Engineering, **2013, 5, 919-927**
Published Online December 2013 (http://www.scirp.org/journal/eng)
http://dx.doi.org/10.4236/eng.2013.512112

Scientific Research

Trust Management for Mobile Media Distribution

Raimund K. Ege
Department of Computer Science, Northern Illinois University, DeKalb, USA
Email: ege@niu.edu

Received May 9, 2013; revised June 9, 2013; accepted June 17, 2013

ABSTRACT

Multimedia content delivery to capable smart phones with high-speed next-generation Internet connectivity is becoming commonplace. However, the openness of delivery demands adaptive and robust management of intellectual property rights. The purpose of this article is to describe a framework to address the central issues in content delivery: a scalable peer-to-peer-based content delivery model. Our method pairs the delivery with a secure access control model that enables data providers to secure a return from making their original content available. Our work resulted in a prototype implementation written in Java that includes a client for the Android mobile platform. Adding robust trust management to scalable peer-to-peer content delivery is the major significance of our work.

Keywords: Multimedia Sharing; Peer-to-Peer Content Delivery; Access Control Management

1. Introduction

It is amazing at what rate that multimedia data are introduced to the Internet and consumed. High bandwidth Internet connectivity is no longer limited to reaching PCs and laptops: a new generation of devices, such as netbooks and smart phones, is within reach of 3G/4G telecommunication networks. Smart phones have ushered in a new era in omni-present broadband media consumption. Services such as YouTube and FaceBook are popularizing deliveries of audio and video content to anybody with a broadband Internet connection. Almost any kind of multimedia data has value to somebody. Releasing it to the Internet carries potential for capturing some of the value, but also carries the risk that the data will be consumed without rewarding the original source.

In this article, we describe a framework for multimedia content delivery that is based on peer-to-peer file sharing. Peers communicate to discover each other, to establish trust, and to exchange data. We describe the implementation of a video player application for the Java and Android platform that delivers video in a secure and managed way.

Delivering multimedia services has many challenges: the ever increasing size of the data requires elaborate delivery networks to handle peek network traffic. A common approach to a large-scale distribution is a peer-to-peer model, where clients that download data immediately become intermediates in a delivery chain to further clients. The BitTorrent protocol is an example of such peer-based data delivery.

Another challenge is to secure and protect the property rights of the media owners. The dynamism of peer-to-peer communities means that principals who offer services will meet requests from unrelated or unknown peers. Peers need to collaborate and obtain services within an environment that is unfamiliar or even hostile. Therefore, peers have to manage the risks involved in the collaboration when prior experience and knowledge about each other are incomplete. One way to address this uncertainty is to develop and establish trust among peers. Trust can be built by either a trusted third party [1] or by community-based feedback from past experiences [2] in a self-regulating system. Other approaches reported in the literature use different access control models [3,4] that qualify and determine authorization based on permissions defined for peers. In such a complex and collaborative world, a peer can benefit and protect itself only if it can respond to new peers and enforce access control by assigning proper privileges to new peers.

The broader goal of our work is to address the trust in peers who are allowed to participate in the content delivery process, to minimize the risk and to maximize the reward garnered from releasing data into the network. In our prior work [5,6], we focused on modeling the nature of risk and reward when releasing content to the Internet. We integrated trust evaluation for usage control

with an analysis of risk and reward. Underlying our framework is a formal computational model of trust and access control. In the work reported here, we focus on the implementation aspects of the framework to establish trust among peers.

Our article is organized as follows: the next section will elaborate on how the data provider and its peers can quantify gain from participating in the content delivery. It also explains our risk/reward model that enables a data source to initially decide on whether to share the content and keep some leverages after its release. Section 3 describes our prototype architecture with its components to identify and authenticate peers, to maintain trust information, and to track swarms of peers as they consume multi-media data. Section 4 introduces prototypes for these components, including a client for the Android platform and its implementation in Java. The article concludes with our assessment of how peer-to-peer systems can shed their freewheeling image via sensible access control additions. The research reported here is an extension of a work that appeared in the Proceedings of the First International Conference on Mobile Services, Resources, and Users (MOBILITY 2011) [7].

2. Qualifying the Value of Multimedia

It is amazing at what rate multimedia data is introduced to the Internet and consumed. Almost any kind of multimedia data has value to somebody. Releasing it to the Internet carries potential for reaping some of the value, but also carries the risk that the data will be consumed without rewarding the original source. In addition to the cost of creating the original multimedia data, there is also a cost associated with releasing the data, *i.e.*, storage and transmission cost.

For example, consider the life of a typical "viral" video found on a popular social media site: the video is captured via a smart phone camera (maybe even accidentally), then is uploaded to the social media site, discussed (*i.e.*, "liked" and "friended"), and viewed by a large audience (measured in millions of hits). The video taker is rewarded with fame, rarely gets a monetary reward, the entity that is getting rewarded is the social media site, which will accompany the video presentation with paid advertising.

