ACT: Audio Conference Tool Over Named Data Networking

Zhenkai Zhu
UCLA
Los Angeles, California, USA
zhenkai@cs.ucla.edu

Sen Wang
Tsinghua University Beijing, China
swangfly@vip.qq.com

Xu Yang
Tsinghua University
Beijing, China
ulyx.yang@gmail.com

Van Jacobson
PARC
Palo Alto, California, USA
van@parc.com

Lixia Zhang
UCLA
Los Angeles, California, USA
lixia@cs.ucla.edu

ABSTRACT

In this paper we present the design of an audio conference tool, which is one of our efforts to explore application designs on top of Named Data Networking. Instead of relying on centralized services as current implementations do, ACT takes a named data approach to discover ongoing conferences as well as speakers in each conference, and to fetch voice data from individual speakers. The resulting design is completely distributed and robust against component failures. We discuss design alternatives and tradeoffs, scalability and security considerations, as well as remaining issues for future work.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Communications ApplicationsComputer conferencing, teleconferencing, and videoconferencing

General Terms

Design

Keywords

named data networking, audio conferencing, decentralized

1. INTRODUCTION

The Named Data Networking (NDN) [3] is a newly proposed Internet architecture, which transforms data, instead of hosts, into a first-class entity. A few primitive prototype applications have been implemented on top of NDN [1, 18] as a proof of evidence for the validity of this new direction of using names, instead of addresses, for data delivery. However, the design space of NDN applications is yet to be fully explored both to help us further understand the functional requirements of an NDN network and identify missing pieces, as well as to develop guidelines for designing new applications and migrating of existing applications onto NDN.

In this paper we report our preliminary experience from the design of an audio conference tool, ACT, on top of NDN. ACT is developed to serve several purposes: (1) to help us gain further understanding of application data naming, its relation with routing scalability, and trust model; (2) to serve as a useful tool for the NDN team collaborations, and (3) to generate "real-world" traffic on an NDN testbed that has been set up, so as to facilitate the development of other aspects of the overall architecture, such as network measurement and diagnosis. We show through the course of design how we exploited the advantages of NDN to accommodate the communication patterns of conference applications as well as the trade-offs we made to achieve system simplicity, flexibility, and robustness.

The rest of the paper is organized as follows. We give a brief review of NDN communication in Section 2, and then describe the design of ACT is Section 3. Section 4 discusses the security considerations, and Section 5 discusses the advantages of ACT compared to traditional conference applications and the design trade-offs we made. Finally we discuss the related work in Section 6 and conclude the paper.

2. NDN BACKGROUND

Communication in NDN is driven by the receiving end, *i.e.*, the data consumer. To receive data, a consumer sends out an ***Interest*** packet, which carries a name that identifies the desired data and (optionally) a set of rules to determine which piece of content to return if more than one data object matches the Interest. A router remembers the interface from which the request comes in, and then forwards the Interest packet by looking up the name in its *Forwarding Information Base (FIB)*, which is populated by routing protocols that propagate name prefixes instead of IP prefixes. Once the Interest packet reaches a node with the requested data, the ***Data*** packet P is sent back. P carries the name and the data, together with a signature signed by the original data producer. P traces in reverse the path created by the Interest back to the data consumer. Figure 1 shows the Interest and Data packet in an NDN network.

Since NDN routers keep Interests until the requested data is returned, when multiple Interests for the same data are received from downstream, only the first Interest is forwarded to upstream towards the data source. The router stores the Interest in the *Pending Interest Table (PIT)*, where each

ICN'11 August 19, 2011, Toronto, Ontario, Canada.

