# Digital Content Information Repository For Future Media Streaming

Kazuhiro Mishima, Hitoshi Asaeda
Keio University, Graduate School of Media and Governance
Fujisawa, Kanagawa 252-0882, Japan
Email: {three, asaeda}@sfc.wide.ad.jp

***Abstract*—Digital content delivery environment has been significantly changed due to advancements of computer and network resources. Since users who want to receive media content must first obtain the content information, a scalable system to provide the available content information to prospective users is required for the future media network. In this paper, we propose a content information repository system using distributed data storage. The proposed system not only provides scalability but manages a huge amount of content information. In this paper, we describe the design and implementation of the proposed content information repository system. The performance evaluation of the actual implementation is also provided on the PlanetLab testbed.**

## I. Introduction

In the recent Internet, users can start transmitting media streaming contents anytime, anywhere. When a user transmits a media streaming content to Internet users, the sender needs to announce the content information to the potential receivers, and the potential receivers must obtain the information to receive the content. Content information is mainly composed of a "media locater", which is a pair of source and group addresses for IP multicast or a URL indicating a source of the media content, content type, and application. Proprietary information for each media streaming application can be indicated as well.

The Session Description Protocol (SDP) [1] is the standard protocol that describes media streaming content information. SDP has been widely used with the Session Announcement Protocol (SAP) [2] and the Session Initiation Protocol (SIP) [3], and VLC and RealPlayer [4] applications use SDP/SAP for content announcement. However, since SAP is a soft-state protocol, all active content information must be periodically transmitted to all potential users, in order to notify the content state is active. Moreover, it is not sufficient to simply use these protocols to announce any kinds of media locators in the future Internet, because heterogeneous Internet users would want to announce a huge number of and variety of streaming contents with different conditions and different ways.

One may conclude that the current search engine is a possible approach to retrieve content information. However, the crawling method takes time for gathering available content information in the entire Internet. It is necessary to implement a large scale content information repository system that provides all up-to-date content information to users for the future media deployment.

In this paper, we propose a content information repository system using distributed data storage. The proposed system manages a huge amount of content information for future needs as well as provides scalability. In this paper, we describe the design and implementation of the proposed content information repository system. The performance evaluation of the actual implementation is also provided on the PlanetLab testbed.

## II. Requirements

We summarize the requirements to design the content information repository system based on the analysis in [5].

*REQ1: Scalability* The content information repository system must be able to deal with a huge amount of content information. And heterogeneous users access to the repository by their own ways. Therefore, "Scalability" is the most important factor.

*REQ2: Register/Retrieve Latency* From end-user's perspective, minimizing "Latency" to register or retrieve content information is also the requirement for the content information repository, because of the dynamic nature of content information announcement scheme; for instance, it is worse that users retrieve obsolete content information as it takes time to retrieve or refresh the available content information from the repository.

*REQ3: Multiple Attributes* The information, which is managed in the content information repository system has multiple attributes, such as description or location of the content. Therefore, the system must deal with "Multiple Attributes".

*REQ4: Scope Awareness* The content information repository system should support to reflect data sender's policy ("Scope Awareness"), because the senders often want to define the content information distribution area for their contents.

*REQ5: Access Control* Unlike globally available content information announced to entire Internet users, the content information repository system should have the capability of the "Access Control" that limits access for secret or private content information only to legitimate users.

*REQ6: Data Protection* Obviously, content information should be protected from illegal modification or removal. It can be done by defining content ownership.

## III. Content Information Repository

We design a content information repository system based on the requirements aforementioned. The content repository

978-1-4244-9920-5/11/$26.00 ©2011 IEEE 

system consists of multiple nodes widely distributed over the Internet. The multiple nodes form a global-scale distributed storage that provides high scalability and low latency for frequent access. Just after a content sender registers his content information into the repository, potential content receivers can retrieve the information from the repository.