Let us first recap our model to asses risk and reward, by quantizing aspects of the information interchange between the original source, the transmitting medium and the final consumer of the data. Our emphasis here is on the reward quantity, rather than on how trust in peers affects the outcome.

In a traditional fee for service model the reward "R" to the source is the fee "F" paid by the consumer minus the cost "D" of delivery:

$$R = F - D$$

The cost of delivery "D" consist of the storage cost at the server, and the cost of feeding it into the Internet. In the case of a social media site, considerable cost is incurred for providing the necessary network of servers and their bandwidth to the Internet. The social media site recovers that cost by adding paid advertising on the source web page as well as adding paid advertising onto the video stream. The site's business model recognizes that these paid advertisings represent significant added value. As soon as we recognize that the value gained is not an in-significant amount, the focus of the formula shifts from providing value to the original data source to the reward that can be gained by the transmitter. If we quantify the advertising reward as "*A*" the formula now becomes:

$$R = F - \left(D - A\right)$$

Even in this simplest form, we recognize that "*A*" has the potential to outweigh "*D*" and therefore reduce the need for "*F*". As the social media site recognizes, the reward lies in "*A*", *i.e.*, paid ads that accompany the video.

Mediation frameworks can capture the mutative nature of data delivery on the Internet (see also our prior work [8]). As data travels from a source to a client on a lengthy path, each node in the path may act as mediator. A mediator transforms data from an input perspective to an output perspective. In the simplest scenario, the data that is fed into the delivery network by the source and is received by the ultimate client unchanged: *i.e.*, each mediator just passes its input data along as output data. However, that is not the necessary scenario anymore: the great variety of client devices already necessitate that the data is transformed to enhance the client's viewing experience. We apply this mediation approach to each peer on the path from source to client. Each peer may serve as a mediator that transforms the content stream in some fashion. Our implementation employs the stream control transmission protocol (SCTP) which allows multi-media to be delivered in multiple concurrent streams. All a peer needs to do is add an additional stream for a video overlay message to the content as it passes through.

The formula for reward can now be extended into the P2P content delivery domain, where a large number of peers serve as the transmission/storage medium. Assuming "*n*" number of peers that participate and potentially add value the formula for the reward per peer is now:

$$R_p = \sum_{i=1}^{n}\left(F_i - \left(D_i - A_i\right)\right) - F_p$$

D_i and A_i are now the delivery cost and value incurred at each peer that participates in the P2P content delivery. F_i is the fee potentially paid by each peer. F_p is the fee

paid to the data source provider. Whether or not the data originator will gain any reward depends on whether the client and the peers are willing to share their gain from the added value. In a scenario where clients and peers are authenticated and the release of the data is predicated by a contractual agreement, the source will reap the complete benefit.

In our model, we quantify the certainty of whether the client and peers will remit their gain to the source with a value of trust. Trust is evaluated based on both actual observations and recommendations from referees. Observations are based on previous interactions with the peer. Recommendations may include signed trust-assertions from other principals, or a list of referees that can be contacted for recommendations. Our model enables an informed decision on whether to accept a new peer based on the potential additional reward gained correlated to the risk/trust encumbered by the new peer.

3. Peer-to-Peer Architecture

The architecture of our peer-to-peer multimedia delivery framework encompasses components that aim to deliver multi-media content from a source to a large number of clients. We assume that the content comes into existence at a source. A simple example of creating such multimedia might be a video clip taken with a camera and a microphone, or more likely video captured via a smartphone camera, and then uploaded to the source. We assume that clients consume the content, e.g., by displaying it on a computing device monitor, which again might be a smartphone screen watching an Internet video. We further assume that there is just one original source, but that there are many clients that want to receive the data. The clients value their viewing experience, and our goal is to reward the source for making the video available.

In a peer-to-peer (P2P) delivery approach, each client participates in the further delivery of the content. Each client makes part or all of the original content available to further clients. The clients become peers in a peer-to-peer delivery model. Such an approach is specifically geared towards being able to scale effortlessly to support millions of clients without prior notice, *i.e.*, be able to handle a "mob-like" behavior of the clients.

The nature of the source data will dictate the exact details of delivery: for example, video data is made available at a few preset quality levels using variable-rate video encoding. The typical video source data stream is a series of sequential frames: each frame is identified by its frame number. Clients request frames in sequence, receive the frame and reassemble the video stream which is then displayed using a suitable video decoder and display utility. The video stream is encoded in such a fashion that missing frames don't prevent a resulting video from being shown, but rather a video of lesser bit-rate encoding, *i.e.*, quality, will result [9]. We explicitly allow the video stream to be quite malleable, *i.e.*, the quality of delivery need not be constant and there is no harm if extra frames find their way into the stream. It is actually a key element of our approach that the stream can be enriched as part of the delivery process.

