P-CDN: Extending Access Control Capabilities of P2P Systems to provide CDN Services

Fatih Turkmen
University of Trento, Italy
Fatih.Turkmen@disi.unitn.it

Pietro Mazzoleni
IBM, USA
pietro@us.ibm.com

Bruno Crispo
University of Trento, Italy
Bruno.Crispo@disi.unitn.it

Elisa Bertino
CERIAS and CS Department, Purdue University, USA
bertino@cs.purdue.edu

Abstract

New important emerging business paradigms, such as "service virtualization" can be made easy and convenient by the use of P2P systems. In these paradigms, often the owners of the services are different (and independent) from the owners of the resources used to offer such services. In comparison to centralized servers, P2P systems can conveniently offer higher availability and more bandwidth as they harness the computing and network resour es of thousands of hosts in a de entralized fashion. Despite these useful features and their su ess in the resear h ommunity, P2P systems are still not very popular in the business world. The main reason for su h a skepti ism is their la k of proper se urity.

In this paper, we address this issue by motivating and explaining the benefits of adding access control in P2P systems to make them more suitable and flexible as a te hni al platform for providing third party servi es. We propose an ar hite ture augmented with a ess ontrol me hanisms to enable ontent delivery on a P2P system. The proposed ar hite ture is shaped a ording to CDN requirements.

We have also tested the feasibility of our approa h with a prototype implementation and the preliminary results show that our system an s ale well also in the presen e of very large number of poli ies.

1 Introduction

1.1 Background motivation

Internet and the Web have been an important multiplier for service providers in the area of information technologies (IT) and content distribution. Today, we see many companies, both small and large, providing a variety of services in a large number of domains, from e-learning and digital library to entertainment and health-care, many of which require handling large amounts of data. From one perspective, these are different forms of content delivery with variable interaction/communication contexts.

In the second half of the nineties, with the advent of "outsourcing", many companies practiced the idea of using resources, both logical and physical, owned and managed by other organizations. This enabled them to partially release the responsibility of resource maintenance and to have drops in the investment costs. More recently, because computer and network infrastructures are becoming a commodity, such an outsourcing model is further evolving into a new business model according to which the party providing the service may be different from the party providing the resources (e.g., memory, computing power, bandwidth, etc.). The main difference of this new model from the outsourcing is the evanescence of an agreement between the resource owner and the service provider. In this model, different providers combine their resources to deliver services to customers in a black-box style. Such a model is often referred to as "service virtualization". One of the major application domains of service virtualization is CDN (Content Delivery Networks) in which a content (especially large media) is delivered to the end users through a network of computers.

The emergence and wide adoption of P2P paradigm (more general than the conventional P2P file sharing) has made such model even more attractive, because each resource provider doesn't need to be a single entity any longer but rather a set of entities organized according to a P2P network. The benefits of using

P2P paradigm for CDN are well described in [12].

Such a shift from two-party to multi-party model implies the redistribution of authority especially when dealing with open P2P systems where users and resources can leave and join the system on a voluntary basis. Unlike the two-party model under which the authorization policy was often dictated by the service provider to the customers, the current multi-party models require several parties, often more than two, involved in the authorization administration. Moreover, the underlying relationship among the parties is peer-to-peer rather than hierarchical. Within this context, each party in the system should be able to express its own authorization constraints. To model such a freedom and flexibility, authorization models and systems have to support *multi-lateral poli ies*. The term multilateral policy emphasizes the fact that each party involved in the process has the right and the interest to specify its own security constraints and policy, and the underlying access control system needs to take all these constraints and policies into account when making access decisions.

1.2 Related Work

When discussing P2P systems, one of the most challenging and still open issues is security. In short, the system must guarantee the availability, integrity and authenticity of the content shared in the network as well as the privacy and authenticity of the subjects acting in the system. Security has a crucial role in making P2P technology available for enterprise and commercial applications.

In a sense, the current intellectual property and digital rights management issues of P2P systems can be cast as access control problems [3]. However, for a long time, authorization in P2P has not been considered a relevant area of research [5, 3]. This is because it is quite complex to enforce authorization in scenarios where the same physical subject can participate in the system under different virtual identities. In P2P, the peers can appear and vanish almost without control, there is no accepted key management infrastructure in place, and the copyright law is difficult to be globally enforced in all the countries reachable by a P2P network. Up to date, to understand whether and which P2P systems should enforce access control and to what extent, and how such enforcement should take place in the network, is still an open issue [5].

