SECURELY DISTRIBUTING CENTRALIZED MULTIMEDIA CONTENT UTILIZING PEER-TO-PEER COOPERATION

Indrajit Ray and Tomas Hajek

Abstract: The ability of peer-to-peer networks to distribute multimedia content efficiently has been demonstrated many times by services such as Napster, Gnutella, and KaZaa. However, such services still suffer from two major disadvantages, viz., providing easy search facilities with little network overhead and providing secure services with the ability to audit the system. The latter feature is increasingly becoming critical as evidenced by the numerous law suites that have been brought against services such as Napster in recent months. In this work, we propose a new system that combines a centralized storage and distribution system with peer cooperation to help distribute multimedia content efficiently. Our system adds security, source verification, and an auditing ability to content distribution while reducing centralized bandwidth usage.

1. INTRODUCTION

With the increasing popularity of peer-to-peer networks like Napster, Gnutella, LimeWire, Scour Exchange, and iMesh that distribute multimedia content, it is evident that people want multimedia content over the Internet in a simple, user-friendly and efficient manner. However, recent lawsuits like those brought by the Recording Industry Association of America against services like Napster, over copyright infringements, often paint peer-to-peer networks as a bad technology. A major reason for such negative publicity is the inability of peer-to-peer networks to properly secure and monitor content distribution thereby (inadvertently) aiding to copyright violations and sharing of illegal media. We believe that this should not be a reason for abandoning peer-to-peer technology. Peer-to-peer networks if properly managed, can help reduce the load on servers distributing multimedia

content, reduce network congestion at such servers and even speed up the delivery process. In this work, we propose a hybrid technology for distributing multimedia content in an efficient but secure manner that also ensures the authenticity of the content. We combine the advantages of a centralized system to store and manage information with the advantages of a peer-to-peer network to distribute information.

Previous work on content distribution has focused two technologies [5]: (1) infrastructure-based content distribution and (2) peer-to-peer content distribution. Infrastructure-based content distribution networks are tied closely to the traditional client-server paradigm. In the client-server model, a machine or set of machines act as servers that store and distribute content upon client requests. The advantage of the client-server paradigm is that it is able to provide a specific performance guarantee. In addition, since centrally managed servers are responsible for storing, managing and distributing contents, these systems provide ample opportunity for proper auditing. The disadvantages of this mode of content distribution are that it suffers from problems such as bandwidth flooding, flash crowds and less fault tolerance. Peer-to-peer networks, on the other hand, rely solely on other clients to store and distribute content. Availability of content is rather unpredictable. Thus, no performance guarantees are provided by such systems. Moreover, since the storage, query and retrieval of information is done in a distributed manner in peer-to-peer networks, auditing the distribution of content becomes next to impossible.

More recently, work has begun to bring together an infrastructure-based system with a peer-to-peer system in order to alleviate the problems in each [5]. These include systems like CoopNet, BitTorrent [3] Chaincast [4], Allcast [2] and Abacast [1]. Although these systems help with flooding problems and content distribution, they do not address the problem of digital rights management; neither do they assure that content served from a client is truly the content that is requested. Contents distributed over these networks can be easily modified to either conceal copyrighted material or to make other material appear to be something that it is not. None of the available systems addresses these issues.

Our system addresses such problems. It utilizes both infrastructure-based and peer-to-peer distribution to circulate multimedia content, including on-demand media and live streaming media, in a secure and resource-efficient manner. Our approach is a hybrid one since it incorporates a centralized storage and management scheme with a peer-to-peer dissemination design. We aim our technology towards subscription services that provide access to on-line movies, music, live broadcasts, and/or any other form of multimedia content, for a fee. This is, by no means, a limitation of our system.

The rest of the paper is organized as follows. In section 2, we provide an overview of our system. Section 3 begins with a discussion of the security and content verification requirements in peer-to-peer content distribution. In section 3.1, we describe our approach to content verification. This is followed in section 3.2 with a description the secure content distribution method and in section 3.3 with an overview of how peers locate content in our protocol. Section 3.4 outlines the functionality requirements of the client for participation in our protocol. Finally section 4 concludes the paper.

2. SYSTEM ARCHITECTURE

We begin by providing an overview of the system architecture. We discuss some of the problems that arise in multimedia content distribution and how our design addresses these issues.