In addition, all frames are encrypted to ensure confidentiality, integrity and authenticated access [10,11].

The central element of our architecture is the central trust management server. Peer clients need to register with this server: all information that establishes the trust in them is maintained here. Other components of the architecture are the provider of the original source data, and peers that consume the multimedia stream. Peers can also serve as further sources in a peer-to-peer download model.

The information maintained for each peer is the peer identification, location, trust value and history of participation in data stream delivery. When a new source registers its data stream with the central trust server, the source also provides minimum trust criteria, which guide the trust server when granting peer access to the stream. Only peers that match the trust requirements with their trust value and history are given access to the data stream. The source also sets a bonus trust value, which is used to adjust peer's trust value to reflect their participation in the content delivery network.

Figure 1 shows the system schematic of our framework for a content delivery network. The elements are "source", "trust server", "relay peer" and "edge peer".

The connections between them represent communication among them as they establish the network and proceed with data exchange. The content delivery network shown is populated with one source, the trust server, and

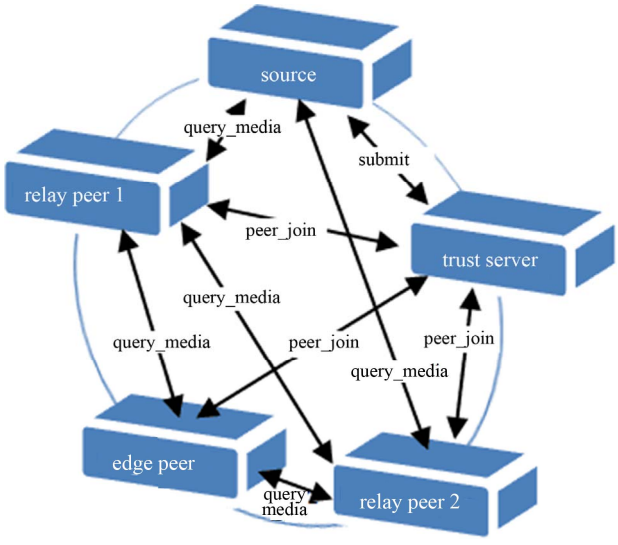

Figure 1. System schematic.

3 peers: 2 relay peers and one edge peer.

The source is where the video data is produced, encoded, encrypted and made available. The source submits the stream info to the trust server. Peers connect to the trust server for authentication and to receive the download credentials. A peer that only downloads is called an "edge peer". Once the peer starts serving the stream to other peers, it becomes a "relay peer". All peers together maintain a peer group, *i.e.* information on which peers are actively part of the content delivery network. The trust server initially informs the peers in the peer group which source to download from: peer 1 is fed directly from the source; peer 2 joined somewhat later and is now being served from the source and peer 1; the edge peer joined last and is being served from peer 1 and peer 2. In this example, peer 1 and 2 started out as edge peer, but became relay peers once they had enough data to start serving as intermediaries on the delivery path from original source to ultimate consumer.

Peers stay in contact with the trust server while media data is exchanged. The peer needs authorization to be able to consume, *i.e.* decrypt and display, the data. Authorization is granted via an authorization token that is used for decryption. The token has a limited life span; it can be revoked by the trust server based on the trust behavior of the peer. Once all streaming for a specific data source has completed, the trust server adjusts the peer's trust value in the trust server's database by a bonus value, reflecting the peer's positive cooperation. Conversely, if the peer violated the trust placed in it, the bonus value is deducted from the peer's trust value.

4. Java Implementation

Our implementation has 4 major components: 1) the trust server; 2) an application that allows a source to submit information about a data stream; 3) a relay peer that consumes data, *i.e.* shows the video, and makes it available to other peers; and 4) an edge peer to run on a mobile device. All 4 components are implemented in Java.

We chose the Android platform to implement a proof-of-concept client for a mobile device. Android is part of the Open Handset Alliance [12]. Android is implemented in Java and therefore offers a flexible and standard set of communication and security features.

The communication among the peers within their peer group uses session initiation protocol (SIP) messaging based on the Sip2Peer library [13]. The actual media exchange uses the Java implementation [14] of the SCTP [15] transport layer protocol. For authentication we use the OpenID [16] which provides a convenient and flexible way to establish the identity of peers, and OAuth [17] which is an open protocol to allow secure authorization in a simple and standard method from desktop and web applications.