Regarding the storage implementation, there are two possible ways, 1) Clustering Relational Database (RDB) or 2) Distributed Hash Table (DHT), but both have pros. and cons. Based on the above requirements, according to [6], [7], [8], RDB has issues in "REQ1: Scalability" and "REQ2: Register/Retrieve Latency" when it uses in the largely distributed networks, while DHT has issues in "REQ3: Multiple Attributes", and "REQ5: Access Control", because of the characteristics of its algorithm. We therefore decide that the proposed repository system adopts the combination of RDB and DHT to compensate the shortcomings for different purposes.

"REQ4: Scope Awareness" is also fulfilled by this storage combination. Users in general want to categorize their contents as global and local contents; global contents are widely announced and distributed to all Internet users without receiver or network classification, while local contents limit access only for users in the same organization or network. This requirement is reasonable for the scope configuration in IP multicast. In the proposed repository, "global storage", which stores global content information, is implemented by DHT, and "local storage", which stores local content information, is implemented by RDB.

Regarding the "REQ6: Data Protection" requirement for DHT, as well as an access control method provided with the information label (described in Section III-E), our DHT-based global storage implementation provides a data protection function using Threshold Cryptography [9] as described in Section III-F. Since RDB in general fulfills REQ6, local content information stored in each node's RDB is protected from anyone outside of the site-local network (by rejecting the access).

### A. Component Overview

The content information repository system consists of multiple nodes. Fig. 1 illustrates the functional components of each node.

*1) Interface Layer:* The "Interface Layer" is a component that receives a query request from an end-user and returns the results (i.e., content information) to him. This layer cooperates with a web-based interfaces; an end-user accesses to the repository system through the "Interface Layer" of one of the repository nodes. Upon receiving a query request, this layer forwards the query to the "Access Layer" using SOAP protocol [10].

*2) Access Layer:* The "Access Layer" is a component that receives a query request forwarded by the "Interface Layer". It analyzes the user's query and controls underlying modules, "Query Processor" and "Query Collector".

*Query Processor:* The "Query Processor" analyzes the user's query and requests the Routing Layer to run the necessary process (detailed in Section III-B).

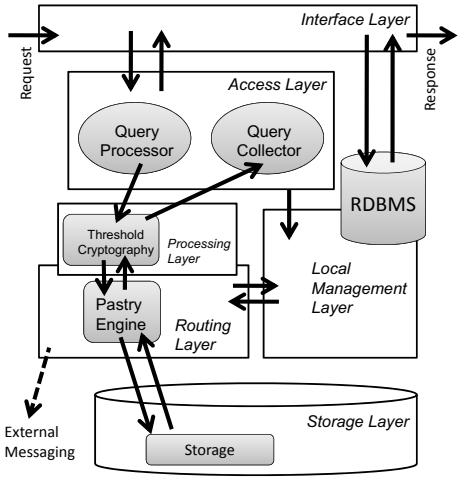


Fig. 1. Components of a repository node

*Query Collector:* The "Query Collector" collects the results performed by the "Query Processor" and returns the results to the "Access Layer". If multiple processes are performed in the "Routing Layer", this module will perform the join process according to the user's query.

*3) Routing Layer:* The "Routing Layer" provides the routing algorithm to collect the target data from the global storage. To ensure that the search function is flexible, the routing engine must be with some distributed hash table routing function and dynamically switch the function by the Access Layer's request. In this paper, Pastry [11] is used as the routing algorithm due to the following two criteria, 1) Optimal retrieval performance of the routing algorithm (Pastry has search performance of $O(logN)$ for $N$ nodes), and 2) Simplicity of prototype implementation. Thanks to the Pastry algorithm, "Scalability" and "Register/Retrieve Latency" requirements are addressed.

The distributed hash table (e.g., Pastry) consists of "Key" and "Value". "External Messaging" is the message that is communicated with a foreign node by the Pastry routing protocol. This inter-node messaging flow is shown in Fig. 1. This message is carried with plain-text data over TCP. This message includes the "Key" and "Value". The search function only supports exact-match lookups. This violates the "Multiple Attribute" requirement. We then propose the function to split information into multiple attributes. This implementation and basic operation are shown in Sec. III-B, while we already formulate another retrieving method as our future work (in Sec. VI).