However, the initial skepticism of the need for an authorization model for P2P is now leaving the way to the acceptance that such model is needed. For instance, *LionShare* [1] is a project which proposes a P2P architecture aiming to facilitate legitimate file-sharing among individuals and educational institutions around the world. In LionShare, authorization plays a main role.

Authorization is not only a requirement for P2P networks built for academical use. Another application requiring authorization is for example *Lo kss* ("Lots of Copies Keep Stuff Safe") [2]. It is a P2P architecture providing librarians with an easy and inexpensive way of collecting, storing, preserving, and providing access to their local copy of the authorized content. In this system, authorization is needed both for the publishers, who give permission to the Lockss system to replicate their collections, and to the peers storing the content, to prevent unauthorized uses of the content they host.

More recent solutions [8] partially address access control by introducing two levels of authorization policies. Policies at the first level, referred to as *Lo al Poli ies*, are enforced by supernodes (or Local Security Managers - LSMs) on behalf of all users belonging to a given Virtual Community. Policies at the second level, referred to as *Closed Collaboration Teams Poli ies*, apply to all nodes belonging to communities which are dynamically created among nodes belonging to different LSMs. An even finer granularity of control has been proposed in [7]. Under such approach, content security is organized at node level. Each node can autonomously specify authorization constraints for each content it shares with other nodes. These solutions are useful in promoting P2P architecture beyond public file sharing. However, the underlying model of all these solutions is still very much hierarchical rather than multilateral.

1.3 Paper contribution and organization

The contributions of this paper are threefold. First, we identify the needs for integrating access control in a P2P infrastructure to support multi-lateral policies. Second, we show how to use such augmented P2P systems to provide a content delivery network (CDN) service. Third, we present the implementation and a partial evaluation of an initial prototype to show the feasibility of such CDN. The rest of the paper is organized as follows. Section 2 describes the Multi-lateral AC model we propose for P2P systems. Section 3 presents the architecture of our system. Section 4 discusses possible authentication infrastructures that can be deployed on the proposed architecture. Section 5 presents a prototype implementation. Section 6 gives an example of policy evaluation cost of our prototype system while Section 7 concludes the paper and outlines future work.

2 Multi-lateral AC Policies

2.1 Multi-lateral policies

Authorization policies, or *poli ies* for short, are logical expressions of constraints defined by a subject to control access (typically read or write operations) to certain physical or logical resources. Such a notion has been widely elaborated during the years and several policy languages (e.g.,[11]) and standards have been developed.

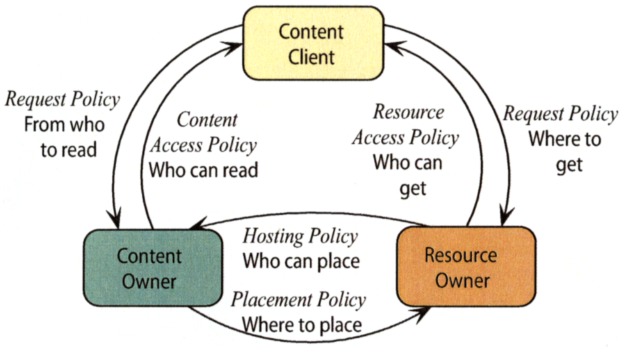

Figure 1. Multi-lateral subjects and policies

The first step in the development of a suitable multi-lateral access control system is the identification of various parties that are active in the system. In our context we primarily deal with *ontent distribution*, thus our model is based on three distinct parties: *Content Owner (CO)*, for subjects owning the content available in the network; *Resour e Owner (RO)*, for subjects making available storage space and processing resources; and *Content Client (CC)*, for subjects interested in accessing the content offered by COs.

According to this 3-party model, we can distinguish the following types of policies in our system:

- *Pla ement and ontent a ess poli ies.* A placement policy, specified by a CO, defines the restrictions concerning content replication across the peers. A CO might be willing to rely on third-party peers to store its content, as long as certain conditions are verified. The CO should thus be able to specify constraints the ROs must verify to host its content. As an example, a CO could state that its content can be replicated only into peers located within his country. Additionally, a CO may specify content access policies restricting accesses to its content by content client peers (i.e. some content can be accessed only by *gold* customers).

