Structuring connections between content delivery servers groups

Jaime Lloret, Carlos Palau and Manuel Esteve

Department of Communications, Polytechnic University of Valencia, Camino Vera s/n, 46022, Valencia, Spain

Abstract

Content Delivery Servers can be grouped into logical networks according several criteria: similar characteristics; functionalities; type of data delivered or proximity. Sometimes, it could be useful to find a way to join these autonomous logical networks. Research in this area focuses in centralized solutions, e.g. using a single point for data delivery between these groups. This paper proposes a novel architecture to interconnect content delivery servers from different groups and explains how they establish logical connections. The topology of logical connections, between servers clustered in different groups, changes depending on their capacity or because of server failures. The proposal is based on a hierarchical structure, in which the role of the server in the architecture is given by several parameters, being also scalable and fault-tolerant. We present the analytical model for this interconnection architecture and results obtained for some defined parameters. Finally, developed an application which used the protocol described in this paper and we deployed the proposed architecture in a real environment to obtain accurate measurements.

Keywords: Content Delivery Networks; logical architecture; groups interconnnection; architecture design.

1. Introduction

The idea behind content delivery networks (CDNs) consists in placing separate servers, called surrogates, near client location. If the user is redirected to a nearby surrogate, which acts as proxy, it can experience a significant reduction in the perceived response time. CDNs act as trusted overlay networks that offer high-performance delivery of common Web objects, static data, and rich multimedia content by distributing content load among servers that are close to the clients. The communication process inside a CDN can be divided into two separate networks: (1) the distribution network, between origin site and surrogates, and (2) the delivery network, between surrogates and clients. The collection of surrogates that compound the CDN replicate content of the origin server [1].

The following paper considers a network which contains several groups of surrogates. Every surrogate in the network use the same application layer protocol and use it to interact with any other surrogate within their group. In this network, it would be required a content delivery between different groups. Three approaches can be used to join the groups:

- Using a single server of data transfer between those groups, which acts as a gateway. This solution undergoes three main drawbacks: (i) the server will have too many logical connections with other surrogates at the same time, so it will need too many resources. (ii) There is a central point of failure and a bottleneck. (iii) When a new group of surrogates joins the architecture, the server must be updated.
- Several to all, surrogates from every group would know about the existence of all surrogates from other networks because an independent entity informs them, so they will be able to reach every

surrogate from other groups directly. It implies that surrogates must be constantly updated to connect all other groups because of continuous surrogates joining or leaving and when a new group joins the architecture all surrogates must be updated. Moreover, surrogates will need to self-organize.

- The use of a common protocol to interconnect surrogates from different groups. All surrogates must be aware of this protocol. In this case, surrogates could have connections with surrogates from other groups based on several parameters such as available capacity, available number of connections, etc. This possibility allows the organization in several layers to manage connections between surrogates and it is highly scalable (the number of groups of surrogates could increase). When a new group of surrogates joins the architecture, the new surrogates would use this protocol to request connections with surrogates from other groups instead of updating all surrogates from all groups as it has been proposed in the previous approach.

The third proposal allows surrogate's self-organization, and also, fault tolerance and recovery procedures. The design could allow load balancing between groups when data is delivered. This paper introduces a novel architecture, and its protocol, that allows joining groups of surrogates based on the third proposal. It solves the problem of knowing which connections have to be established between surrogates from different groups taking into consideration surrogate's capacity. Although there are different discovery mechanisms in the literature, such as the ones used in pure P2P networks [2], hybrid P2P networks [3] [4], and unstructured P2P networks [5] or in content delivery systems [6] and in distributing systems [7]. The previous mechanisms do not consider peer grouping to structure the connections between clusters and in none of them connections are established using the capacity of the nodes.

The architecture proposed could be easily deployed over the model described in RFC3466 [8] by the Content Distribution Internetworking Working Group [9]. It can also be used to join different Content Delivery Networks such as the ones developed by service providers Akamai [10], Speedera [11] or CODIS [12], the ones developed by different vendors, such as Cisco's ECDN solution or Nortel's Content Director/Cache [13] or to join open developments such as Globule [14], Coral [15], CoDeeN [16] or SCDN [17], many of them with different structure and operation protocols. So the protocol described in this paper could be deployed to run over that implemented protocols.

This paper is structured as follows. Section 2 outlines the architecture and describes the system parameters. Our motivations and other works related with grouping nodes and establishing connections between them are presented in section 3. Section 4 explains the protocol and its operation. The analytical model is shown in section 5. Section 6 shows the desktop application developed, the testbed and the performance evaluation results. Finally, in section 7, the conclusions are summarized.

2. Outline of the system

The architecture ranges from some to all surrogates from the groups to interconnect them. When a surrogate searches content, first it tries to get it from its own group and, in case of failure, it would try to get it from other groups. Once a surrogate has the information, it will act as a cache for its group.

The proposed architecture is divided in two layers:

1. *Organization Layer*: Surrogates in this layer maintain and manage the architecture establishing connections between surrogates from the distribution layer. It is divided in two sub-layers for scalability reasons. The first sub-layer surrogates (L1-S) organize the distribution layer surrogates (DS) in zones and a set of surrogates from the group implements it. The second sub-layer surrogates (L2-S) allow the interconnection with other groups and one or few surrogates from all groups implement it.
2. *Distribution Layer*: Each group of surrogates has its own distribution layer, implmented by a set of its surrogates. Each DS has connections with DSs from other groups as a hub-and-spoke system. DSs are in charge of delivering data between groups.

Each group of surrogates has both layers. All layers are shown in figure 1.

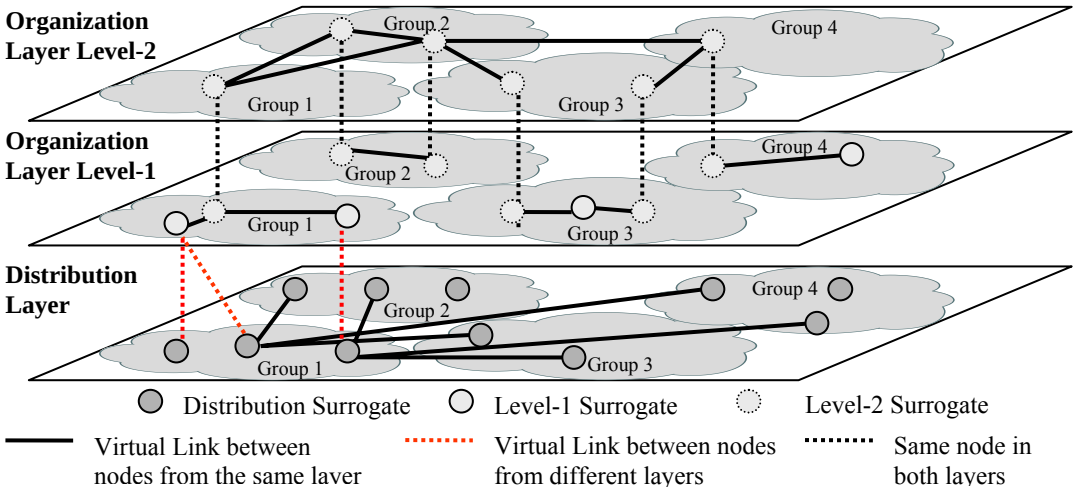

Figure 1. Architecture layers.