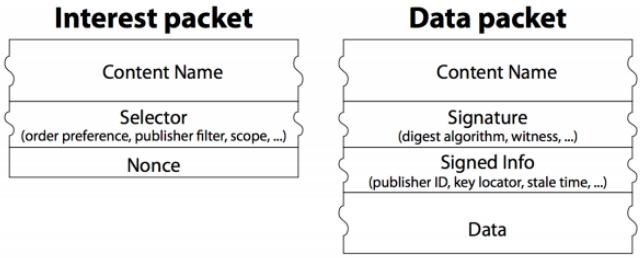

Figure 1: Packets in the NDN Architecture

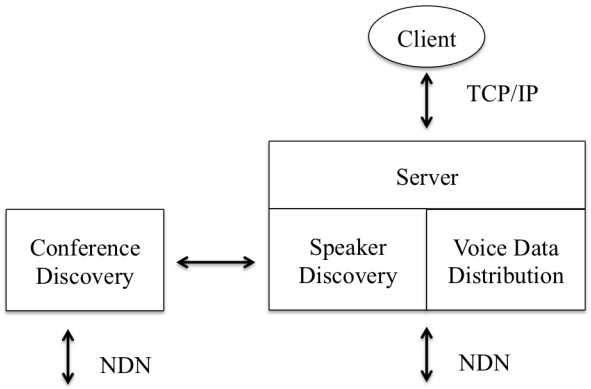

Figure 2: An Overview of ACT Design

entry contains the name of the Interest and a set of interfaces from which the matching Interests have been received. When the Data packet arrives, the router sends a copy to each of the interfaces in the corresponding PIT entry. It then removes the PIT entry and caches the Data in the *Content Store*, which is basically the router's buffer memory subject to a cache replacement policy. An NDN Data packet is meaningful independent of where it comes or where it may be forwarded to, thus the router can cache it to satisfy potential future requests. Note that each Interest retrieves one data packet, hence one can control data flow by regulating Interest generation and forwarding. Also note that NDN data delivery has both multicast delivery and content distribution function automatically built in.

3. ACT DESIGN

We start this section with an overview of ACT design. As an audio conferencing tool, ACT must perform the following three basic tasks for each user interested in participating in a conference: reporting existing conferences (that the user is eligible to learn), delivering voice data to all participants of a conference in real time, and media data processing together with a user interface.

The first task is to collect the latest information about all existing conferences, both scheduled and ongoing ones, and the speakers of each ongoing conference that the user wants to join. This requires that each Interest packet for conference information must be propagated to everywhere across the network. ACT chooses the names for conference information data from a broadcast name space. Once a user learns about the interested conference and the speakers in that conference call, she can receive the voice data by sending Interests directly towards each speaker. Hence, ACT uses two difference name spaces, broadcast name space for conference information discovery and user location based name space for voice data delivery. Interests for the former operates at the frequency of conference launching or speaker joining, while the Interests for voice data streams need to be generated at the frequency of audio packet generation which can be orders of magnitude higher.

The third task, media data processing and user interface design, is a necessary component of ACT but largely decoupled from network specifics. To focus our effort on NDN specific design issues, ACT simply adopted a client-server based open source audio application package, and embeds the modules for speakers discovery and voice data distribution in the original server code, leaving the client intact. A converted server runs on the same node with the client, and communicates with the client using the standard IP protocol stack, while at the same time communicates with other converted servers over NDN.

Figure 2 shows a simple block diagram of our design. A separate module handles the conference discovery and facilitates the task of joining a conference. The decoupling of conference discovery from voice media delivery gives us the flexibility to extend ACT with other features such as video, distributed whiteboard and text messaging. In the rest of this section we describe each of the above three pieces in more detail.

3.1 Conferences Discovery

ACT allows each user to retrieve a list of scheduled or ongoing conferences, and to announce a new conference to potential participants. To retrieve conference information, a user must know the name for the conference information data a *priori*, before she has any knowledge about the conferences to be discovered. Hence, conference initiators and participants must agree on the name prefix for conference description data by following some established application naming conventions [2]. An example of the name for conference data within the NDN scope is */ndn/broadcast/conference/conference-list*. The first name component defines the scope of the network (within NDN testbed), the second one indicates a broadcast namespace (within the NDN testbed scope). The next name component identifies the application type, and the last component specifies the data that the participant is interested in.