*4) Processing Layer:* The "Processing Layer" implements the function that cannot be achieved only with the routing engine. This layer is the upper layer of the "Routing Layer". This layer provides "Access Control" and "Data Protection" functions for the basic algorithm of the Pastry protocol. The operation of these functions is shown in Sec. III-E and Sec. III-F.

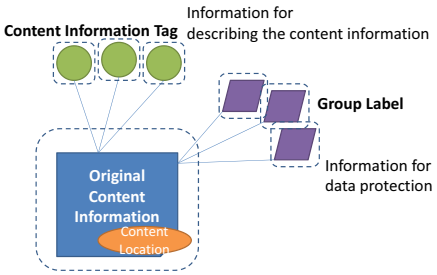


Fig. 2. Data Structure of Content Information.

*5) Local Management Layer:* The "Local Management Layer" is the component that manages the registered data from its own nodes. This layer manages information for updating and deleting to prevent any unexpected data modification. The data on the "Local Management Layer" stores into local storage.

*6) Storage Layer:* The "Storage Layer" is the component that stores all of the registered data. This layer manages the storage data by Time-to-live (TTL) to prevent unnecessary data from being stored in the storage.

*7) Local Management Layer:* The "Local Management Layer" is the component that manages the data being registered by a site-local node. The RDB function that stores the local content information is also included in this layer. Original content information registered in the site-local node is stored into the RDB, in order to modify or delete the content information.

*8) Storage Layer:* The "Storage Layer" is the component that stores the global content information registered by a site-local or other nodes. This layer manages the storage data by Time-to-live (TTL) as detailed in Sect. III-C.

### B. Basic Operation

*1) Registered Data Example:* To explain the operational flow, the following quoted data is used for example. This example is a modified format of Session Description Protocol (SDP) [1]. Attribute 's' is used to represent the title of the content, 'i' is used to describe the content, 'c' is for connection target, and 't' is for start and end timing.

s:SDP Seminar
i:A Seminar on the session description protocol
c:IN IP4 224.2.17.12/127
t=2873397496 2873404696

The data structure we use is shown in Fig. 2. First of all, the repository system registers the "Original Content Information" that contains the entire content information. It includes the location of the content (called "Content Location"). After the registration, the repository system splits information into each attribute (called "Content Information Tag") from the original information. Each split attribute is registered as individual information. The pointer for the original information is included in the attribute. In addition, to restrict the data access, "Group Label" is added to each attribute when needed, as described in Sect. III-E.

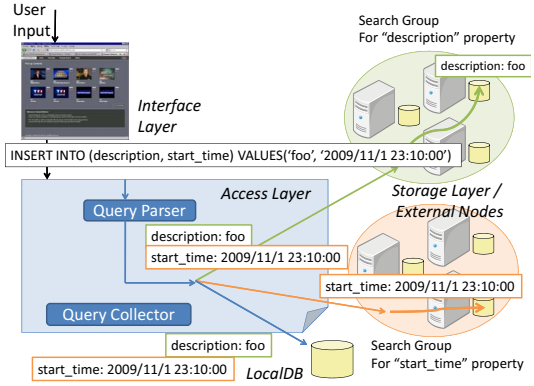


Fig. 3. Example operation of Data Registration.

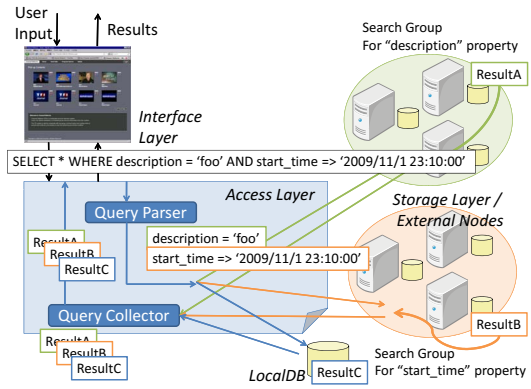


Fig. 4. Example operation of Data Retrieve.