Each L2-S has L1-S and DS functionalities and each L1-S has DS functionalities. First surrogate, from a new group, that joins the architecture has all functionalities; next surrogates acquire functionalities as a function of some parameters described in a later section.

Security is an important issue in this architecture, so a new surrogate has to authenticate with a L1-S to become a DS. Afterwards, it belongs to that L1-S zone. DSs use L1-S connections to request connections with DSs from other groups. When a DS becomes a L1-S, it starts using organization layer functionalities while using distribution layer functionalities, but stops accepting new connections from DSs from other groups avoiding computation consumption due to DS functionalities.

L2-Ss must establish connections with L2-Ss from their group and from other groups, and L1-Ss must establish connections with L1-Ss or L2-Ss from their group only.

The routing algorithm to route information between L1-Ss and between L2-Ss is a two levels SPF (Short Path First) [18]. It allows fast searches to find DSs adjacencies, but it can be changed for other routing protocol depending on the CDN's characteristics. In [19] the authors provide some measurements from a two level SPF routing protocol, and they show how it could provide fast convergence. Experiments given in [20] show that a SPF database having 10^4 external updates from other nodes will consume 640 Kbytes of memory.

2.1. NodeID parameter

It is a 32-bits identifier. It is unique for each surrogate and shows its age in a certain group. L1-Ss, asking to its closest L2-S, assign nodeID sequentially to new DSs. A DS can't join the architecture without a *nodeID*, and in case of assignation failure when it joins the architecture, it has to ask again for a *nodeID*. For security reasons, L1-Ss just know their neighbors' network layer addresses, other surrogates' *nodeID*s and how to reach them. Requests and replies are always routed through L1-S's network. Ll-Ss route information inside the group using *nodeID* values. On the other hand, because it is based on identifiers, messages are not network layer dependent. It can be deployed over IPv6 or any other network layer protocol.

2.2. GroupID parameter

It is the group identifier. First time a surrogate joins the architecture, it has to choose its *groupID*, so it would belong to that group and it must authenticate with a L1-S having the same *groupID*. Then, the L1-S would assign a *nodeID* to it. *GroupID* could be assigned manually, using its global position or using the application layer protocol signature, but two groups can not have the same *groupID* in the network. If it is the first node in the group it would become the L2-S of that group, with *nodeID*=0x01, and it might authenticate with L2-Ss from other

groups. Neighbor L2-Ss would propagate this new entry to the L2-S's network. L2-Ss route information between groups using the *groupID*.

2.3. δ parameter

It is the surrogate suitability parameter, and depends on surrogate's bandwidth and in the time it has belonged to the architecture. It is used to know which surrogate is the best one to promote. The age is defined in a non-linear form because first surrogates are more important than the last ones, in terms of stability. Surrogates with higher bandwidth and older are preferred, so they would have higher δ. Every β DSs, surrogate with higher δ would become L1-S. Every α L1-Ss, surrogate with higher δ would become L2-S. α and β values depend on the number of surrogates in the group and their data traffic. α and β values for a P2P architecture partially decentralized networks are shown in [18]. Equation 1 defines δ.

$$\delta = (BW_{up} + BW_{down}) \cdot K_1 + (32 - age) \cdot K_2 \quad (1)$$

Where *age*=$\log_2$(*nodeID*), so age varies from 0 to 32. The age is defined in a logarithmic form instead of a linear form, because it gives higher granularity in low *nodeID* values. Figure 2 shows δ parameter values for some common bandwidth values as a function of surrogate's age (considering K_1=1 and K_2=128). Older surrogates stand higher roles than new ones unless they have low bandwidth. Surrogates with high bandwidth and relatively new ones could have higher δ values.

2.4. λ parameter

It represents the surrogate's capacity, and depends on surrogate's bandwidth (in Kbps), its number of available connections (*Available_Con*), its maximum number of connections (*Max_Con*) and its % of available load. It is used to determine the best surrogate to connect with.

In order to define surrogate's bandwidth weight in the calculation of λ, surrogates with total bandwidth (upstream plus downstream) equal or lower than 256 Kbps have the same weight. λ parameter is defined by equation 2.

$$\lambda = \frac{\text{int}\left[\frac{(BW_{up} + BW_{down})}{256} + 1\right] \cdot Available\_Con \cdot (100 - load) + K_3}{Max\_Con} \quad (2)$$

Where $0 \leq Available\_Con \leq Max\_Con$. *Load* varies from 0 to 100. A load of 100% indicates the surrogate is overloaded. K_3 gives λ values different from 0 in case of a load of 100% or *Available_Con*=0. We have considered $K_3=10^3$ to get λ into desired values. Figure 3 shows λ parameter values, when *Max_Con*=32, for some common bandwidth values as a function of its available number of connections with other surrogates. Surrogate's load is fixed to 75% in these scenarios.

2.5. Virtual-Link cost and metric

Virtual-Link cost is based on surrogate's capacity. The more the surrogate's capacity, the lowest its cost is. Equation 3 defines virtual-link cost.

$$C = \frac{K_4}{\lambda} \quad (3)$$

With $K_4=10^3$, we obtain values higher than 1 for λ values given in equation 2. Figure 4 shows virtual-link cost values, when *Max_Con*=32, for some common bandwidth values as a function of its available number of connections with other surrogates. Surrogates with *Available_Con*=0 and/or *Load*=100 will give *C*=32.

Metric is based on the number of hops to a given destination and the link cost of those surrogates involved in the path. The *metric* is used by the SPF algorithm [21] to obtain the best path to reach a surrogate. Equation 4 shows the metric to a destination *j*.

$$Metric\ (j) = \sum_{i=1}^{n} C_i \quad (4)$$

Where C_i is the i^{th} surrogate virtual-link cost and *n* is the number of hops to a destination j.

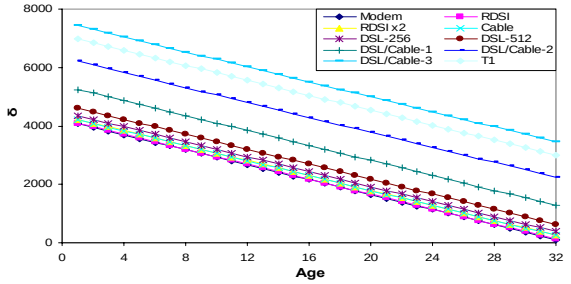

Figure 2. δ parameter values.