- *Hosting and resour e a ess poli ies.* A hosting policy is specified by an RO. It constrains the hosting of certain contents at a given peer. For instance, an RO can specify that only COs within the same company can replicate their content into its shared storage space. A resource access policy manages the use of physical resources of an RO. As an example, the RO can specify that its resources can be used for downloading or processing only between 8pm and 8am.

- *Request poli ies.* A request policy is specified by a CC and expresses the constraints of a CC with respect to the fulfilment of its requests, and in particular, the conditions that ROs and COs have to meet in order to be considered as candidates for a given request. For example, a CC can require the content to be downloaded only from ROs adopting a secure connection and from COs whose content is updated at least once a day.

Figure 1 illustrates the relations among the various parties in our model. Note that, in the figure the head of the arrow specifies the subject who expresses the policy whereas the tail is the subject to whom such policy is directed. Arrows represent relationships and they do not necessarily imply a direct interaction (e.g., CC doesn't directly communicate with CO but CO access policy plays an important role for CC's access decision).

3 System Architecture

Although our multi-lateral access control model could be applied to any P2P network architecture, in this paper, we restrict our design to unstructured P2P networks since they are widely used. Among the unstructured P2P networks, there are traditional completely unstructured networks such as the first version of Gnutella where there is no organization and any search is executed by broadcasting the query to all nodes in the network. This can result in a huge waste of network bandwidth. A variant of this model (i.e. FastTrack) is to introduce some hierarchy in which a special set of nodes is dynamically elected to manage the other nodes in the network. These nodes, usually called supernodes, store the directory information of the contents of the nodes they manage. In such systems, a query is then answered only by a subset of nodes and the amount of messages is reduced by two to three orders of magnitude. In this paper, we assume such a hierarchical unstructured P2P infrastructure. In the following subsections, we describe the system components, integration of access control mechanism and the system primitives our architecture needs.

3.1 System Entities

The main components of our system are CO, RO, CC nodes and SNs (Supernodes). A CO is the content owner providing content to the system according to its policy. Once logged in the P2P system, the CO can use it to place its content. ROs are the owners of the physical resources that constitute the P2P network insfrastructure while CCs are the ones making requests for the contents available in the system. Supernodes are special types of RO nodes because besides hosting as any other ROs they have high bandwidth to receive mutliple queries and enough storage and processing power to perform access control decisions for the nodes they manage.

3.2 System Primitives

Traditionally, a P2P infrastructure supports the following three basic primitives during the operation: *BootStrap()*, used when a peer wants to enter the network and the initial handshake occurs, *Sear h(ontent)*, by which a content is searched in the network, and *Get(ontent)*, used by CCs to download it from one or more ROs. We slightly modify the BootStrap primitive and name it *Registration()*, and introduce a new primitive *Pla e(Content)* by which COs upload their content into ROs. We now look in more detail what each of these primitives does.

- *Registration*: When a CC or a CO want to join the P2P system , a registration is performed contacting a supernode. This primitive includes the exchange of authentication credentials (i.e. identity certificates) and transfer of policies to the supernode. A default list of supernodes is hardcoded in the client software to allow users to know who to contact the first time. Another important operation performed by this primitive is the policy update. Whenever a CC/CO needs to update its policy, it sends, along with its authentication credentials, the policy version number and policy text to the supernode.

- *Sear h*: CCs submit a content request along with their security credentials to the supernode they are registered with. The supernode forwards the query along with the CC credentials and policy to other supernodes if it can't answer by itself. These supernodes in turn match the CC policy with the access control policies of the ROs and COs under their management and then builds the list of ROs which not only have the required content but also whose resources access policy authorizes the CC's request and whose resource satisfy the request policy of the CC. The name of the CC together with this list, all signed by the supernode, are then returned to the CC.

- *Get:* When a CC contacts the RO (i.e., identified by the result of a previous Search operation) for some content, the CC submits its request, the signed list returned by the supernode and its security credentials. The RO can either decide to trust the supernode's decisions or performs the control by itself.