Peer communication is achieved via session SIP messages. Each message has a message type and carries a payload. The initial message is of type "peer_join" that a new client peer sends to the trust server. Another important type of message is "query_media", which in-quires about which media is available and maintained by the peer group. The answer to this message is a list of which peers are able to serve which parts of the available media. The answer also provides communication details such as the IP and port number at which a peer will serve up frames of the media. Every peer constantly monitors the rate of response it gets from the other peers and adjusts its connections to the peers from which the highest throughput rate can be achieved.

In the following, we will showcase four prototypes of these components and discuss details of their implementations.

4.1. Trust Server

The central component of our architecture is the *Trust Server*. It maintains a database of all peers and a tracks the collection of data streams that are made available by sources. The information maintained for each peer is its name, its OpenID and associated identity and security information, the exact network location, the current trust value, and the history of current and past streams that it participated in.

Our Trust Server prototype presents a display of all peers. **Figure 2** shows a snapshot of the *Peer Trust Manager*.

All peers are maintained centrally by the trust server. When a peer connects with the trust server, authentication follows a multi-factor process: 1) the peer's openId is validated via the openID provider; 2) the peer's public key is retrieved from the trust server's database; and 3) the openID is used to request an OAuth authentication token. In our current implementation, we retrieve the OAuth token from the same provider that maintains the peer's OpenID. The OAuth token is specific to the data

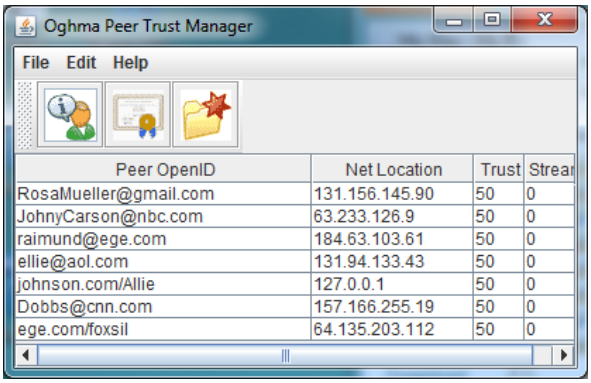

Figure 2. Trust manager peer list.

stream that the peer has requested. The token is the key used to decrypt the data stream.

Figure 3 shows the security information for a specific client, in this case the Diffie-Hellman public key [11] of the peer. It is assumed that the peer stores the matching private key. All communication past authentication is encrypted using these keys.

The Trust Server also maintains the list of currently available data streams. For each stream it captures the stream's name, its uniform record locator (URL), the trust value set by the source to qualify peers, and the bonus value used to reward/penalize peers for participating. **Figure 4** shows a snapshot of the *Active Stream Manager*.

For example, consider the stream named "Blizzard of 2011": it can be streamed from the URL "oghma://welcome.today.ege.com:5012". It requires peers to have a minimum trust value of 65, and if the peer successfully completes participation, the peer's trust value will be incremented by a bonus of 10.

The snapshot in the figure also shows the log window, which displays the current status of the trust server: all activities of the content delivery network are logged here.

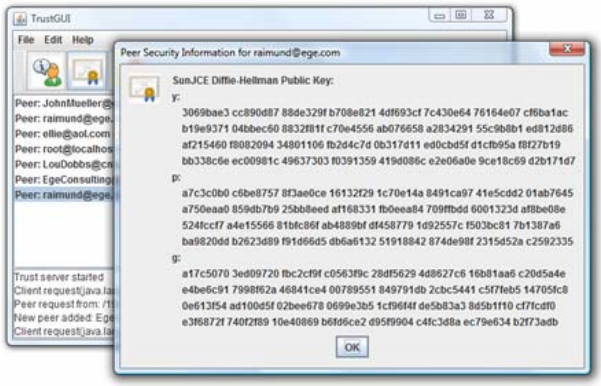

Figure 3. Peer security information.

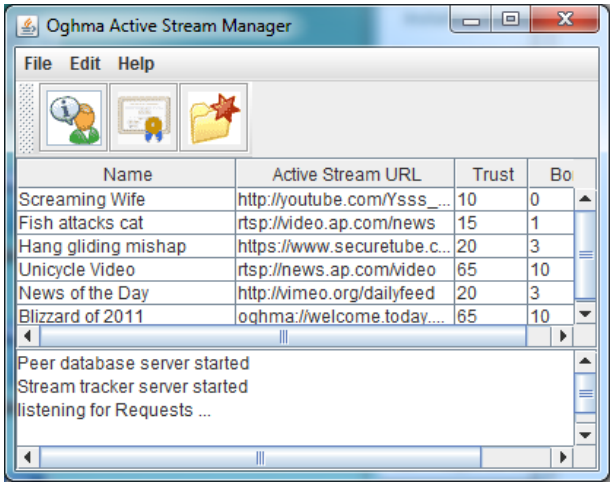

Figure 4. Active stream list.