*2) Data Registration/Retrieve Operation:* The data registration operation is shown below. Fig. 3 shows the example of the data registration operation. Fig. 4 shows the example of the data retrieval operation.

1) "Interface/Access Layer" accepts the end-user's query.
2) Register or retrieve local content information into/from the local storage. Register or retrieve global content information into/from the global storage.
3) (Global Information) "Query Processor" divides end-user's query into sub-queries by the attribute.
4) (Global Information) "Query Processor" inquires Pastry Engine for character-based sub-query.
5) (Global Information) "Pastry Engine" registers or retrieves all query data.
6) (Global Information) "Query Collector" collects the results of the registration/retrieval.
7) (Registration only) All registered information is registered into the "Local Management Layer" for the data deletion or management.
8) "Interface/Access Layer" returns the results to end-user.

Fig. 5 shows the split sub-queries based on example data.

*3) Data Deletion Operation:* The data deletion operation is described below.

1) "Interface/Access Layer" accepts the end-user's query.
2) Retrieve the target data from the "Local Management

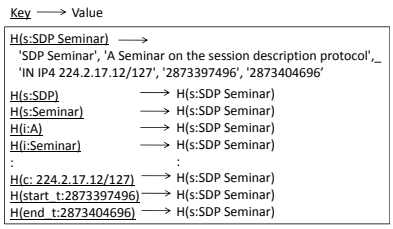


Fig. 5. Split data

Layer".

3) (Global Information) Delete the target data through the "Pastry Engine".
4) Delete all target data from the "Local Management Layer".
5) "Interface/Access Layer" returns the results to the end-user.

### C. TTL Data Management

In order to prevent useless data from being kept in global storage, our system has a TTL (Time-to-Live)-based Data Management function for the global content information. When data arrives, the TTL is automatically reduced from the storage. TTL can be changed by the configuration, and extended by the special process. The TTL Management Function is implemented in the "Storage Layer". The default TTL is 86400 seconds (1day).

There are two methods to extend the TTL, 1) Re-register the data: Register the data in the same search key and data, 2) Search the data: the search is performed with the same data. To prevent the data deletion by TTL, the "Local Management Layer" automatically performs the re-registering function.

### D. Improvement on Data Retrieval Performance

To reduce the data retrieval cost, the performance improvement process is implemented in the "Access Layer" and the "Storage Layer".

*1) Dynamic Indexing:* In the "Access Layer", once the search is performed, pointer information to the original data is created according to the user's request and the retrieval result. This process is called "Dynamic Indexing". Whenever the retrieval is done, the Dynamic Indexing process is executed. The performance of the user who will retrieve later improves because the retrieval history is accumulated one by one.

The operation flow is shown in Fig. 6. *UserA* is a user who retrieves the data previously, and *UserB* is a user who retrieves the data later. *UserB* can retrieve the data from any nodes.

1) The "Interface/Access Layer" accepts the end-user's query.
2) Retrieve the target data based on the query.
3) A pointer information to the original data is created according to the user's request and the retrieval result.
4) New pointer information is stored into the "Storage Layer".
5) The "Interface/Access Layer" returns the results to the end-user.

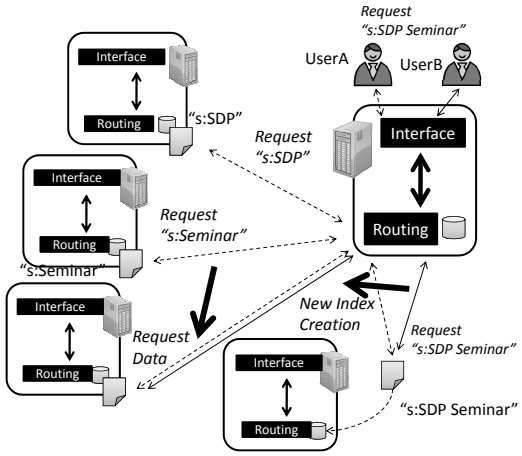


Fig. 6. The data operation of Dynamic Indexing

Using this operation, when the combination of the search keywords that have already been done will be requested later by a user (e.g. *UserB*), the pointer information that links to the original data can be retrieved. As the result, it is possible to reduce the query count, the time to retrieve, and the data traffic in the retrieval processing.