Following the established naming convention, a conference initiator announces the conference by creating the conference description data with a proper name under the conference-list prefix. Since there may be multiple different conference description data packets, and any of them can satisfy the Interest with name */ndn/broadcast/conference/conference--list*, we need to ensure that a participant can learn all conferences under that prefix. Furthermore, ACT must also promptly remove finished or canceled conferences from the conference list presented to the users.

3.1.1 Conference Announcement

ACT announces a conference by creating a data object describing the conference in the *Session Description Protocol (SDP)* [4] format according to the user's input, such as

estimated starting time, media type supported, etc. The name of the data is constructed by appending the specific conference name (or the signature of the conference name if the possibility of name collision is high) to the conference-list prefix. This data is then stored in the application buffer until the conference ends. As an example, assuming that node A in Figure 3 announces a conference named "icn2011". Node A listens to Interests with the conference-list prefix, and whenever it receives such an Interest, it responds with the Data packet containing the conference description. As the Data packet travels to the requester(s), it may be stored at the intermediate routers to satisfy Interests from other potential participants in the future. A *Freshness* time is specified in the Data packet which determines the maximum time it can be considered valid. In general, the conference initiator would only receive one conference-list Interest within the *Freshness* seconds, as the Interests for the same name will be consumed by the routers that have the cached conference description data.

A conference initiator may also delegate the task of announcing the conference to a third-party node or other participants in the conference. The delegatee in turn can store the conference description data and respond to conference-list Interests on behalf of the initiator in cases when the initiator gets disconnected, or simply goes offline when the conference is still ongoing.

3.1.2 Conference Enumeration

When a user turns on ACT, the conference discovery module sends an Interest with the name of conference-list prefix, as node B in Figure 3 does. This Interest can be satisfied by any conference description Data packet. Since multiple conference announcements may exist, the module needs to send multiple Interests to discover all existing conferences. However simply sending the same Interest again may not bring back new conference description data, as the Interest is likely be satisfied by the previous Data packet that the module just fetched and is now in the router's Content Store.

NDN addresses this issue by providing *exclusion filter (EF)* as a means for the requester to express explicitly the names of the data it has already received. Depending on the number of names to be excluded, the conference discovery module may simply enumerate the known conference names in the EF of one Interest or split them among multiple Interests [18] so that the size of Interest packet does not exceed path MTU. For example, after node B in Figure 3 learns the conference "icn2011", it will send another Interest with the same name of conference-list but specify in the EF to exclude "icn2011", to retrieve description data about an unknown conference if any exists.

Other mechanisms are also being developed which, together with EF, achieve information discovery in efficient ways.

To get informed immediately whenever a new conference is announced, ACT keeps an outstanding conference-list Interest all the time. That is, it issues another Interest excluding all known conferences as soon as the previous Interest brings back a Data packet or otherwise expires. Although a large number of ACT conference discovery modules may be running simultaneously and all sending out Interests for conference-list all the time, Interests carrying the same name will be aggregated by intermediate routers, and thus there is at most one PIT entry for the name conference-list at any router.

3.1.3 Conference List Maintenance

ACT takes a soft-state approach to ensure that the conferences list is up-to-date. The conference discovery module keeps a refresh timer T_{rf} and a removal timer T_{rm} for each known conference. The value of T_{rf} is specified by the conference initiator, which should be roughly the same as the *Freshness* used for the conference description data. And the value of T_{rm} is the value of T_{rf} plus an estimated maximum delay to retrieve data (taking into account possible Interest or Data packets losses). When the T_{rf} times out, the corresponding conference is no longer included in the EF, and a new Data packet about this conference is expected to be retrieved shortly if the conference is still scheduled or ongoing. Both T_{rf} and T_{rm} are reset if such data comes. If T_{rm} expires, it means that no party is announcing the existence of corresponding conference and it should be removed from the conference list.