4.2. Submit Source Dialog

Arguably, the source is the most important component of our framework. Without interesting and compelling video content, no content delivery network will be successful. **Figure 5** shows a screen capture of the *Secure Media Submission* dialog.

The source provides the stream information such as its name and exact stream in location, *i.e.* uniform record locator (URL). Peers will use this address to retrieve the stream. The source also set the minimum required trust level: each peer must have at least this value to participate in the stream delivery. And finally, the source sets the trust bonus that successful participation will earn the peer. The bonus is also used as penalty to a misbehaving peer's trust value.

Not shown here are the other parts of our prototype that allow a peer to adjust stream properties, and also revoke as stream completely, *i.e.* remove it from the content delivery network.

4.3. Relay Peer

The current implementation of our prototype relay peer is as a Java application. Nothing would prevent us from providing the same capability as a web application.

The Java application is used to allow a peer to authenticate with the trust server, get a listing of available streams, make a selection, display the stream and finally also make the stream available to other peers downstream. **Figure 6** shows a screen capture of the *Java Peer Client* prototype:

Once the peer is authenticated with the trust server, it can request a list of available streams. **Figure 6** shows all streams that are currently available with their name, required trust value, and potential bonus. The peer can make any selection. The reasons why all stream are dis-

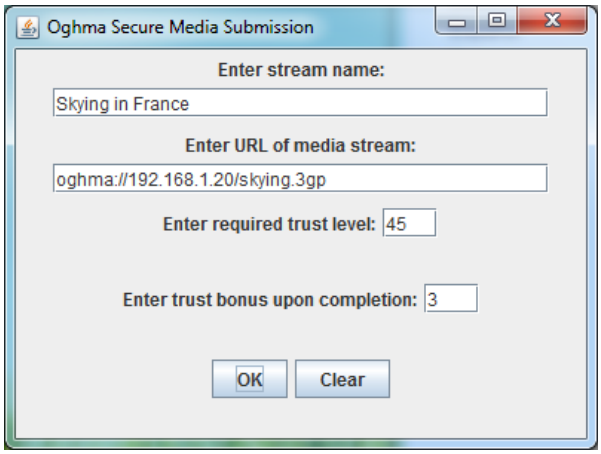

Figure 5. Secure media submission.

played, even the ones which require a higher trust value than what the peer currently has, is to give the peer an incentive to first participate in another stream to add the bonus to its trust value. However, only streams can actually be selected for which the peer is currently qualified.

Figure 7 shows such a case:

Once the peer has selected a stream for viewing, the trust server will transmit the necessary information to enable the peer to start download. It will get the set of all locations at which the media stream is available, plus the necessary access token to enable stream decryption. The peer then contacts the source locations at their streams' URLs and starts downloading data, *i.e.* the sequential frames of the video stream.

In general, peers can do 3 things: 1) they continuously request frames from other peers (the original source is viewed as just another peer) and store them; 2) they may display the frames as video to the user of the peer device; 3) and they make the stored frames available to other peers.

Figure 8 shows our prototype Java implementation of

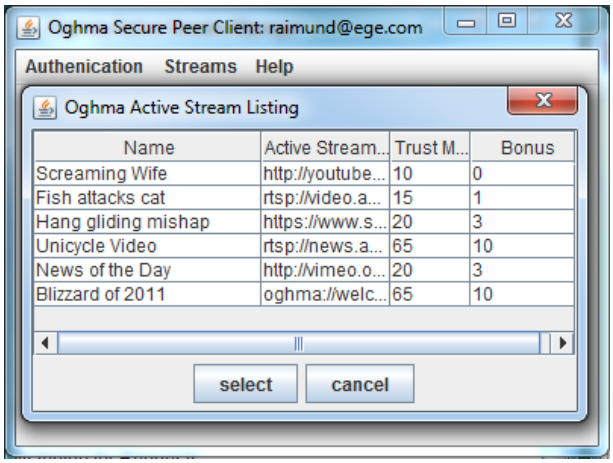

Figure 6. Peer stream selection.

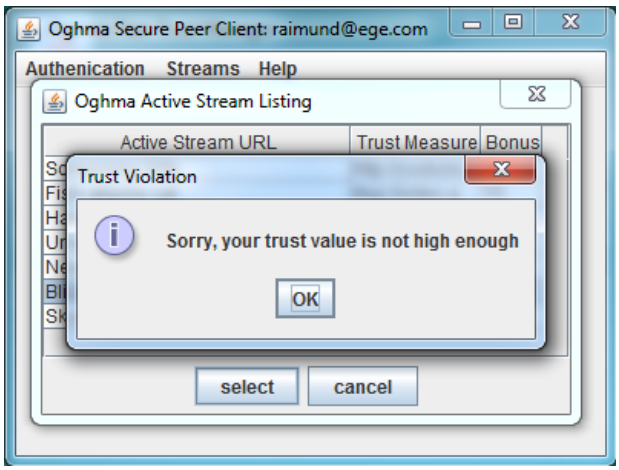

Figure 7. Peer stream selection failure.