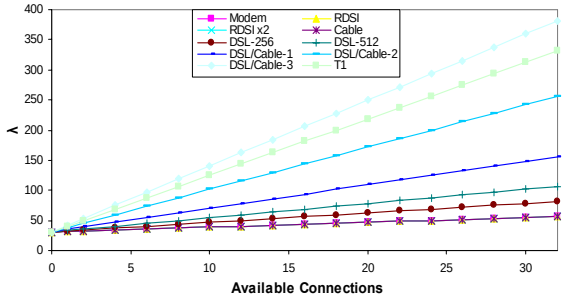

Figure 3. λ parameter values.

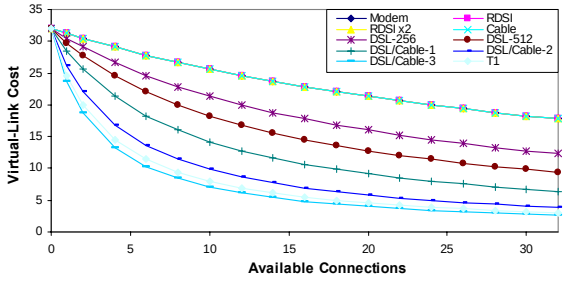

Figure 4. Virtual-link cost values.

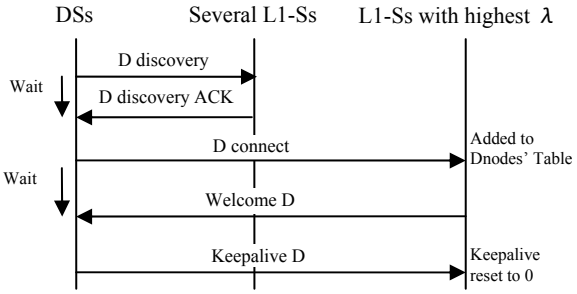

Figure 5. Messages when enters a new DS

Table 1 describes all the parameters described in this section.

3. Protocol description

When new surrogates join the architecture, they are defined as DSs. Firstly, a DS issues a discovery message with its *groupID* to L1-Ss known in advance or by bootstrapping [22]. Only L1-Ss with the same *groupID* would reply with their λ parameter. The new DS would wait for a hold time and choose the L1-S with higher λ. If there is no reply for a hold time, it would send a discovery message again. Next, DS sends a connection message to the elected L1-S. This L1-S will reply a welcome message with the assigned *nodeID* and information related with the backup L1-S (it will be used for recovery purposes). Then, it would add DS's entry to its DSs' table. Finally, DS would send keepalive messages periodically to the L1-S. These steps are explained in figure 5. If the L1-S does not receive a keepalive message from the DS for a deadline time, it would erase the entry from the database.

DSs send request messages to the L1-S that is connected, to establish connections with DSs from other groups. This message contains the destination *groupID*, and sender's *groupID*, *nodeID* and network layer address. Then, L1-S routes it to the L2-S in its group using the SPF algorithm and the L2-S forwards it to the L2-S network. When a L2-S receives this message from other group, it sends a message to every L1-Ss in its group in order to find the DS with highest λ. L1-Ss would reply with their two DSs with highest λ. After a certain period of time, L2-S chooses the two DSs with highest λ and sends them a message.

The highest one would be the preferred; the second one would act as a backup. This message contains the *nodeID* and the requesting DS's network layer address. When these DSs receive the message, they send a message to connect with any other group DS. Next, they send to the L2-S in its group a message indicating the establishment of a connection with the other group DS. If L2-S does not receive this message for a hold time, it will send a new message to the next DS with highest λ. This process would be repeated until L2-S receives both confirmations. When the requesting DS from other group receives these connection messages, it adds the DS with highest λ as its first neighbor and the second one as its backup. Finally it replies with an ack message.

Table 1

Summary of the parameters described

Description	Parameter
Surrogate Identifier	nodeID
Group of Surrogates Identifier	groupID
Surrogate suitable parameter	δ
Age of the surrogate in the system	age
Surrogate's capacity	λ
Number of available connections	Available_Con
Maximum number of connections	Max_Con
% of available load	Load
Virtual-link cost	C
Metric to a destination j	Metric (j)
Parameters used to know when it is needed a surrogate in the higher layer	α and β
Normalizing constants	K_1, K_2 and K_3

If the requesting DS does not receive any connection from other DS for a deadline time, it would send a requesting message again. Finally, both DSs would send keepalive messages periodically. If a DS does not receive a keepalive message from the other DS for a deadline time, it would erase this entry from its database. Afterwards, every time a DS needs to request for content from other group it has to look up its DS's distribution table. Figure 6 shows graphically the previous steps..

When a L2-S receives a new *groupID* in its *groupID* table, it would send a message to all L1-Ss in its group. Then, L1-Ss will forward this message to all DSs in their zone. Subsequently, DSs start the process to request DSs from the new group as it is explained in figure 6.

When a L2-S calculates that the L1-Ss' λ average in its group fulfills equation 5 (so, there are few L1-Ss in the group), it sends a message to all L1-Ss to request a new L1-S.

$$\lambda \le 1.05 \frac{k_3}{Max\_Con} \tag{5}$$

We have chosen $K_3=10^3$ to get λ into the desired range of values. When a L1-S receives the message, it would reply with the *nodeID* of the DS with highest δ in its zone. L2-S would process all replies and would select the highest δ DS. Then, it would send a message to the L1-S with highest δ in its group. This message would be routed to the chosen DS. After this moment,, the DS would become a L1-S and it would send a disconnection message to its corresponding L1-S. If the L2-S does not receive changes for a hold time, it would send a new request message to the second highest δ DS. If it fails again, it would start the process but avoiding those DSs. Previous steps are explained graphically in figure 7.

If a new L1-S has to establish a connection with any L1-S known in advance or by bootstrapping [22]. First, it sends a discovery message with its *groupID*. Only L1-Ss with the same *groupID* would reply with their λ. New L1-S will wait for a hold time and would choose L1-Ss with highest λ. If there is no reply, the new L1-S would send a discovery message again. Then, new L1-S would send a connection message to the chosen L1-Ss. They will reply with a welcome message indicating it is connected to the architecture and they will become neighbors. New L1-S would send them its neighbor list to update their L1-S network database and all of them will recalculate new routes using the SPF algorithm and the aforementioned metric. Next, they would send their database to the new L1-S to build its L1-S network database. Next iterations, it would only receive updates. New L1-S would send keepalive messages to its neighbors periodically. These steps are represented graphically in figure 8. If it does not receive a keepalive message from its neighbor for a hold time, it would erase this entry from its database.