3.2 Speakers Discovery

The participants of a conference fall into two categories, the speaker(s) who produces the voice data, and the listeners who request the voice data. ACT only needs to know the speakers in a conference in order to retrieve data from them. For small-scale conferences where every participant is likely to speak, each user is simply both a speaker and a listener. For large conferences, generally speaking, there are fewer speakers than listeners. ACT scales well by tracking only a list of active speakers, instead of all the participants.

Speaker discovery is done when a user joins a conference. The mechanism of speaker discovery is similar to conferences discovery. An example name prefix for speaker description (speaker-list prefix) is */ndn/broadcast/conference/[conference-name]/speaker-list*. When in "speak" mode, an ACT converted server generates and keeps in buffer the data in SDP format describing the name prefix it intends to use for voice data, media type, public key locator, etc. The name of the SDP data is constructed by appending the user name of the speaker to the speaker-list prefix. When changed to "listen" mode, ACT removes the SDP data from the buffer and stops responding to the speaker-list Interest. How a speaker responds to the Interest and how ACT discovers and keeps a list of active speakers are exactly the same as in conferences discovery, except that there is no delegation of speaker announcement.

3.3 Voice Data Distribution

A topology-dependent name prefix (*e.g.* /ucla.edu/cs/-zhenkai) is used by each speaker for its voice data. Because a speaker may generate multiple data streams, further name components such as device ID and audio codec can be appended to the name prefix to identify each stream. A device ID represents the physical device that produced the data stream, and it should be unique within the local network so that the access router knows where to forward Interests for a particular data stream. To ensure uniqueness, a 32-bit random string can be used for the device ID. A speaker should include all name prefixes for all the data streams it produces in the speaker description data.

Segments from each voice data stream produced by the speaker are sequentially named and stored in a circular buffer.

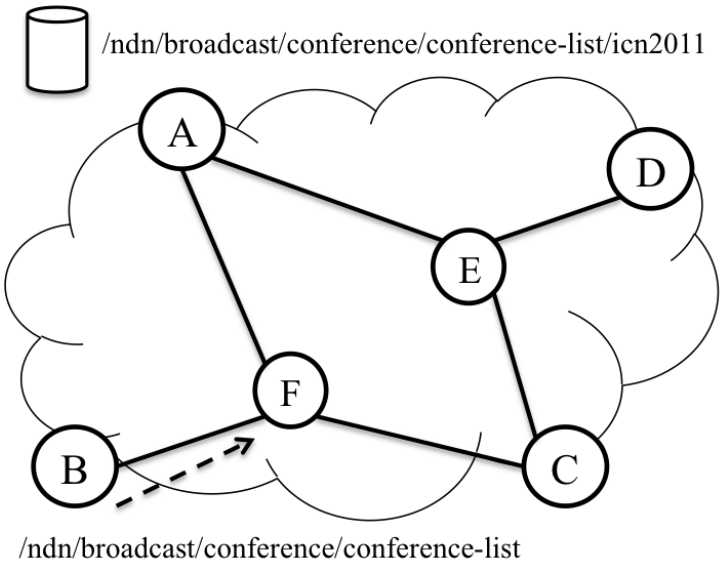

Figure 3: Conference Discovery

A typical name for a data segment may look like: */ucla.edu-/cs/zhenkai/[device-id]/[codec-name]/[seg-num]*.

An ACT converted server sends Interests for all the voice data streams it learned from speaker discovery. When the segment number of a data stream is unknown, an Interest is sent asking for the latest data under the stream name prefix (thus no cached content can satisfy this Interest). This ensures that a listener fetches the latest voice content produced by the speaker, instead of lagging behind. When a data segment from this stream is brought back, the server learns the segment number being used at the speaker and starts to explicitly set segment numbers in the future Interests.