- *Pla e:* For placement primitive, each CO forwards the placement request with resource requirements and the placement policy to its supernodes. Each of these supernodes checks the hosting policies of its ROs, in addition to checking the resource constraints. If the supernode can't find a match to meet the requirements and the security constraints, then it forwards them to the neighbouring supernodes. Finally, the supernode returns the list of ROs that authorize the CO's placement request if there is any. Among those, an additional evaluation has to be executed to verify that no conflicts exist between access policies defined by RO and CO. When a match is done between a CO and RO for a placement, the content is uploaded to the matched RO resources together with the content access policy of CO. Later, content access policy of CO is passed to the supernode of matched RO for evaluations.

In the remainder of the discussion, we focus on these four primitives and the inter-play with multi-lateral access control policies.

3.3 System Components

In our system, the architecture of CCs, COs and ROs, a part from possibly dedicating storage for the security policies, is not much different from that one of clients and servers of tradional P2P systems. Supernodes however, have a different architecture compared to traditional ones. As we can see from Figure 2, our supernodes consist of the following components:

- *Request Dispat her*: It is responsible to handle all incoming requests and dispatch them to the appropriate subcomponent. Request dispatcher

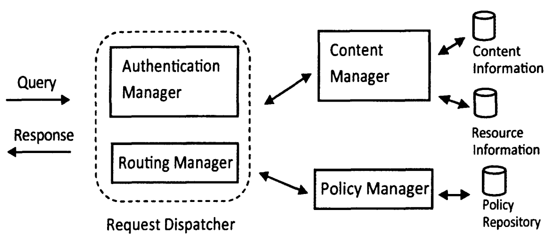

Figure 2. SuperNode Architecture

is composed of two subcomponents: Authentication Manager and Routing Manager. Authentication Manager handles all authentication issues occurring during communication. Routing manager subcomponent deals with internal cmmunication and with the communications between the supernodes and ROs to keep indexes of their content and policies and communicatons with neighbouring supernodes to forward requests when needed.

- *Content Manager* : it is responsible for returning the set of ROs, among the ones administered by the supernode, that have the requested content. If it can't find any match then it sends an empty list to the Request Dispatcher for further search in neighboring supernodes.

- *Poli y Manager.* Policy Manager determines which ROs, among the ones returned by the Content Manager, authorize the request after evaluation of COs, ROs and CCs policies involved in the query. It manages the policy repository supporting policy updates. The repository is composed of three directories for each type of policies: CO directory (placement and content access policies), CC directory (request policies) and RO directory (hosting and resource access policies).

4 Authentication

Our system must also address the problem of authentication. First of all client nodes and supernodes need to authenticate each other. If the client is not authenticated, then the supernode is exposed to DoS attacks by malicious clients that can flood the supernode with policy update requests. On the other hand, if the supernode is not authenticated, clients can be subject to man-in-the-middle attacks, where an impostor masquerading as a supernode can mislead clients.

To ensure authenticity and integrity, all the messages exchanged in our system are digitally signed. So for example, every policy and policy update sent from the CC to the supernode is signed. This signature is used by the supernode to verify that the policy really belongs to the claiming CC. In this way it protects itself against false accusations caused by incorrect policy enforcements of clients, since the supernode can prove that the policy enforced on behalf of the CC is actually the one sent and signed by the CC.

The supernode needs to sign the answers sent back to its clients, for non-repudiation purpose. A misbehaving supernode can evaluate the policies incorrectly or can lie about resource availability. Our system cannot prevent such misbehaviour; however, since supernode's statements are signed, a malicious supernode can be easily detected by the client and banned from the system. Another benefit in requiring each supernode to sign its statements is the possibility for clients to trust the supernode with respect to the policy evaluation process, thus off-loading such task to the supernode.

When there is flooding between supernodes, there must be an authentication between them to avoid abuses similar to the ones mentioned above.

Our system assumes the availability of a public key infrastructure (PKI) as the one described in [4] to provide authentication.

4.1 Enforcement

In our system, we assume some degree of trust between supernodes, CCs and COs. However since P2P systems are highly dynamic, an additional mechanism to audit if the policies of different parties are in fact implemented, can be sometimes useful. For performance reasons we have to accept methods that trade between performance and assurance, and implement only detection of policy violation rather than prevention. A CO for example, based on some configurable parameter (i.e. once a day), may want to double check the RO if it is implementing its policy correctly. It can do so by acting as a CC. It can submit a request to the supernode and then verify the results it gets from the supernode and from the selected RO. Additional detection mechanisms (e.g. state replication) suggested in [9] can be applied here as well.