When a L2-S checks there are too many L1-Ss in the group for few L2-Ss (see equation 6), it will send a message to the highest δ L1-S to generate a new L2-S.

$$\frac{number\_of\_L1-S}{Max\_Con \bullet number\_of\_L2-S} \ge 1 \tag{6}$$

Every time a L1-S receives that message, it will become a L2-S. Then, it sends a message to its neighbors to inform them about it. If the L2-S initiating this process does not receive changes in its L1-S's table for a hold time, it will send a new request message to the second highest δ L1-S. Figure 9 represents graphically the previous steps.

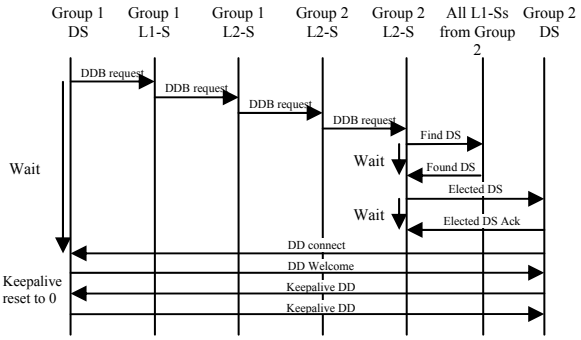

Figure 6. DS message to request connections.

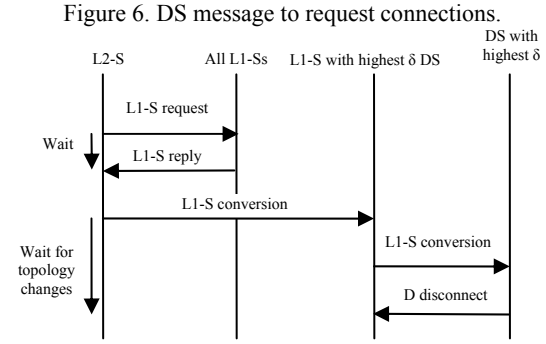

Figure 7. Messages to request a new L1-S.

When the L1-S becomes a L2-S, it sends a discovery message, with its networkID, to find L2-Ss from other networks. If there is not any reply in a certain period of time, it will begin the process again. Each L2-S from other networks replies to this message with their networkID and their λ parameter. It chooses L2-Ss with higher λ and sends them a connection message. Then, they reply with a welcome message indicating that a new node has joined the architecture. After that, it sends them its neighbor list. Its neighbors add this new entry to their topological database and recalculate routes using SPF algorithm. When they finish, they will send their database to the new L2-S to build its database. Next database messages would be only updates. Finally, it would send them keepalive messages periodically to indicate that it is still alive. If it does not receive a keepalive message from a neighbor for a dead time, it will erase this entry from its database. Figure 10 shows graphically the previous procedure. More information about the fault tolerance design could be found in [23]

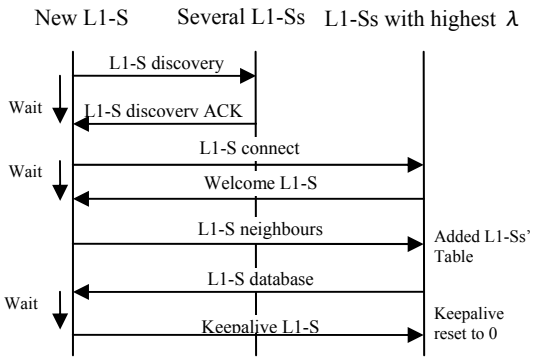

Figure 8. Messages when enters a new L1-S.

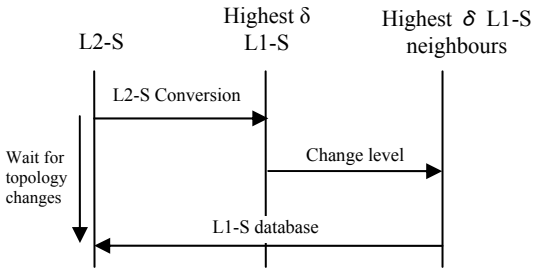

Figure 9. Messages when a L1-S becomes a L2-S

4. Analytical model

We use graph theory to define the network. Given $G = (V, \lambda, E)$ a network of nodes, where V is a set of DSs, λ is a set of capacities (λ_i is the i^{th}-DS capacity and $\lambda_i \neq 0$, $\forall$ i^{th}-DS) and E is a set of connections between DSs. Let k be a finite number of disjoint groups of V, $\forall$ $V = \cup (V_k)$. Given a DS v_{ki} (i^{th}-DS from the k group), there is not any connection between DSs from the same groups ($e_{ki\text{-}kj}=0, \forall\ v_{ki}, v_{kj} \in V_k$). Every v_{ki} DS has a connection with one v_{ri} from other group ($r \neq k$). Let $D=[M_{hj}]$ be the capacity matrix, where M_{hj} is the capacity of the j^{th}-DS from the h group. Let's suppose $n=|V|$ (number of DSs in whole network) and k the number of groups of V, then we will obtain equation 7.

$$n = \sum_{i=1}^{k} |V_k| \tag{7}$$

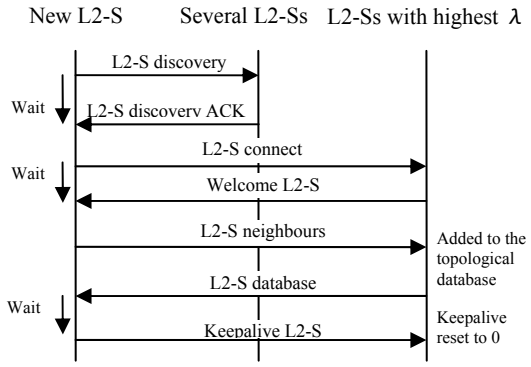

Figure 10. Messages when there is a new L2-S

On the other hand, the number of connections $m=|E|$ depends on the number of groups k, the number of DSs in each group k_m and the number connections that a DS has with DSs from other groups (we will suppose that a DS has connections with all other groups, so it will have *k-1* connections). Equation 8 gives the *m* value.

$$m = \frac{1}{2}\sum_{i=1}^{k}\sum_{j=1}^{k_m}\sum_{l=1}^{k-1} v_{km}(l) \qquad (8)$$

Where $v_{km}(l)$ is the l-connection from the m-DS from the *k* group. Assuming *r* logical groups with *d* DSs inside each one of them, the number of connections in the system is shown in equation 9.

$$m = \frac{d \cdot r(r-1)}{2} \qquad (9)$$

Figure 11 shows the number of connections in the system for scenarios with 50, 70 and 100 subsets of nodes. Keeping d values in the range between 0 and 10^3.