*2) Local Caching:* The "Storage Layer" has the local caching function to improve performance. The Local Caching is a technique to copy the original data into the local node's storage when the data is retrieved.

The data copied by the local caching will be stored into the local node. During data retrieval, if the corresponding data exists in the local node, the "Storage Layer" selects the data with higher priority. Thanks to this function, when the retrieval for the same data is performed on this node, the data acquisition time is fast, and the traffic is also reduced as the retrieval process is concluded only in the local node.

### E. Access Control using Group Label

To fulfill the "Access Control" requirement, for the global storage, we propose the access control method [12] using the "Group Label". The "Group Label" is a key to be created by SHA-1 hashing. A user who does not know the Group Label cannot retrieve the information from the global storage. Using this method, the repository system includes the function to exchange information only among users who know the Group Label. As shown in Equ. 1, the key for registering the access-controlled data is generated with the Group Label $label$.

### F. Data Protection using Threshold Cryptography

To meet the "Data Protection" requirement, the repository system adopts the data protection function using Threshold Cryptography [9].

Threshold Cryptography is a secret sharing algorithm. It splits an original data $S$ into $n$ pieces of the split data (shares), and restores the original data $S$ by collecting $t(t \leq n)$ pieces of the shares. In this algorithm, the original data can be restored if $t$ or more pieces of the shares are collected, but cannot be restored if $t-1$ or less pieces of the shares are

collected. In other words, to restore the original data $S$, the algorithm only needs to collect $t$ pieces of the shares, and works even if $n - t$ pieces of the shares are lost.

Threshold Cryptography has three advantages for our system, 1) Encryption strength: A strong encryption can be achieved by simple computation compared with a general information encryption algorithm, 2) Strength in data loss: even if there are some troubles in the repository nodes, the original data can be restored by the shares which is required to decode, and 3) Strength in data corruption: even if some of shares ($\leq t - 1$ shares) are corrupted, the original data can be restored by $t$ shares. This gives better robustness than the common replication mechanism.

As shown in Equ. 1, the key used for registering the split data is generated with the original key $org\_key$ and the counter value $counter$.

$$Key = H(org\_key \mid\mid label \mid\mid counter) \quad (1)$$

## IV. Evaluation

### A. Prototype Implementation

In order to evaluate our architecture, we implemented the content repository system shown in Sec. III-A as the prototype. In our prototype application, we implemented "WebUI Module", "Agent Module", and "Storage Module". "WebUI Module" is the application to implement the "Interface Layer" that can enable the access from end-users. "Agent Module" contains "Access Layer", "Local Management Layer", and "Processing Layer". "Storage Module" contains "Routing Layer" and "Storage Layer". Each components communicate using SOAP. Users can register, and retrieve content information by accessing WebUI (included in "WebUI Module") of either of node. Our system can be individually installed on each user's site. In this case, the global data storage is attempted to share among another user's site. In result, global content information is shared into all sites, and local content information is limited to share only on the each user's site.

We used Perl language for developing "WebUI Module" and "Agent Module", and also used SOAP::Lite for inter-component communication and SQL::Statement for query analysis. We used C language for developing "Storage Module", and also used libcsoap for inter-component communication. Java is widely used for implementing the distributed data storage. However, there is a memory overhead issue [13] by Garbage Collection in Java. In this situation, we dared to implement the data storage application using C language.

### B. Prototype Evaluation

To evaluate the performance of content information repository, we installed the "Storage Module" into 80 nodes (including US32, EU25, JP14, Asia9) over the PlanetLab testbed [14] and analyzed the behavior of the system. All nodes of "Storage Module" connect to the parent node which is installed at a PlanetLab node at our laboratory (Japan). In addition, other modules are installed on this parent node. PlanetLab is useful for the evaluation, because: 1) performance measurement can be carried out on a number of geographically distributed nodes over the real Internet, and 2) the actual implementation and its behavior can be justified during the experiment. Note that the network and the node conditions on the PlanetLab were often changed or sometimes unstable. We observed outlier during the measurement and omitted to specify it in this paper.