The Interest and Data flow in lock-step, each Interest retrieving a single Data packet. In a network with high latency the Interest-Data round trip can delay the reception times of the voice data packets, rendering them unplayable or with low quality. To address this issue ACT takes the same pipeline approach as described in [1]. The listener maintain a certain number of outstanding Interests for each voice data stream. Whenever a Data packet comes back, it issues a new Interest, thereby restoring the number of outstanding Interests in the pipeline.

3.4 ACT Implementation

We have implemented a proof-of-concept ACT converted server as an extension to the open source project Mumble [15]. We made *murmurd*, the server part of Mumble to exchange data with other murmurds over NDN, based on the CCNx project [18] of PARC that provides both a content router and a set of interface libraries that simplifies the process of application development. At the time of this submission the ACT development is still in progress. The basic function of audio conference is working (e.g. managing speakers, retrieving voice data, etc), and trials of ACT have been carried out on NDN testbed, and the subjective voice quality was quite good. A number of other functions such as encryption-based access control remains to be done.

Stepping up a level, we believe that ACT shows a promising example of how to move existing applications into NDN. ACT keeps the Mumble's client-server model intact, and modified the server to perform NDN data transport. Not only this approach leaves the entire client part untouched, but also the converted ACT server can be a stand alone module sitting between an NDN network and the existing Internet to enable an ACT conference call across the boundary.

4. SECURITY CONSIDERATIONS

A conference tool should be able to offer two basic security functions, conference data authentication (or encryption if secrecy is required), and conference participant control. Assuming that a usable trust management function is provided by either NDN or some other systems [5, 6, 7], so that the application can securely verify the public key of individual participants, ACT supports the two basic functions in the following way.

Packets generated by ACT are digitally signed as a result of built-in security primitive in NDN: the name in each NDN packet is bound to packet content with a signature, ensuring the authenticity of voice data. The corresponding public keys are also distributed over NDN, as specified in the "key-locator" field of the content object. The name prefix used for the public key should be the same as the data its private counterpart signed, this avoids the case where the content fetching goes smoothly but the public key is not reachable due to routing failures.

ACT employs an encryption-based access control scheme to support private conferences as needed. The initiator of a private conference produces a public-private key pair for the conference, which is used to distribute confidential credentials. This provides flexibility of addressing various security needs in a conference. It keeps the public key and distributes the private key to the legitimate participants. Assuming the initiator knows the list of legitimate participants for the conference, whenever an Interest for the conference's public key comes from a legitimate participant P, the initiator responds

with the conference private key encrypted with P's public key, so that only P is able to decrypt.

The conference initiator encrypts a session key using the conference public key and publishes it under a name. Legitimate users who have the conference private key can fetch and learn the session key. All the data communication can then be protected by encryption using the session key. If the initiator decides to revoke the session key, it just publishes a new one under that name.

The initiator can also decide to not respond to the conference-list Interests from unauthorized users, so that the conference name will not even show up in their conference lists.

When the conference initiator decides to delegate the conference announcement task to another node, the delegatee must publish the initiator's public key under routable names, and the initiator should include these names in the "key-locator" field of the conference announcement content object. Thus, no matter where a conference participant fetches the conference description data from, there is always a valid locator to retrieve the initiator's public key, so that it can verify the integrity and the provenance of the conference announcement. Besides, the delegatee needs to distribute the session keys to legitimate users in the conference that is not open to public.

The net result offers much better security than most deployed conference applications, which are often unauthenticated and unencrypted. However, the need to sign and verify every packet suggests an ominous need for efficiency in signature schemes. Fortunately, the previous work [1] suggests that per-packet RSA signatures for real-time data are practical on commodity end-user platforms today.

5. DISCUSSIONS

In this section we discuss the advantages of ACT compared to the traditional conference application implementations, as well as the design trade-offs.

5.1 Scalability

Traditional conference applications typically require central servers that receive data stream from speakers and deliver the streams to all the (legitimate) listeners. The available bandwidth at the server becomes the bottleneck that restricts the maximum number of conference participants. The problem is due to the host-to-host communication model in the current IP networks, which causes the same data piece to be repeatedly transmitted over the same link multiple times.