5. Performance evaluation

The performance evaluation of this proposal under real constraints, have been carried out by means the development of a desktop application using Java programming to run and test the proposed architecture and its protocol. Object-oriented programming allows us to have several modules, so we can change easily parts of the application to adapt

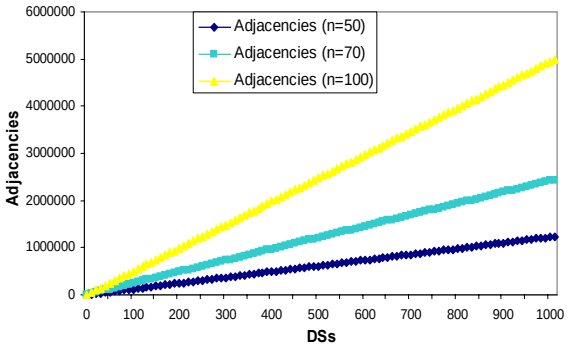

Figure 11. Number of connections between DSs.

it to different types of CDNs. It allows the node to run DS, L1-S and L2-S functionalities, to deploy the architecture properly. The application let us choose the group (or CDN) connected to and we can vary several parameters such as k_1, k_2, k_3, *Max_Con*, maximum % of CPU load used, upstream and downstream bandwidth, keepalive time, timers and so on. Its main window (figure 12) shows all parameters configured (such as *groupID* parameter) and values calculated and obtained from the network such as δ, λ and *nodeID* parameters.

5.1. Testbed and measurement system

The measurement of the operation of the architecture interconnection protocol has been performed over two different testbeds:

1. First testbed (used in scenario 1, 2 and 3). It was composed by 24 Intel ® Celeron computers (2 GHz, 256 MB RAM) with Windows 2000 Professional Operative System to know the protocol bandwidth consumption. They were connected to a Cisco Catalyst 2950T-24 Switch over 100BaseT links. One port was configured in a monitor mode (receives the same frames as all other ports) to be able to capture data using a sniffer application. This testbed is used for 3 scenarios:
   a) First scenario had only one group (see figure 13 a)). It had only one L2-S, which was also a L1-S, and there were 23 DSs in the group. We started taking measurements before we started the L2-S, 20 seconds later we started all DSs.

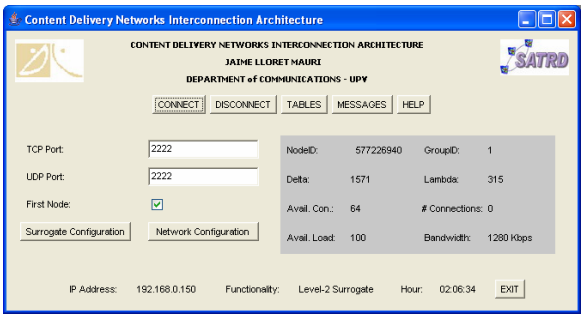

Figure 12. Desktop Application.

b) Second scenario had two groups (see figure 13 b)). Every group had one L2-S, which was also a L1-S. There were 11 DSs in each group. We started taking measurements before we started first L2-S. First we started the L2-S from first group, 20 seconds later we started the L2-S from the second group, 20 seconds later we started 11 DSs from the first group and 20 seconds later we started 11 DSs from the second group.

c) Third scenario had three groups (see figure 13 c)). Every group had one L2-S, which was also a L1-S. There were 7 DSs in each group. We started taking measurements before we started L2-Ss. First we started a L2-S from first group, 10 seconds later we started 7 DSs from the first group, then, 10 seconds later, we started a L2-S from the second group, 10 seconds later, we started 7 DSs from the second group, 10 seconds later, we started a L2-S from the third group, then, 10 seconds later, we started 7 DSs from the third group.

2. Second testbed (used in scenario 4). It was composed by 42 AMD Athlon™ XP 1700+ computers (1.47 GHz, 480 MB RAM) with Windows XP Professional Operative System. They were connected to several Cisco Catalyst 2950T-24 Switches over 100BaseT links. The implemented scenario has 3 groups interconnected (see figure 13 d)). All these networks have only one L2-S (which are also L1-Ss). First network has 12 DSs, second network has 13 DSs and the third network has 17 DSs. In order to take measurements from the scenario, we connected every group to a switch and all Switches were connected to a central switch as a star topology. One port of the central switch was configured in a monitor mode, like in testbed 1, to be able to capture data using a sniffer application. We started taking measurements before we started the L2-S from the first group, 10 seconds later we started the L2-S from the second group, 10 seconds later we started the L2-S from the third group, 10 seconds later we started sequentially all DSs from the first group, 10 seconds later, we started all DSs from the second group and finally, 10 seconds later, all DSs from the third group.

5.2. Measurement results

Figure 14 a) shows the bandwidth (bytes per seconds) consumed while running the protocol for the first scenario. There are several peaks because some times discovery messages and keepalive messages from DSs and L2-Ss are in addition (it occurs every 20 seconds). There are also DDB request messages to other groups, but there are no replies because there is only one network. The number of bytes per second when the network has converged is over 2,000, but there are peaks over 12,000 bytes every 60 seconds due to keepalive messages. Figure 14 b) shows the number of broadcasts per second for the first scenario. It shows that the most number of broadcasts are between 20 seconds and 40 seconds. It is because DSs are looking for adjacencies with DSs from other networks, in stationary mode there are less number of broadcasts.

Figure 15 a) shows the bandwidth consumed in the second scenario. First peak (around 20 seconds) shows when all 11 DSs from the first group established a connection with the L2-S of their group. Starting from 40 seconds, the number of bytes per second is incremented because of the number of DDB request messages sent from first group DSs to look for adjacencies with DSs from the second group. Moreover, the average number of bytes per second is higher than in the first scenario because there are more keepalive messages. The number of bytes per second when the network has converged is over 3,000 bytes per second. On the other hand, peaks reach 12,000 bytes per second. Figure 15 b) shows the number of broadcasts sent per second in the second scenario. Many broadcasts are sent between 20 and 80 seconds. Starting from 80 seconds they are sent less frequently.

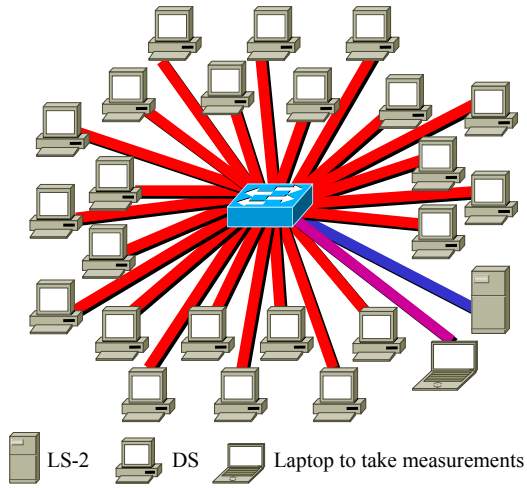

Figure 13 a) First scenario.