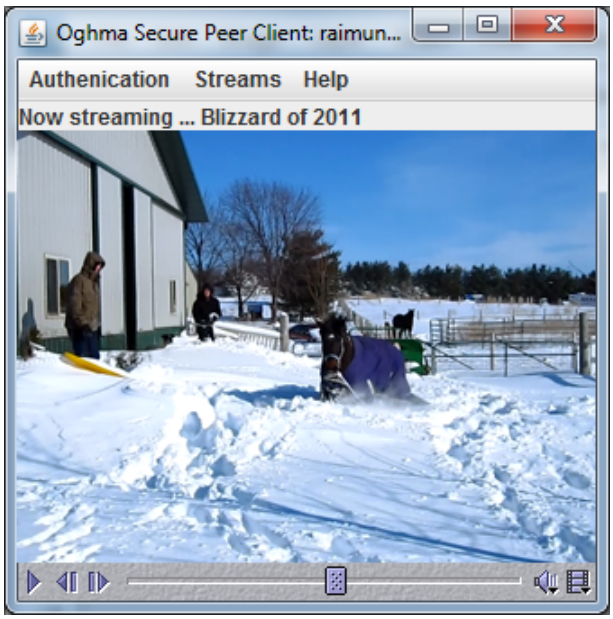

Figure 8. Peer streaming video.

our Peer Client while it displays the requested video:

Peers don't need to provide all 3 services. A peer that provides only service (1) and (2) is an "edge" peer, *i.e.*, an end user consumer. A peer that provides service (1) and (3) is a "relay" peer. Relay peers are specifically important for peers that have limited access to the public Internet, *i.e.*, peers behind network boundaries, such as a NAT firewall. In addition, peers stay in contact with each other to continuously update the peer group and source data availability.

A peer client consists of three processes:

1) A process to communicate with the trust server. The client initiates the negotiation with the server to get admitted into the content delivery network. Upon success, the server informs the client which sources the client should use accompanied by their access tokens. The client will update the tracker on its success in down-loading the source data;
2) A process to request data from the given sources. Frames may be requested from multiple sources. Frames that are received are decrypted using access token for the given data source.
3) A process to sequence the frames received from sources and to assemble them into a usable media stream.

Our prototype uses the SCTP [15] transport layer protocol. SCTP is serving in a similar role as the popular TCP and UDP protocols. It provides some of the same service features of both, ensuring reliable, in-sequence transport of messages with congestion control. We chose SCTP because of its ability to deliver multimedia in multiple streams. Once a client has established a SCTP association with a server, packages can be exchanged with

high speed and low latency. Each association can support multiple streams, where the packages that are sent within one stream are guaranteed to arrive in sequence. Each source can divide the original video stream into set of streams meant to be displayed in an overlay fashion. Streams can be arranged in a way that the more streams are fully received by a client, the better the viewing quality will be. The first stream is used to deliver a basic low quality version of the video stream. The second and consecutive streams carry frames that are overlaid onto the primary stream for the purpose of increasing the quality. In our framework we also use the additional streams to carry content that is "added value", such as advertising messages or identifying logos. The ultimate client that displays the content to a user will combine all streams into one viewing experience.

4.4. Android Edge Peer

The final component of our prototype framework is our proof-of-concept edge peer implementation for the Android platform. In the following we will show three screenshots: **Figure 9** shows the login screen, **Figure 10** shows the stream selection screen, and finally **Figure 11** shows the video being streamed onto the mobile device.

First, **Figure 9** shows the login screen. Like in the Java application, each peer is authenticated with its OpenID credentials. The user enters userid and password,

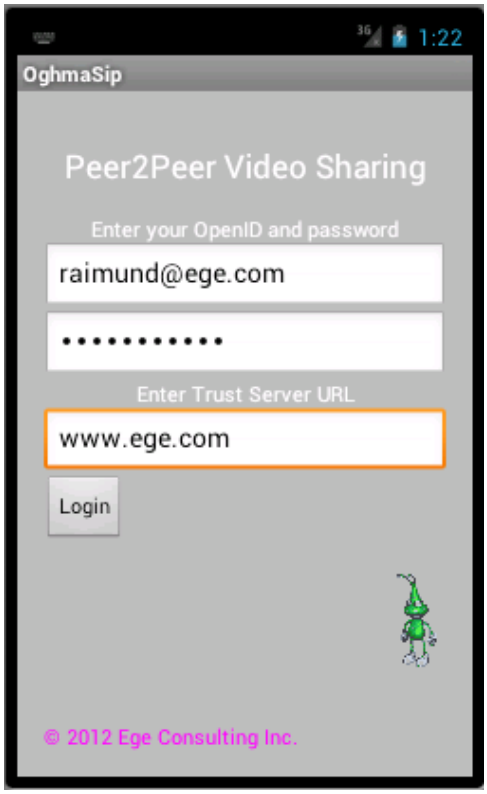

Figure 9. Oghmasip login screen.