Figure 1. Architecture of Content Distribution System

The proposed content distribution system consists of three major subsystems – the *content-server and peers*, the *key-and-location server*, and the *clients*. The system is initialized with a single content server, which is the primary source for dissemination of multimedia content, and one key and location server. When a user needs access to the content hosted in this content server, the user downloads an application to his/her machine. The application spawns a client module on the machine to access the content

from the content server. The application also (optionally) starts a “peer” content server on the same machine that can help distribute the content that the client has retrieved. Figure 1 gives an overview of the system architecture.

We target our technology to a fee-for-service type subscription facility. We require that users register with a centralized server, the key-and-location server, in order to gain a username and password, and a certificate that attests to the fact that the user is a bonafide user of the services. The key and content server may or may not be co-located with the content server. Once this information is received the user can request content from the content server. The content on the server is encrypted and sent to the client along with a key to unlock the content. The content may be locally cached, buffered, or otherwise temporarily stored at the clients’ machines. We use this cached data for efficient distribution of content to other clients (the “peers” see Figure 2). A client requests content from the key and location server. If a Client B requests content that has previously been sent to another client A, the key and location server directs the Client B to attempt to obtain the content from Client A. We assume that Client A has the capacity to buffer a good portion of the content it has/is receiving or temporarily store it. We also assume that the client is willing to share the space and bandwidth to support such sharing. This may need some kind of an incentive from the initial server. Discussion regarding such incentive is beyond the scope of the paper. The central key and location server provides Client B with the key to decrypt the content from Client A as well as Client A’s address and a list of other clients that have recently requested the same content.

In the event that Client A is not available to service Client B, Client B either contacts other potential peers (from the list provided earlier) or notifies the central server that client A cannot fill the request and Client B will fall back on the central server to begin content retrieval. If Client B makes a request to the other peers in the address list provided by the central server, and if such a Client C responds favorably, then Client B retrieves the content from Client C and disconnect from the central server. If Client C also does not respond then Client B continues to receive content from the central server while continuing to request the content from any other peers in its address list for that content. If there are no other clients in the list then Client B simply continues to retrieve content from the server. At the same time, if Client B is willing and able to, it may also become a peer content server to other clients. Figure 2 depicts this interaction between the clients and the servers.

1. Client B logs in to request location of content and key from the key and location server. The latter responds with the key and a list of

- clients that recently requested the same content – for example, A and C.
2. Client B requests content from Client A and C. However, Client A fails because either it does not have the content any more or it is down. Client C has not responded yet.
 3. Client B falls back on Content Server and begins content retrieval.
 4. Client C replies positively to B's request. B disconnects from Content Server and gets remaining content from Client C.

Note that a fourth client D may also decide to retrieve the same content as Client B. At this time client B will figure in the address list of peers that the key-and-location server sends Client D, even though Client B is only midway through its retrieval process. In such a case we allow Client D to retrieve part of the content from Client B (acting as a peer content server) and the remainder from other peers or the content server.

Figure 2. Interaction of Clients and Servers

Our approach helps reduce the load at the main server. This is because, except for the very first client request, most others are fulfilled (at least partially) by “peers”. These peers can be in one of two states – either they are midway through the download process or have finished the process and disconnected. Either way, this allows a lesser number of connections at the main server. The only information that the key and location server needs to manage is the address list. Two problems need to be addressed here. The first is that A and C may have disconnected because of poor quality, a system crash, or simply due to the transient nature of peers. If A and C disconnected in a controlled manner, (i.e. not a system crash or network

failure), then the clients would have notified the central server that they were leaving and the server would have removed A and C from the list of available peers. The second is that A and C requested different content. In this latter case, it is likely that client A and C would not have been in the list provided to client B in the first place. Note however, that the address list does not have to be completely accurate since, as a last resort, the content is always available from the content server.

3. SECURITY AND CONTENT VERIFICATION

Our approach of using peer-to-peer network for content distribution has some serious security concerns. First, is the problem of ensuring the confidentiality of data while in transit. This involves both the confidentiality of the media that is being exchanged across the network as well as the confidentiality of other supporting materials (like user information, keys etc.) that need to be shared for the protocol to work. The second problem is ensuring that only legitimate users have access to this data. The third problem, which is perhaps the most critical, is ensuring that the content received is indeed the content that is requested. This problem can occur in one of several different ways. For example, it is rather trivial to label a virus or Trojan horse as a popular multimedia file such as an MP3 file, and place it for distribution to other peers under the many existing peer-to-peer file-sharing systems. Consider systems such as Napster, Gnutella, and KaZaa; they allow simple text searches based either on the file name or the media tag sometimes provided in the header of a multi-media file. An example format of such a tag is shown in Table 1. The ID3 tag is embedded in MP3 files but nearly all MP3 encoding software is capable of modifying this information.