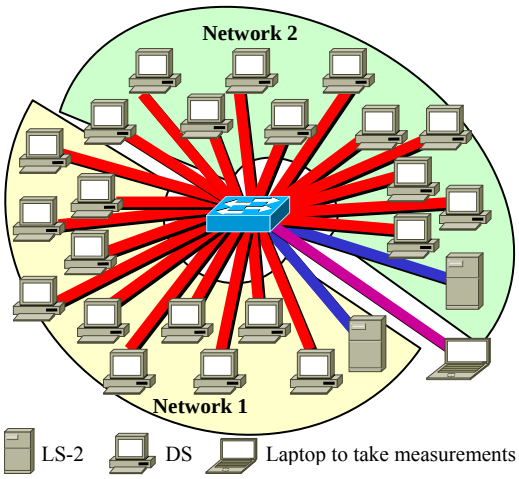

Figure 13 b) Second scenario.

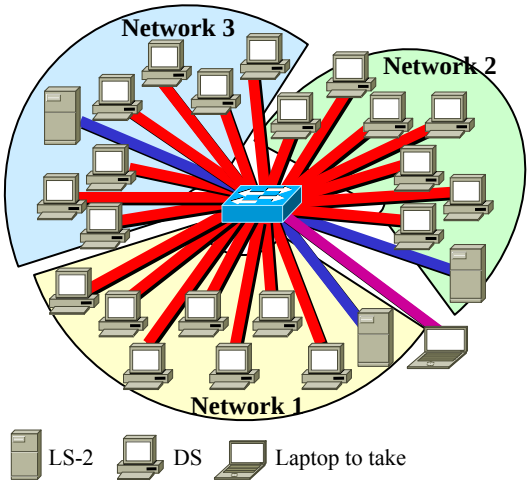

Figure 13 c) Third scenario.

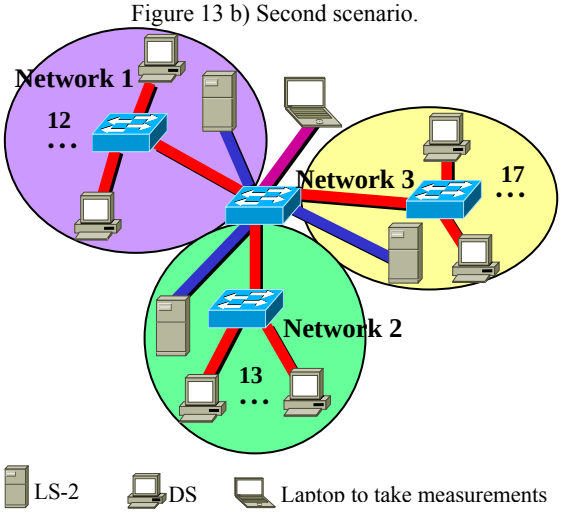

Figure 13 d) Fourth scenario.

Figure 16 a) shows bandwidth consumed in the third scenario. First peak (over 30 seconds) shows when first 7 DSs from the second group were started. Once the third group is started, the bandwidth consumed is higher than in the first and in the second scenario, but the numbers of bytes in the peaks are only a little bit higher. The number of bytes when the architecture has converged oscillates from 2,000 to 6,000 bytes per second. Figure 16 b) shows the number of broadcasts per second in the third scenario. Many broadcasts are sent between 20 seconds and 80 seconds as in the second scenario. Although higher peaks are between 60 and 120 seconds, the number of broadcasts is similar to the second scenario.

Figure 17 a) shows the bandwidth consumed in the scenario of the second testbed. The number of bytes per second when the network has converged oscillates from 4,000 to 8,000 bytes per second. Peaks because of keepalive messages are not so significant in this case. Figure 17 b) shows the number of broadcasts per second in the scenario of the second testbed. The highest peak appears aroud 70 seconds (when DSs from the third network were started).

Simulation measurements for more types of topologies could be found in [24].

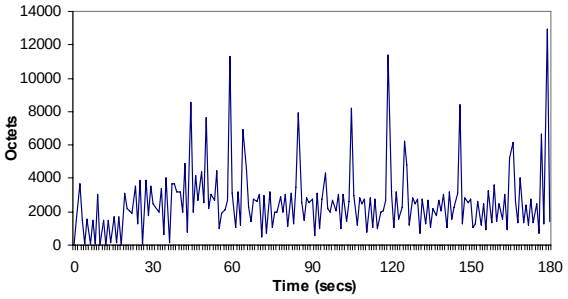

Figure 14 a). First scenario bandwidth utilization.

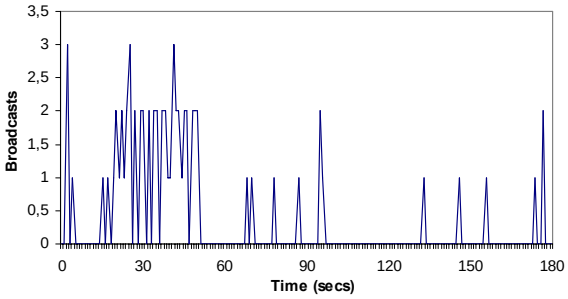

Figure 14 b). First scenario number of broadcasts.

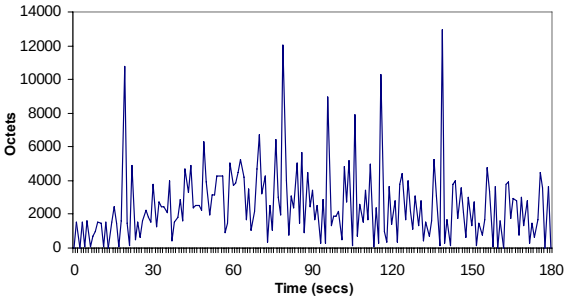

Figure 15 a). Second scenario bandwidth utilization.

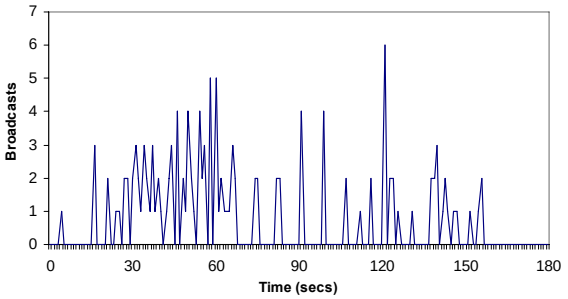

Figure 15 b). Second scenario number of broadcasts.