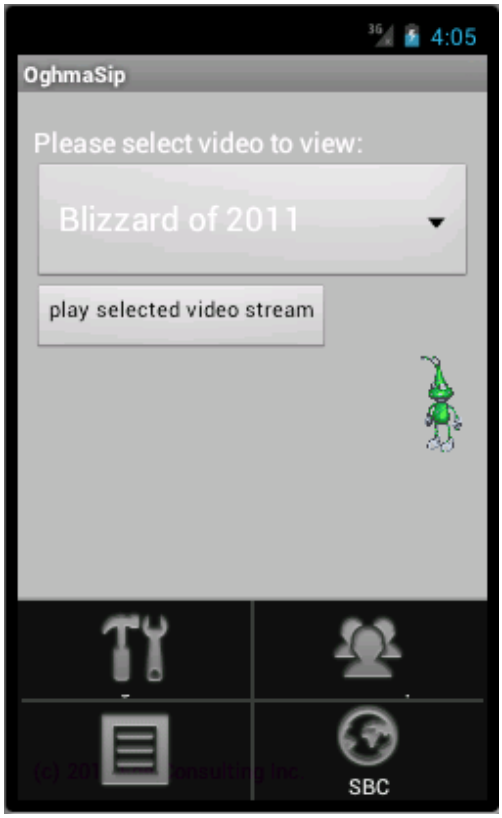

Figure 10. OghmaSip available video screen.

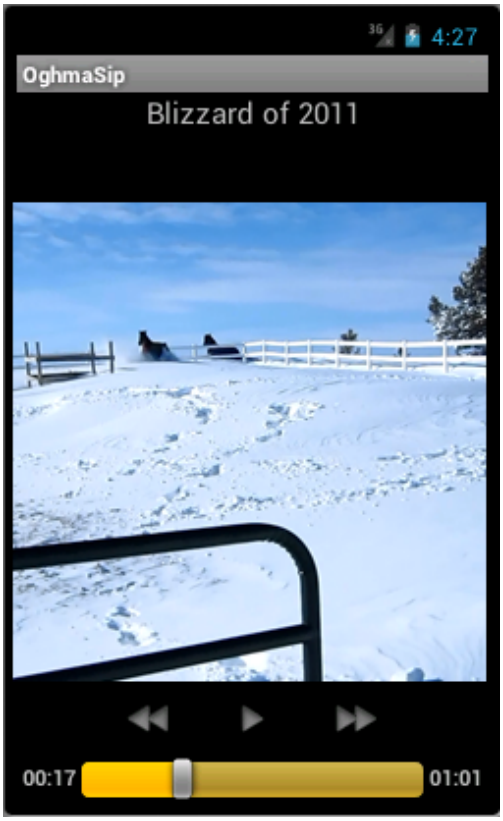

Figure 11. OghmaSip video delivery screen.

plus the URL of the central trust server. If the peer is new to the content delivery network, it will also generate a public/private pair of Diffie-Hellman keys [11], keep one private and submit the public one to the trust server.

Once authentication is achieved, *i.e.* the OpenID provider has sent the authorization token, the user is show which streams are currently available on the next screen.

Figure 10 presents a drop-down list of streams that are available to be served to this mobile device. The screen also offers a menu to allow the peer to modify configuretion data, see information about other peers in the peer group, and to manage its own security information.

Once the "play selected video stream" button is pressed, the user is shown the video display screen, as shown in **Figure 10**. Once a sufficient read-ahead buffer has been accumulated, the video stream starts playing on the Android device (**Figure 11**).

Our Android prototype implementation uses a second feature of the SCTP protocol: we use is its new class *SctpMultiChannel* which can establish a one-to-many association for a single server to multiple clients. The *SctpMultiChannel* is able to recognize which client is sending a request and enables that the response is sent to that exact same client. This is much more efficient than a traditional "server socket" which for each incoming request spawns a sub-process with its own socket to serve the client. Each packet that is received on the channel carries a *MessageInfo* object which contains information on the actual client that is the actual other end point of this association. The code to receive *SctpMultiChannel* packets is logically similar to any UPD or TCP style of socket receive data packets programming.