Table 1. ID3 tag information format

Data	Length in bytes
Tag	3
Title	30
Artist	30
Album	30
Year	4
Comment	30
Genre	1

Some individuals that share content often mislabel their shared files either by changing the file name or the embedded tag as mentioned earlier. This leads to people believing they are downloading a particular piece of content when it is in fact something completely different. This may be

partially because of people trying to avoid prosecution for sharing copyrighted material or simply a way to distribute their own content by giving it a popular name. This could possibly lead to someone looking, for example, for Bach Symphony No. 5, finding it on a peer system under a file name search (i.e. bach_symphony_no5.mp3) take the time to retrieve it, only to find out it is a rap song containing profanity or a religious organizations talking on the evils of music and the internet.

Another way that the content can be modified without the end-user realizing it until it is too late is to edit the downloaded content and then re-distributing it. For example, consider a video media. The user can open the media in some video editor and use it to re-arrange scenes or substitute different clips for the original ones. Such video editors are easily available and do not require much expertise. Similar media editors are also available for audio files and other types of multimedia. Note that for such an attack to be launched it is required that we are able to store the media file in a local space and then re-distributing it – not just temporarily storing it in the local cache. In addition, the media should be readable via non-proprietary media players. The former is rather difficult to achieve for streaming media like the one we are targeting. The second on the other hand is easy to achieve.

The problem of verifying the authenticity of the media is compounded by the requirement that it may be necessary to be able to verify the media while it is being downloaded from the server or a peer. Thus, a solution that is based on creating a message digest over the complete media is unacceptable. It requires that the entire media be first received before any verification process is initiated. The verification problem is further aggravated by the fact that for streaming media, delivery occasional packet errors and losses are still acceptable but re-transmission is definitely not. This is because re-transmission can lead to unacceptable delays. What this means is that if during transmission of the media, some packets get corrupted or even lost, the network should not try to recover from it via the usual practice of re-transmission.

In our content distribution design, we solve these problems as follows. We utilize a novel scheme involving erasure codes based on Rabin's information dispersal algorithm (IDA) [6], hash functions and digital signatures to ensure authenticity of content. For confidentiality, we use a secure communication channel between the sender and the recipient. In the following, we elaborate on each of these techniques employed.

3.1 Content Verification

To enable content verification as well as allow playback of media before having received the entire stream, we require the content provider to

fragment the media into smaller fragments, compute message digests for each fragment and disperse the message digest using IDA [6], and compute digital signatures over a group of packets and distribute the same using IDA. The following steps are performed over each media that is to be distributed. Also the steps are performed only once at the main content server and we assume that all computations are done in $\text{GF}(2^8)$.

1. The content server breaks up the media into a number of fixed size *packets*. A set of n consecutive packets is grouped together for streaming and forms a block. Thus a *block* of packets is nothing other than a concatenation of n consecutive packets of the original media. The same operation is performed on each block. All blocks are of equal size. For the discussion we will consider operations on the first block.
2. For each packet P_i in the block of n packets, the content server computes a *packet hash*, $H(P_i)$ using a secure hash function like SHA or MD5.
3. The packet hashes in a block j are concatenated to form a *hash block* $F^j = H(P_1) \| H(P_2) \dots \| H(P_n)$ (where $\|$ is the concatenation operator) such that F^j has N 8-bit byte. Let b_i be the i^{th} character in F^j .
4. Two copies of F^j are made. A hash function is computed on one copy. This hash is called the *group hash* for the block of packets that is being considered now. Note that this *group hash* has the property that if any packet in the block is modified the *group hash* will change. The *group hash* for the j^{th} is denoted by $H_G(G_j)$. This group hash is digitally signed by the content server using its private key C_{prv} to generate the *signed group hash* $[H_G(G_j), C_{prv}]$.
5. The other copy of F^j is divided into N/m blocks such that each block is of length m as follows: $F^j = (b_1, \dots, b_m), \dots, (b_{N-m+1}, \dots, b_N)$. That is, $F^j = S_1 \| S_2 \| \dots \| S_{N/m}$ where $S_i = (b_{(i-1)m+1}, \dots, b_{im})$, $1 \leq i \leq N/m$.
6. Choose n vectors $\vec{a}_i \in_m (a_{i1}, \dots, a_{im})$, $1 \leq i \leq n$, such that every subset of m different vectors are linearly independent as specified in [6]. Using the set of vectors $\vec{a}_i = (a_{i1}, \dots, a_{im})$, $1 \leq i \leq n$, F^j is processed and divided n pieces $F_1^j, \dots, F_n^j$ as follows: $F_i^j = (\vec{a}_i \cdot S_1, \dots, \vec{a}_i \cdot S_{N/m})$, $i = 1 \dots n$, where $\vec{a}_i \cdot S_k = a_{i1} \cdot b_{(k-1)m+1} + \dots + a_{im} \cdot b_{km}$.
7. The digitally signed *group hash*, $[H_G(G_j), C_{prv}]$ generated in step 4 above, is also divided into n portions using the same set of vectors $\vec{a}_i$ and in the same manner.
8. Each signed group hash segment and hash segment that is created following the above steps, is appended at the end of the corresponding packet to form an authenticated packet. A group of n authenticated packets form an authenticated group and is streamed at the same time.