TABLE I
Processing Time in Each Components. (sec.)

| Component | Register | Retrieve |
|---|---|---|
| Int./Acc. Layer | 0.08 | 0.17 |
| Rtg./Str. Layer | 1.09 | 0.90 |
| Sum. | 1.17 | 1.07 |

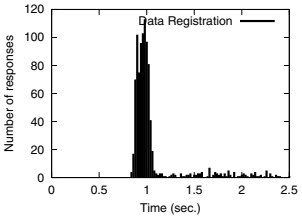


Fig. 7. Processing Time of Data Registration.

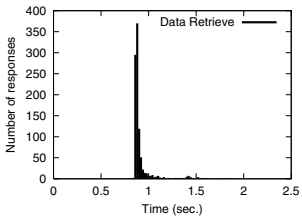


Fig. 8. Processing Time of Data Retrieve.

In our system, the data which register, retrieve or modify is written the SDP(Session Description Protocol [1])-based message. Example of data shows in below.

> <Key> Content ID (150bits SHA-1 Hash)
> <Value> s:'SDP Seminar', i:'A Seminar on the session description protocol', c:'IN IP4 224.2.17.12/127', start_t:'2873397496', end_t:'2873404696'

The processing time required to register or retrieve data was measured. In this evaluation, we measured 1) estimate time to register the data, and 2) estimate time to retrieve the data. This evaluation made each 1000 times, and evaluated the result. The data storage has already registered 10000 content information. Tab. I shows the result of measurements in each components.

Fig. 7 shows the performance of registration. In our measurement, the average time was 1.17 seconds, while the minimum delay was 0.86 seconds. Fig. 8 shows the performance of retrieve. In our measurement, the average time was 1.07 seconds, while the minimum delay was 0.86 seconds.

### C. Operation in channel-reflector.net

We are operating our implementation on physical network to evaluate the system in physical world, and promote our system to new users. This operation is performed at the web site of "channel-reflector.net". Each user can access the interface, and also experience our content repository system through the Channel Reflector Operational Site (http://www.channel-reflector.net/).

As a demonstration of our site, we provide the content information of "Arts for All - New Year's Eve Beethoven Cycle in Tokyo" [15] through this site. For the period of this demo, there are 65 unique access in the content information

search, 14 unique access to refer the detail of content. As a result, 17 unique users access to the content.

## V. Related Work

Sdr [16] is a well-known session directory system that has been intensively used in IP multicast environment. It assumes to use the Session Announcement Protocol (SAP) [2] that cooperates to distribute available multicast group information to the directory system. As described earlier, in a SAP announcement procedure, entire content information must be periodically transmitted and all active information must be continuously refreshed. If content information is no longer announced, its description eventually times out and is deleted from the available content list. Although SAP improves protocol robustness and keeps all the content directory instances synchronized, periodic data transmission of all active content descriptions increases additional transmission overhead and latency, and further reduces scalability especially when the number of contents increases [17], [5]. Hence in the practical situation, the SAP has major limitations on the requirements discussed in Section II.

As another notification method, E-mail or RSS is a widely used method to retrieve content information. However, these methods need to join the source of the content before receiving by users. Since the information which is not joined cannot be retrieved, users must use another method to search the information. It is difficult to find available content information in the entire Internet.

A web search engine is of wide use and is flexible to show many kinds of information. However, as described in Section I, because the crawling method takes time for gathering all available content information in the Internet. Especially, real-time streaming contents would be highly dynamically launched and terminated repeatedly, and their contents may be changed frequently. It is hence difficult to provide all up-to-date content information for the future media network.

In summary, existing information management architecture remains the difficulty of retrieving the content information over the Internet. SAP has problems in REQ1, REQ3, REQ5, and REQ6. E-mail or RSS has problems in REQ1 and REQ2. A web search engine has problems in REQ2, REQ4, REQ5.