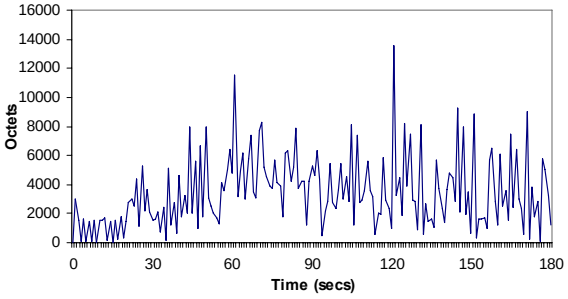

Figure 16 a). Third scenario bandwidth utilization.

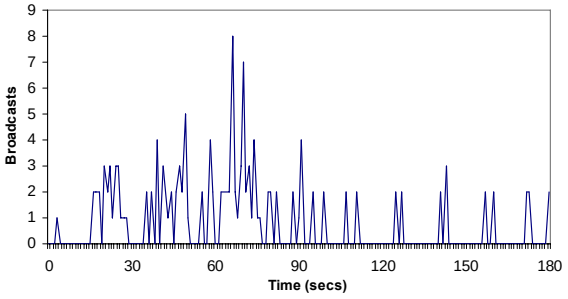

Figure 16 b). Third scenario number of broadcasts.

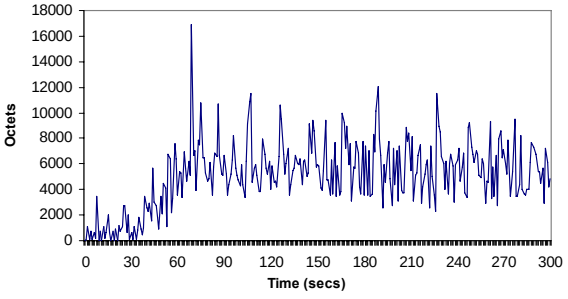

Figure 17 a). Fourth scenario bandwidth utilization.

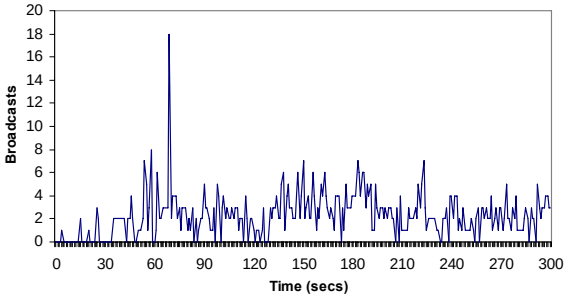

Figure 17 b). Fourth scenario number of broadcasts

6. Related works

There are several works in the literature where nodes are clustered into groups and connections are established between nodes from different groups, but all of them are developed to solve specific issues. To the extent of our knowledge, there is not any previous interconnection system to structure connections between groups of nodes like the one proposed in this work.

A. Wierzbicki et al. presented Rhubarb [25] in 2002. Rhubarb organizes nodes in a virtual network, allowing connections across firewalls/NAT, and efficient broadcasting. The nodes could be active, if they establish connections, or passive, if they don't. Rhubarb system has only one coordinator per group and coordinators could be grouped in groups in a hierarchy. The system uses a proxy coordinator, an active node outside the network, and Rhubarb nodes inside the network make a permanent TCP connection to the proxy coordinator, which is renewed if it is broken by the firewall or NAT. If a node from outside the network wishes to communicate with a node that is inside, it sends a connection request to the proxy coordinator, who forwards the request to the node inside the network. Rhubarb uses a three-level hierarchy of groups, may be sufficient to support a million nodes but when there are several millions of nodes in the network it could not be enough, so it experiences scalability problems. On the other hand, all nodes need to know the IP of the proxy coordinator nodes to establish connections with nodes from other virtual networks.

Z. Xiang et al. presented a Peer-to-Peer Based Multimedia Distribution Service [26] in 2004. The paper proposes a topology-aware overlay in which nearby hosts or peers self-organize into application groups. End hosts with in the same group have similar network conditions and can easily collaborate with each other to achieve QoS awareness. When a node in this architecture wants to communicate with a node from other group, the information is routed through several groups until it arrives to the destination. In our proposal, when a node wants to communicate with a particular node from other group, it sends the information to the adjacent DS from that group which will contact with the node of that group, so we will have fewer hops to reach a destination (L1-Ss and L2-Ss networks are used to organize connections between DSs only).

There are other architectures based on super-peer models such as Gnutella 2 [27] and FastTrack [28] [29] networks. Each super-peer in these networks creates a group of leaf nodes. Superpeers perform query processing on the behalf of their leaf nodes. A query for a leaf node is sent to a superpeer which floods the query to its superpeer neighbors up to a limited number of hops. The main drawback in this architecture is that all information has to be routed through the super-peer logical network. In opposition, as all DSs, in our proposal, are connected with one DS from other networks, it can send the information directly to the DS of that network, and that DS would be responsible of forwarding the information to its own network, avoiding overloading the higher layer network and allowing more scalability.

Finally, there are some hierarchical architectures in which nodes are structured hierarchically and parts of the tree are clustered into groups such as the one presented by Liu Hongjun et al. [30] and the one presented by B. Thallner et al. [31]. In some cases, nodes have connections with nodes from other groups although they are in different branches of the tree, but in all cases, the information has to be routed through the hierarchy to achieve nodes from other groups, so all hierarchy layers could be overloaded in case of having many data to be transferred. On the other hand, in case of many groups, the hierarchical structure could become unstructured due to many connections establishments between nodes from different groups placed on different layers of the hierarchy. Our proposal uses higher layers for organizational purposes, improving scalability and restricting only DSs layer to transfer data. The organization layer would remain structured while the architecture is running and the corresponding nodes are only focused on providing adjacencies for DSs.

7. Conclusions

Currently there are many CDNs deployed. The need to develop a system to interconnect them is growing. We have presented an architecture to join content delivery servers groups that establishes

connections between surrogates from different groups basing this decision on surrogate's capacity. It can be deployed not only as a new CDN architecture clustering surrogates into groups, but to solve the problem of joining CDNs. It is based on two layers. Organization layer has two sublayers. First one joins all groups and second one organizes surrogates in zones. Both sublayers help to establish connections between distribution layer surrogates. It is scalable because organization layer is based on the system described in [18]. Once the connections are established, content delivery could be done without using organization layer surrogates because they are used only for organizational purposes.