It can be shown quite easily that if at least m authenticated packets in a group G_j are received, then the receiver can reconstruct the *hash block* F^j as well as the *signed group hash* $[H_G(G_j), C_{prv}]$. For lack of space we omit the details here. The interested reader is referred to [6]. Thus the authenticated group has the property that even if $n-m$ packets in the group are lost or corrupted, the remaining packets can still be authenticated and the signature of the content server can still be verified. However, if more than $n-m$ such packets are modified (by malicious intent, say) it can be easily detected. The parameter n will be chosen depending on the bandwidth of the connection over which the media is being streamed, the maximum transmission unit allowed and the buffering capabilities at the receiver. The parameter m will typically be chosen based on an estimation of the error rate of the communication channel subject to the restriction $m \leq n$.

Adding the signature has the drawback of adding overhead in each message sent as well as additional computation to verify each fragment. However, we believe the benefit of being able to verify the origin of content and maintain security outweighs the added overhead.

3.2 Secure Content Distribution

One of our goals is to minimize the possibility that the transmitted content is accessible to any party that does not have valid rights to the content. Additionally, we want to make it possible for clients to pass along fragments that they currently have to other clients who have the rights to view the content. Further, we need to make the encryption quick enough to compute and easy to share between clients. In our system, we propose to use a secret key to encrypt the multimedia content in storage at the content server. Each authenticated block of the content is encrypted by a secret key (the same key encrypts all authenticated blocks of the entire media). The encrypted media is transmitted to a client. If the media gets temporarily stored at a client, it is stored in exactly the same encrypted manner as in the content server. The key-and-location server makes the decryption key available to each client that successfully authenticates against the server. We assume that all clients have exactly the same privileges to access any media. In addition, we assume that if a client needs to find out which other clients have access to certain contents, the first client has to be a registered user. Thus, it is reasonable to assume that there is no other profit consideration for a client to re-distribute a downloaded content to other clients than what is already accorded by the service provider.

In order for a client to authenticate itself to the key-and-location server, the client needs to previously register a username (login) and a password with the server. This is done with a secure registration website using SSL

technology. Once the user is registered, they can simply login to the secure site (during the registration session or later via an SSL encrypted session) and begin to request content. Our system keeps track of what content is downloaded by which user; it also maintains a database to track usage statistics and to allow users to view content (they have already purchased or have rights to view) from other computers, possibly without having to purchase the content again.

Once the user requests the content, the server sends the client, the key to decrypt the requested content as well as a list of other clients that the current client is able to receive the content from. The SSL connection with the key-and-location server is now terminated. When the client is ready to accept media it connects to either the content server or to a peer and obtains the encrypted content. We assume that the client software is secure (tamper proof) and that the user cannot use the client software or otherwise obtain any information from within the client software (for example the decryption key to decrypt the content for later use) without going through the authentication process through the central server.

Note that since a client that has successfully authenticated and retrieved some content, can decrypt the content and play it, there are means in which clients can pirate material. It is a fairly simple matter for a client to simply play back media and record it with another device, defeating the encryption. As an example, many computers can output their display to devices such as VCR's or other computers. This results in the possibility of someone simply playing the content on one machine and recording it on another.