## VI. Conclusion

In this paper, we proposed a brand-new content information management architecture as a "Content Information Repository" system. We explained requirements of a content information management system, and proposed the content repository system using a distributed data storage which has scalable and low latency data management manner. We then illustrated the design of the content information repository system with 6 components (Interface, Access, Routing, Processing, Local Management, and Storage layer), and provided the actual implementation.

We evaluated the behavior of the implemented system in the PlanetLab overlay network, and measured data transfer estimate time. The experimental results have shown that our system can distribute and retrieve information to the entire Internet in feasible delay. As a result, our system fulfilled the requirements (from REQ1 to REQ6) of content information management. By our system, we expected not only to improve the convenience of end-users, but also to encourage to develop new streaming service architecture.

Our future work includes the deployment of the content information repository system to the Internet communities, for further evaluation with a large number of nodes. We also plan to compare the search performance with another search method (e.g. n-gram).

## References


[1] M. Handley, V. Jacobson, and C. Perkins, "SDP: Session Description Protocol," RFC 4566 (Proposed Standard), Internet Engineering Task Force, Jul. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4566.txt

[2] M. Handley, C. Perkins, and E. Whelan, "Session Announcement Protocol," RFC 2974 (Experimental), Internet Engineering Task Force, Oct. 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2974.txt

[3] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," RFC 3261 (Proposed Standard), Internet Engineering Task Force, Jun. 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3261.txt

[4] A. Lippman, K. Aizawa, R. L. Stevenson, and Y. Zhang, "Video coding for multiple target audiences," in *SPIE VCIP '99: Proceedings of Visual Communications and Image Processing*, vol. 3653, no. 1. SPIE, 1999, pp. 780–782.

[5] H. Asaeda and V. Roca, "Requirement for IP multicast Session Announcement in the Internet," IETF, Internet Draft, Sep. 2010, draft-ietf-mboned-session-announcement-req-03.

[6] J. Risson and T. Moors, "Survey of Research towards Robust Peer-to-Peer Networks: Search Methods," RFC 4981 (Informational), Internet Engineering Task Force, Sep. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4981.txt

[7] B. T. Loo, J. M. Hellerstein, R. Huebsch, S. Shenker, and I. Stoica, "Enhancing p2p file-sharing with an internet-scale query processor," in *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*. VLDB Endowment, 2004, pp. 432–443.

[8] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu, "Mariposa: a wide-area distributed database system," *The VLDB Journal*, vol. 5, no. 1, pp. 048–063, 1996.

[9] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[10] W3C, "Soap version 1.2 part 0: Primer (second edition)," April 2007, w3C Recommendation. [Online]. Available: http://www.w3.org/TR/soap12-part0/

[11] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," *Lecture Notes in Computer Science*, vol. 2218, pp. 329–350, 2001.

[12] K. Mishima and H. Asaeda, "Distributed session announcement agents for real-time streaming applications," in *GLOBECOM Workshops, 2008 IEEE*, 30 2008-Dec. 4 2008, pp. 1–6.

[13] M. Hertz and E. D. Berger, "Quantifying the performance of garbage collection vs. explicit memory management," *SIGPLAN Not.*, vol. 40, no. 10, pp. 313–326, 2005.

[14] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "PlanetLab: an overlay testbed for broad-coverage services," *SIGCOMM Computer Communication Review*, vol. 33, no. 3, pp. 3–12, 2003.

[15] "Arts for All - Streaming Live from Japan to the World in 3D and HD," 2010. [Online]. Available: http://a4a.wide.ad.jp/en/

[16] L. Clark and M. A. Sasse, "Conceptual Design Reconsidered: The Case of the Internet Session Directory Tool," in *HCI '97: Proceedings of HCI on People and Computers XII*. Springer-Verlag, 1997, pp. 67–84.

[17] H. Asaeda, W. Pokavanich, and S. Yamamoto, "Channel Reflector: An Interdomain Channel Directory System," *IEICE Transactions on Communications*, vol. E89-B, no. 10, pp. 2860–2867, October 2006.