5 Prototype Implementation

To show that our architecture can be deployed efficiently in practice, we implemented a prototype of our design features using an existing file sharing P2P system. We used GnucDNA[1], an open source P2P client core component for Gnutella and G2 networks, to develop our system components. It can run as both a supernode (ultrapeer in G2 terms) and a peer node according to its configuration.

[1]http://www.gnucleus.com/GnucDNA/

5.1 Policy manager service

We extended GnucDNA by implementing our policy manager as a web service.

The approach of running the policy manager as a web service has several advantages. First, it significantly eases the integration process. Because of the nature of open software, the components of our system are developed by different teams and are implemented in different programming languages. GnucDNA is implemented in C++ and the core of our policy manager, Sun Engine, is implemented in Java. While alternative integration methods are available, running the policy manager as a web service preserves the performance of all other components.

Another advantage in implementing the policy manager as a web service is that it can serve more than one supernode. When the policy manager runs in a trusted environment, policies from several supernodes can be handled by a single policy manager.

The policy manager service has been developed using Java and XACML [11] policy specification language.

To organize XACML policies into our structures, as well as to handle policy references (Policy Finder Modules) , we extended the XACML implementation developed by Sun [6]. SUNXACML 1.2 is an open source project providing complete support for all the mandatory features of XACML. As noted earlier, each node defines its own access control policies using XACML and these policies are stored in the policy manager service of the supernodes.

5.2 System Primitives Implementation

Our system primitives are converted into system operations in the following ways.

- **Registration:** The registration operation is done right after the handshake between the supernode and the peer node. The policy file/s are uploaded to the supernode according to the peer type. The content policy for a CO or the hosting and resource access policies for an RO or the request policy for a CC.

- **Data Placement:** When a peer has content to replicate, it sends a request that contains the CO profile, hosting policy and resource constraints to its supernode asking for a list of peers at which to replicate such content. The supernode first checks to see if it has any node that satisfies the resource constraints of the CO's request. If so, it serves locally the request. If none of its nodes satisfies this request, the supernode floods this query to other supernodes. The list of nodes that can host the content is returned to the requested CO. The CO selects one RO in the list and uploads its content.

- **Data Search:** The supernode maintains an index of data stored in its nodes and uses it to answer the queries. Once the list of nodes that host the requested content is identified, the list of such nodes is passed with the client profile to the policy manager service. The service returns the subset of nodes which authorize the client's request.

- **Data Download:** Once the search operation is completed, the client obtains a list of nodes which potentially host the content. The client contacts (one of) these nodes presenting its profile. The RO, checks the client profile once again to authorize the request (this check can be skipped if the node trusts the supernode), then the RO responds with data download.

6 Performance Evaluation

In this section, we provide some preliminary performance results of our access control enabled GnucDNA client. As noted earlier, there are many primitives that are usually supported by conventional P2P systems (such as get(data), search(data)). Our implementation not only affects the way these primitives operate but also adds new features. At the same time, it adds overhead to the system.

Our analysis summarized below revealed that the network overhead is not very much. The average size of an XACML policy composed of 4 resources, 1 subject and 4 rules is 15KB. This type of policy is quite representative of the policy normally used in many applications. Although it really depends on the settings used, the maximum number of leaf nodes connected to an ultrapeer in GnucDNA is usually around 100 [10]. In the worst case scenario, where all these nodes would be CC, CO and RO and they are all registering (so they all need to send their security policies) the full load is around 4500KB (3*100*15). As this is the worst case scenario in terms of network overhead caused by the attached access control mechanism, we can conclude that our system doesn't add very much load to the network.

In the next section, we provide some experimental results about policy evaluation.

6.1 Policy Evaluation

The main overhead of our solution is on supernodes since they now need to evaluate security policies. Thus estimating this overhead is crucial to analyze the feasibility of our system. The worst case of policy evaluation is when all peers satisfy the submitted CC content request. This means all the policies stored in the policy manager must be evaluated. To obtain a rough a idea about the cost of evaluation, we have conducted policy evaluation experiments. The experiments included one request arriving at a time and evaluated against all available policies. There are 10, 100, 1000 and 10000 policies in the experiments. Each policy has a random number of rules (1 to 6). We tried different sets of similarities among the policies. 20% means that 20% of the total number of policies are the same. 100% means that all the evaluated policies are the same.