We have defined several parameters to know the best surrogate to promote to higher layers or to connect with. Real measurements have proved it is a feasible architecture because the bandwidth consumption to manage the system is low and it can be used in real environments. The difference between the number of bytes per second when there are two groups joined and when there are three groups joined is not significant, it depends more on the number of nodes than on the number of groups The relationship between the number of groups and the number of messages generated is not linear. Neither the number of bytes per second nor the number of broadcasts per second have duplicated although we have duplicated the number of DSs (from the third scenario to the fourth scenario). In future works keepalive time intervals, to have fast failure recoveries, will be studied.

8. Bibliography

[1] A. Vakali, G. Pallis, Content delivery networks: status and trends, IEEE Internet Computing 7 (6) (2003), pp. 68-74.

[2] L. Zou, E. Zegura, M.H. Ammar, The effect of peer selection and buffering strategies on the performance of peer-to-peer file sharing systems, in: Proceedings of Tenth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, MASCOTS, Fort Worth, TX, 2002.

[3] S.G.M. Koo, K. Kannan, C.S.G. Lee, A genetic-algorithm-based neighbor-selection strategy for hybrid peer-to-peer networks, in: Proceedings of the 13th IEEE International Conference on Computer Communications and Networks, ICCCN'04, Chicago, IL, October 2004, pp. 469–474.

[4] Simon G.M. Koo, Karthik Kannan and C.S. George Lee, On neighbor-selection strategy in hybrid peer-to-peer networks, Future Generation Computer Systems 22 (7) (2006), pp. 732-741.

[5] D.S. Bernstein, Z. Feng, B.N. Levine, S. Zilberstein, Adaptive peer selection, in: Proceedings of the 2nd International Workshop on Peer-to-Peer Systems, IPTPS03, Berkeley, CA, February 2003.

[6] J. Byers, J. Considine, M. Mitzenmacher, S. Rost, Informed content delivery across adaptive overlay networks, in: Proceedings of ACM SIGCOMM 2002, Pittsburgh, PA, August 2002.

[7] S.G.M. Koo, C.S.G. Lee, K. Kannan, A resource-trading mechanism for efficient distribution of large-volume contents on peer-to-peer networks, in: Proceedings of the 14th IEEE International Conference on Computer Communications and Networks, ICCCN'05, San Diego, CA, October 17–19, 2005, pp. 428–433.

[8] M. Day, B. Cain, G. Tomlinson, P. Rzewski, A Model for Content Internetworking, RFC 3466, 2003.

[9] CDI Working group, available from, http://www.ietf.org/html.charters/cdi-charter.html

[10] Akamai, available from: http://www.akamai.com.

[11] Speedera, available from: http://www.speedera.com.

[12] H. Hlavacs, M. Haddad, C. Lafouge, D. Kaplan and J. Ribeiro, The CODIS Content Delivery Network, Computer Networks 48 (1) (2005), pp 75-89.

[13] D. Verma, Content Distribution Networks, An Engineering Approach, Wiley, New York, 2002.

[14] G. Pierre, M. van Steen, Design and implementation of a user-centered content delivery network, in: Proceedings of the Third IEEE Workshop on Internet Applications, San Jose CA (USA), June 2003.

[15] Michael J. Freedman, Eric Freudenthal, and David Mazières, Democratizing Content Publication with Coral, in: Proceedings 1st USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '04), San Francisco, CA, March 2004.

[16] V.S. Pai, L. Wang, K.S. Park, R. Pang, L. Peterson, The dark side of the Web: an open proxy's view, ACM SIGCOMM Computer Communication Review, 2004.
[17] B. Molina Moreno, C.E. Palau Salvador, M. Esteve Domingo, I. Alonso Pena and, V. Ruiz Extremera, On content delivery network implementation, Computer Communications 29 (12) (2006), pp. 2396-2412.
[18] J. Lloret, F. Boronat, C. Palau, M. Esteve, Two Levels SPF-Based System to Interconnect Partially Decentralized P2P File Sharing Networks, in: Proceedings of the International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services, Papeete, Tahiti, October 2005.
[19] C. Alaettinoglu, V. Jacobson and H. Yu, "Towards Millisecond IGP Convergence", in: Proceedings of NANOG 20, October 2000.
[20] J. Moy, RFC 1245 - OSPF Protocol Analysis. July 1991. Available from: http://www.faqs.org/rfcs/rfc1245.html
[21] J. M. McQuillan, I. Richer & E. C. Rosen, The New Routing Algorithm for the ARPANET, IEEE Transactions on Communications 28 (1980), pp.711-719.
[22] C. Cramer, K. Kutzner, and T. Fuhrmann, Bootstrapping Locality-Aware P2P Networks, in: Proceedings of IEEE International Conference on Networks, Vol. 1. Pp. 357-361. 2004.
[23] Jaime Lloret, Juan R. Diaz, Fernando Boronat and Jose M. Jimenez, A Fault-Tolerant P2P-based Protocol for Logical Networks Interconnection, in: Proceedings of the International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services, Joint ICAS'06 and ICNS'06, Silicon Valley, USA, July 2006.
[24] Jaime Lloret, Juan R. Diaz, Jose M. Jimenez, Jesus Tomas, A P2P architecture to join supernodes from partially decentralized networks, in: Proceedings of the International Conference on Telecommunication Systems, Modeling and Analysis 2005, Dallas, Texas, November 2005.
[25] Adam Wierzbicki, Robert Strzelecki, Daniel Świerczewski, Mariusz Znojek, Rhubarb: a Tool for Developing Scalable and Secure Peer-to-Peer Applications, in: Second IEEE International Conference on Peer-to-Peer Computing (P2P2002), Linköping, Sweden, 2002.
[26] Z. Xiang, Q. Zhang, W. Zhu, Z. Zhang, and Y. Zhang, Peer-to-Peer Based Multimedia Distribution Service, IEEE Transactions on Multimedia 6 (2) (2004).
[27] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, S. Shenker, Making gnutella-like networks scalable, in: ACM SIGCOMM, 2003, pp. 407–418.
[28] Nathaniel Leibowitz, Matei Ripeanu, and Adam Wierzbicki. Deconstructing the Kazaa Network, in: Proceedings of the 3rd IEEE Workshop on Internet Applications (WIAPP'03), June 2003.
[29] Jian Liang, Rakesh Kumar and Keith W. Ross, The FastTrack overlay: A measurement study, Computer Networks 50 (6) (2006), pp. 842-858.
[30] L. Hongjun, L. Ping Luo and Z. Zhifeng, A structured hierarchical P2P model based on a rigorous binary tree code algorithm, Future Generation Computer Systems 23 (2) (2007), pp. 201-208.
[31] B. Thallner, H. Moser, Topology control for fault-tolerant communication in highly dynamic wireless networks, in: Proceedings of the Third International Workshop on Intelligent Solutions in Embedded Systems, May 2005.