Vol. 11 (4), pp. 383-399.