It is even simpler with music files since most sound cards have the ability to play back one file and record whatever is being played to another file. This option is sometimes listed under recording options as record "What You Hear" or record "stereo mix". We acknowledge this problem but believe it to be a nontrivial problem and we currently deem it outside the scope of this paper. This problem may be a good topic for future work.

Figure 6 gives the steps involved in a client obtaining the multimedia content starting with the client registration process.

3.3 Locating Content and Distributing It

We want to reduce the chance of a server being overloaded with requests from clients while maintaining jitter free playback. We refer to the server being overloaded by client requests as flooding. Flooding degrades the performance of the server and subsequently all clients in a client-server model. It is also possible that a server will subsequently crash if it has spawned too many processes trying to deal with the incoming requests.

Figure 3. Content Retrieval Process

To offload some of the content distribution from the server we use the clients to distribute content to other clients. The main idea behind this approach is that clients that recently received a multimedia stream or fragments of multimedia content, will, with considerably high probability, retain the content in a cache; thus can pass a portion of that content along to other clients. This would help to distribute the content while removing some of the bandwidth and retrieval costs from the server.

We create a content distribution tree. The content distribution server is at the root of the tree and the key and location server is a disjoint entity, which communicates with all clients, as depicted in Figure 7. The content distribution server will communicate with the key and location server to coordinate the addition of new content, removal of content, and to ensure

that the content server has content encrypted with the correct keys. The key and location server helps create and maintain the content distribution tree.

To request content, a client logs in at a secure site and identifies the content. The key and location server in turn provides the client the following items: (1) the key to decrypt the content, (2) the number of fragments that the content is split into and the total length of the content, (3) the address of the centralized content server and (4) a list of clients that may potentially possess the data. This list includes, for each client, the client addresses, time at which its request began to be fulfilled and an estimate of the number of clients it can handle.

Figure 4. Content Distribution Tree

After the client receives the above information, it sends a request for the content to up to five (this number may be adjusted) of the available client-servers. The client waits for the response from the client-servers, chooses the one that responds first or times out. If the client times out or there are no peers in the list, then the client will request the information from the central content distribution server.

The client then sends a contact list message to the key-and-location server with information on the peers it attempted to contact and failed to receive content from. The key-and-location server reduces the estimate for the number of clients they can handle by one for all the clients on the contact list. When the number is reduced to 0, the client is removed from the list.

The location server next provides an updated list to the client. The client, in turn, attempts to request the next set of content fragments from another set of peers. This is done while the client is getting fragments from the central content server. If the client gets a response from a peer, then it requests that

the peer continue sending content fragments to it and disconnects from the content server. The client then sends a message to the location server with the information on which client it is getting content from, if it is willing to distribute content, and how many connections it may be able to handle.

The location server updates its location information to reduce the amount of clients that the responding peer can handle. If the new client is willing and able to participate in content distribution, then the location server will add an entry into its list and record the time the client began receiving content and the number of connections the client estimates it can handle.

In order to maintain the list of available clients the location server reduces the estimate of the number of clients a peer can handle. The reduction is made each time a client sends a contact list back with failures to respond. If the client received corrupt or malformed content from the peer, this is also noted in the contact list and the location server will remove these clients from its list. Clients that are disconnecting from the service notify the location server, and are removed from the list.

If a client is currently receiving content from a peer that is leaving, the leaving peer notifies the client. The receiving peer then requests the content from the central content server, requests an updated list from the location server and attempts to contact other peers in a similar fashion as described earlier.

3.4 The Client

In our system we use client cooperation and as such we require that the client to perform a number of essential tasks. The most important is to deliver content to the end user. We must do this in a timely manner with jitter free playback while utilizing as much of the free client resources as available to serve other clients.

The previous sections have dealt with how the client will locate content and receive it from the central content server or from other clients. There are a few points we still need to address. The first major issue is how we determine how many client connections a client-server can support.

To do this we first do the following simple calculation. During client setup we ask the user to specify what type of a connection they have available and then during each request we can determine how much of the connection bandwidth we need. Using our connection requirements as a baseline for the requirements of other clients we divide our total link capability by our link requirements; this way we have a reasonable estimate of the maximum number of connections the client can handle.

This process no doubt assumes that the users will not be using their machine's resources for anything else, which is unreasonable. Therefore,

during client setup we must ask the user if they wish to cooperate with other users to help distribute content. If the user is willing then we prompt for the maximum percentage of resources the users would be willing to allow the distribution system to use. The users may change this option at any time.