ACT on the other hand leverages the named data approach to eliminate the repeated transmissions of data packets. The Interests for the same piece of data sent by different listeners will be aggregated by intermediate routers, and only one Interest reaches the targeted speaker. An optimal multicast tree is built automatically through the Interests aggregation, over which the data is delivered efficient to all listeners. Later arrivals of Interests for the same data can be satisfied by the cached Data packets along the paths. Consequently, each piece of data is transmitted over a link no more than once. With this most efficient data distribution, there is no limit on the number of participants ACT can support in an audio conference, under a reasonable assumption that only a few participants may speak at any given time.

5.2 Robustness

Another issue with the traditional conference application implementation is a single point of failure. All participants at least depend on central servers for rendezvous, and in many cases also for data re-distribution. If a server crashes or gets disconnected, all conferences hosted on it fail despite that the participants are still connected and functioning. ACT takes advantage of NDN to form a completely distributed design that is robust again component failures. Wherever failures may be, all connected participants can continue their conferences.

Every coin has two sides. To achieve the most efficient data delivery, ACT necessarily sets up state in routers. Each outstanding broadcasted Interest for conference information occupies an entry in the PIT of each router in the proper broadcast scope. Such cost may well be affordable if the broadcast is well scoped. For broadcast in a global scope, one can explore multiple approaches to keep the cost affordable [8, 9].

5.3 Mobility

Mobile computing devices such as smartphones and tablets are becoming ubiquitous with time. Unfortunately few existing Internet applications are mobile friendly. For example, a client of a conference application may require the user's manual intervention after a move: it needs to register its new IP address with the server and restart joining process for a conference again.

In ACT, nothing needs to be changed if the moving node is a listener: it simply starts sending Interests for voice data as soon as it is re-connected. And a conference announcement is still reachable even when the initiator moves, as the discovery Interests are broadcasted.

As the name prefix for the audio data may change after a speaker moves, the speaker needs to modify its description data. However, the new description data is likely to be excluded by listeners since the speaker is known. ACT reserves a special character '#' for moving speakers. After a speaker moves, it temporarily prepends the character '#' to its user name, so that the new description data would not be excluded by the EF of the discovery Interests. Because there is an outstanding Interest for speakers from all participants, the listeners immediately retrieve the updated speaker description and override the stale one. The speaker switches to its original user name after T_{rm} seconds, when all the listeners should have received the new description data.

To deal with dynamically changing sets of conferences and speakers, ACT requires that there are always outstanding Interests for conferences, and also for speakers if a user is in an ongoing conference, so that as soon as a new conference is announced or a new speaker (including the moving speakers with a temporary name) shows up, the corresponding description data will be brought back right away. One can support such "always-on" Interests either by frequent refreshes if they have a short life time, or otherwise introducing the concept of "long-lived Interests" to reduce the application workload of refreshes. Long-lived Interests represent an interesting new issue that deserves further investigation. Different from other PIT entries used to retrieve existing data and thus short-lived, a long-lived Interest is essentially equivalent to multicast state at routers, the state

that is readily available for most prompt and efficient data delivery. Further study is needed to evaluate such tradeoffs.

6. RELATED WORK

Internet conferencing has had a long history starting from conference applications over APRAnet in the late 70's. In mid 90's MBONE [10] was established and a set of tools and protocols were developed based on the lightweight IP Multicast delivery model [11]. When a conference takes place, several multicast groups and ports are assigned to each service (audio, video, whiteboard, *etc.*). Announcements of the conferences are periodically multicasted to the MBone. Session tools "hear" the announcement and present to the user a list of scheduled or ongoing conferences. Separate programs are launched accordingly by the session tools when the user decides to join a conference. The robust and efficient distribution scheme led to the success of multicast-based conferencing; however, these applications went away after MBONE got dissolved.