5. Conclusions

In this article, we described a framework for new content delivery networks that almost implements access control for its participating peers. We have described a prototype implementation written in Java to establish a P2P network, where a group of peers disseminate information on which sources are available to download, and it includes a Java-based client for the Android platform for smartphones. Such P2P content delivery has a great potential to enable large scale delivery of multimedia content. Our framework is designed to enable content originators to assess the potential reward from distributing the content to the Internet. The reward is quantified as the value added at each peer in the content delivery network and gauged relative to the actual cost incurred in data delivery but also correlated to the risk that such open delivery poses.

The scenario of a typical "viral" video found on a social networking site is considered. The video is captured via a cell phone camera by a user, and the user then uploads the video onto the site. The social networking site stores the video and makes it available to other users for free. The video becomes popular and is viewed by a large audience driving traffic to the social networking site. The only entity that is getting a reward is social media site, which accompanies the video presentation with paid advertising. The only benefit that the original source of the video gets is notoriety. Using our model, the original data owner can select other venues to make the video available via a peer-to-peer approach. The selection on who will participate can be based on how much each peer contributes in terms of reward but also risk. Peers will have an interest in being part of the delivery network, such as Facebook and YouTube which have recognized its value. Peers might even add their own value to the delivery and share the proceeds with the original source. Whereas in the social media approach, the reward is only reaped by one, and the original source has shouldered all the risk, *i.e.*, lost all reward from the content, and our model will enable a more equitable mechanism for sharing the cost and reward.

REFERENCES

[1] Y. Atif, "Building Trust in E-Commerce," *IEEE Internet Computing*, Vol. 6, No. 1, 2002, pp. 18-24. http://dx.doi.org/10.1109/4236.978365

[2] P. Resnick, K. Kuwabara, R. Zeckhauser and E. Friedman, "Reputation Systems," *Communications of the ACM*, Vol. 43, No. 12, 2000, pp. 45-48. http://dx.doi.org/10.1145/355112.355122

[3] E. Bertino, B. Catania, E. Ferrari and P. Perlasca, "A Logical Framework for Reasoning about Access Control Models," *Proceedings of the 6th ACM symposium on Access Control Models and Technologies*, Chantilly, 3-4 May 2001, pp. 41-52. http://dx.doi.org/10.1145/373256.373261

[4] S. Jajodia, P. Samarati, M. L. Sapino and V. S. Subrahmanian, "Flexible Support for Multiple Access Control Policies," *ACM Transaction Database System*, Vol. 26, No. 2, 2001, pp. 214-260. http://dx.doi.org/10.1145/383891.383894

[5] L. Yang and R. Ege, "Integrating Trust Management into Usage Control in P2P Multimedia Delivery," *Proceedings of 20th International Conference on Software Engineering and Knowledge Engineering*, Redwood City, 1-3 July 2008, pp. 411-416.

[6] R. K. Ege, Y. Li and R. Whittaker, "Extracting Value from P2P Content Delivery," *Proceedings of the 4th International Conference on Systems*, Cancun, 1-6 March 2009, pp. 102-108.

[7] R. K. Ege, "OghmaSip: Peer-to-Peer Multimedia for Mobile Devices," *The 1st International Conference on Mobile Services, Resources, and Users*, Barcelona, 23-29 October 2011, pp. 1-6.

[8] R. K. Ege, L. Yang, Q. Kharma and X. Ni, "Three-Layered Mediator Architecture Based on DHT," *Proceedings of the 7th International Symposium on Parallel Architec-*

tures, Algorithms, and Networks, Hong Kong, 10-12 May 2004, pp. 317-318.

[9] C. Wu and B. C. Li, "R-Stream: Resilient Peer-to-Peer Streaming with Rateless Codes," *Proceedings of the* 13*th ACM International Conference on Multimedia*, Singapore, 6-11 November 2005 pp. 307-310. http://dx.doi.org/10.1145/1101149.1101211

[10] P. Gutmann, "The Design of a Cryptographic Security Architecture," *Proceedings of the* 8*th USENIX Security Symposium*, Washington DC, 23-26 August 1999, pp. 153-168.

[11] Network Working Group, "Diffie-Hellman Key Agreement Method, Request for Comments: 2631," RTFM Inc., 1999.

[12] Open Handset Alliance, 2010. http://www.openhandsetalliance.com/

[13] Sip2Peer, SIP-Based API for Robust Connection and Communication among Peers, 2011. http://code.google.com/p/sip2peer/

[14] java.net—The Source for Java Technology Collaboration, The JDK 7 Project, 2010. http://jdk7. java.net/

[15] R. Stewart, "Stream Control Transmission Protocol, Request for Comments: 4960," IETF Network Working Group, 2010. http://tools.ietf.org/html/rfc4960

[16] OpenID, 2012. http://www.openid.net

[17] OAuth, OAuth Community Site, 2012. http://www.oauth.net/