Some users may wish to be very stingy with their resources or may not know what kind of a connection they have. For this reason, we have designed our system to allow for transience of clients and to have a backup mechanism if client-servers cannot handle the estimated connections.

Another issue that we address is the storage of content on users' machines. Again, we need to prompt the user for an amount of disk space to allow for temporary storage. This may not be much of a problem since the cost of disk storage has dropped significantly. It is not uncommon to see machines with tens of gigabytes of storage capacity. Note, however, that this is temporary storage. When a client requests content, it is placed into temporary storage in its encrypted and fragmented form on the users' machine. The client can then decrypt the fragments and stream them to the media player of the users' choice, given that it supports multimedia streaming.

The client will delete the temporary files upon closing or after a pre-determined period. If the client crashed or the machine was turned off suddenly, then the client checks its temporary storage directory for content and deletes it when started again

4. CONCLUSIONS AND FUTURE WORK

In this work, we presented a design for a secure multimedia content distribution system using both centralized and distributed resources. We have covered the issue of authenticating the source of the content, using signatures, and added mechanisms for securing content in transmission using encryption. Our design contains mechanisms for dealing with transient clients and for providing content to users.

The benefits that our system provides include reducing some of the distribution load from the server onto other clients and adding content verification and security to content. We create a distribution tree and utilize our location server to track the location where other clients may be able to find the requested content. Once a location is determined, clients can request information from other clients or fallback on the centralized content server.

Our system makes multimedia content distribution more cost effective by reducing server bandwidth. We also believe that the security and auditing

possibilities of our system helps reduce the amount of copyright infringements that are associated with many other distribution systems.

We still need to test performance and determine if modifications to our design need to be made. We expect that the system will perform better when there are a large number of clients requesting a relatively small number of content. This is because there would be more overlap in content on a greater number of clients.

We need to evaluate which encryption algorithms are most efficient for encrypting multimedia content. We also need to determine if a 1500-byte fragment (including headers) is the best size for splitting up the content. We would like to address the possibility of using watermarks or tamper resistant hardware devices to further increase the security of our system. This may help in addressing how to eliminate, or in the least reduce, the ability of individuals to violate copyrights by capturing content with other devices and then redistributing the content.

References

- [1] C. Shi and B. Bhargava, "A Fast MPEG Video Encryption Algorithm", *Proceedings of the 6th ACM International Conference on Multimedia*, Bristol, UK, September 12-16, 1998.
- [2] F. Bao, "Multimedia Content Protection by Cryptography and Watermarking in Tamper-resistant Hardware", *Proceedings of the 8th ACM International Conference on Multimedia*, Los Angeles, CA, October 30 - November 3, 2000.
- [3] H. Kosch, "MPEG-7 and Multimedia Database Systems", *ACM SIGMOD Record*, vol. 31, June, 2002.
- [4] K. Andreev, B. M. Maggs, A. Meyerson, and R. K. Sitaraman, "Designing Overlay Multicast Networks for Streaming", *Proceedings of the 15th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, San Diego, CA., June 7-9, 2003.
- [5] M. Doernhoefer, "Surfing the Net for Software Engineering Notes", *ACM SIGSOFT Software Engineering Notes*, vol. 26, Jul, 2001.
- [6] M. Rabin, "Efficient Dispersal of Information for Security, Load Balancing and Fault Tolerance", *Journal of the ACM*, vol. 36, April, 1989.
- [7] N. Menon and P.W. Wong, "Protecting Digital Media Content", *Communications of the ACM*, vol. 41, July, 1998.
- [8] S. Rollins and K. C. Almeroth, "Pixie: A Jukebox Architecture to Support Efficient Peer Content Exchange", *Proceedings of the 10th ACM International Conference on Multimedia*, Juan Les Pins, France, December 1-6, 2002.
- [9] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai, "Distributing Streaming Media Content Using Cooperative Networking", *Proceedings of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, Miami Beach, FL., May 12-14, 2002
- [10] Y. Chawathe, S. Ratnaswamy, L. Breslau, N. Lanham, and S. Shenker, "Making Gnutella-like P2P Systems Scalable", *Proceeding of the 2003 Annual ACM SIGCOMM Conference*, Karlsruhe, Germany, August 25-29, 2003.