Most of the conference systems today fall into the client-server model [12, 13, 14, 15]. Typically a centralized server receives media streams from all the participants in a conference, processes the content (*e.g.* mixing, transcoding, filtering), and re-distributes the streams to the participants. This requires great processing power, bandwidth as well as reliability at the central server. The popular software Skype [16] takes a mixed approach: the user registration and listing is done with a centralized server, while the voice communication happens directly between two peers. When it comes to conferencing, however, one of the participants has to assume the role of a central server. Although it eliminates the need of a common processing and re-distribution central server for all the conferences, a per-conference "central server" still exists, which limits the number of participants in a conference. There are also pure peer-to-peer conference system designs [17]. However, a full mesh delivery model is required. That is, each participant delivers its data stream to and receives data streams from every other participants in a conference using unicast. The inefficiency in data delivery makes it impossible to scale with a large number of participants.

7. CONCLUSION

NDN not only move content efficiently and scalably, it can also implement real-time services such as voice calls and transactions. In this paper, we design a completely distributed audio conference tool using a named data approach to demonstrate that using NDN offers more flexibility in design space and results in a intrinsically more scalable and secure solution. Advanced services (e.g. video, whiteboard, text message) can be easily built on top of ACT as additional components. We believe that ACT can serve as a good example on how to design applications over NDN and also a useful tool for research collaborations as well as personal communications.

8. ACKNOWLEDGMENT

Thanks to Michael Plass, Nick Briggs, James D. Thornton and Rebecca Braynard from PARC and Jeff Burke from UCLA for insightful discussions during ACT design.

9. REFERENCES

[1] V. Jacobson, D. K. Smetters, N. Briggs, M. Plass, P. Stewart, J. D. Thornton and R. Braynard, "VoCCN: Voice Over Content-Centric Networks", *A*CM ReArch'09, December 2009.

[2] V. Jacobson, D. K. Smetters, J. D. Thornton, M. Plass, N. Briggs, and R. Braynard, "Networking Named Content", *A*CM CoNext'09, December 2009.

[3] L. Zhang *et al.*, "Named Data Networking (NDN) Project", *P*ARC Technical Report NDN-0001, October 2010.

[4] M. Handley, V. Jacobson, and C. Perkins, "SDP: Session Description Protocol", *I*ETF RFC 4566, July 2006.

[5] P. Gutmann, "Plug-and-play PKI: A PKI your mother can use", In *P*roceedings of the 12th USENIX Security Symposium, August 2003.

[6] D. Wendlandt, D. Anderson, and A. Perrig, "Perspective: Improving SSH-style host authentication with multi-path probing", In *P*roc. USENIX Annual Technical Conference, June 2008.

[7] E. Osterweil, D. Massey, B. Tsendjav, B. Zhang, and L. Zhang, "Security Through Publicity", *H*OTSEC'06, 2006.

[8] P. Sharma, D. Estrin, S. Floyd, L. Zhang, Technical Report, 1997.

[9] N. Sturtevant, N. Tang, L. Zhang, "The Information Discovery Graph: Towards a Scalable Multimedia Resource Directory", In *P*roceedings of IEEE Workshop on Internet Applications, 1999.

[10] M. Macedonia and D. Brutzman, "MBONE, the Multicast BackbONE", *h*ttp://www-mice.cs.ucl.ac.uk-/multimedia/projects/mice/mbone_review.html

[11] *h*ttp://ee.lbl.gov

[12] K. Singh, G. Nair, and H. Schulzrinne, "Centralized Conferencing using SIP", *I*P telephony workshop, 2001.

[13] *h*ttp://www.asterisk.org/

[14] *h*ttp://www.metaswitch.com/conferencing/feature-list.aspx

[15] *h*ttp://mumble.sourceforge.net/

[16] *w*ww.skype.com

[17] J. Lennox and H. Schulzrinne, "A Protocol for Reliable Decentrialized Conferencing", *N*OSSDAV'03, 2005.

[18] *h*ttp://www.ccnx.org/content/download-releases