We measured the response times of the policy manager service for each of these settings. The experiment testbed constitutes of a Windows Vista computer with PentiumIV 3.6 Ghz CPU and 2 GB of RAM.

Figure 3 shows the results of the first experiments. According to the results, evaluation of 100 policies takes around 1.2 second which is moderate. As the number of nodes connected to an ultrapeer is around 100 at maximum, this timing gives a rough idea of time cost of evaluation. It also shows that the existing XACML engines can be used for our system. However, if we consider the possibility of having much more policies, then the evaluation process is a bottleneck, thus an optimization is required.

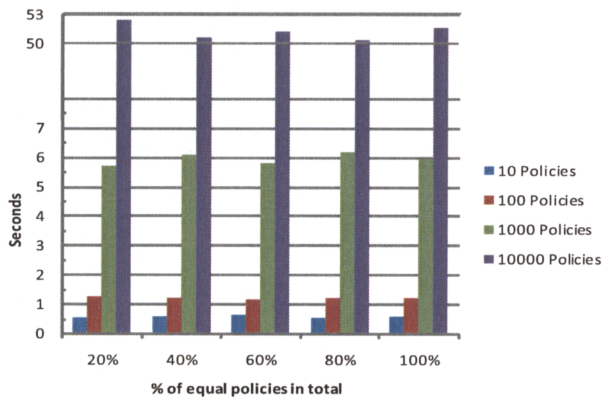

Figure 3. Policy Evaluation Times

7 Conclusions and Future Work

In this paper we introduced the concept of multilateral access control model in which each peer can specify its own policy to make sure that its security requirements are also considered every time a security sensitive action is performed in the system. There are a lot of applications where entities are organized as peers rather than in hierarchical manner and that would benefit from our model. As an example, we show how to apply this model to P-CDN, a CDN implemented by a P2P network. We prototyped the system and presented some preliminary performance results. These results has shown that policy evaluation can become a bottleneck as the number of policies grows to the order of thousands. Current and widely used implementation of policy engines such as Sun XACML do not scale well beyond hundreds of policies. As future work, we plan to optimize policy engines implementing new policy organization methods as the one presented in [8]. We also plan to remove the current limitation of the number of policy that a peer can support. Finally, we will test the new solution on a WAN by using a testing environment such as PlanetLab or Grid5000.

References

[1] Lionshare project [online]. *http://lionshare.its.psu.edu/main/.*

[2] Lockss program web page [online]. *http://lockss.stanford.edu/.*

[3] S. Androutsellis-Theotokis and D. Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Survey*, 36(4):335371, December 2004.

[4] K. Berket, A. Essiari, and A. Muratas. Pki-based security for peer-to-peer information sharing. *In 4th International Conference on Peer-to-Peer Computing (P2P 2004) Zurich Switzerland*, 2004.

[5] N. Daswani, H. Garcia-Molina, and B. Yang. Open problems in data-sharing peer-to-peer systems. In *ICDT '03: Proceedings of the 9th International Conference on Database Theory*, pages 1–15, London, UK, 2002. Springer-Verlag.

[6] S. X. implementation Web Page [Online]. http://sunxacml.sourceforge.net/.

[7] I. Jim Lowrey Endeavors Technology. Peer-to-peer. a security nightmare or a secure opportunity, 2002.

[8] P. Mazzoleni, B. Crispo, S. Sivasubramanian, and E. Bertino. Efficient integration of fine-grained access control in large-scale grid services. *IEEE International Conference on Service Computing (SCC)*, 2005.

[9] B. C. Popescu, B. Crispo, and A. S. Tanenbaum. Secure data replication over untrusted hosts. In *HotOS*, pages 121–126, 2003.

[10] A. Singla, C. Rohrs, and L. LLC. Ultrapeers: Another step towards gnutella scalability. *Gnutella RFC*, 2001.

[11] XAMCL and OASIS Security Services Technical Committee. eXtendible Access Control Markup Language (xacml) committee specification 2.0, Feb 2005.

[12] D. Xu, S. Kulkarni, C. Rosenberg, and H. Chai. A cdn-p2p hybrid architecture for cost-effective streaming media distribution. *ACM/Springer Multimedia Systems Journal*, 2006.