Decentralized Bittorrent Social Online Network

Chang-Shyh Peng and Suhail Alsuhail
Computer Science Department, California Lutheran University
Thousand Oaks, CA, 91360, USA
peng@CalLuheran.edu

ABSTACT

With the emergence of Facebook, Twitter, and LinkedIn, web-based social online networks have become increasingly popular among users worldwide as a way to connect with friends, family, and colleagues. There are however some serious drawbacks with many of these social online networks; including false user identity, security of private data, server farms' consumption of energy, etc. This paper proposes a Decentralized BitTorrent Social Online Network that builds on open source protocol. The design goals of this novel model confidentiality, integrity, scalability, and authenticity.

KEYWORDS

Distributed, social, online, network.

1 INTRODUCTION

Social Online Network (SON) service is an online service that focuses on building social networks between individuals or groups of individuals in a centralized management platform. They allow users to interact and share media with each other based on interests, connections, and so on [1,2]. Examples of such service are Facebook, Twitter, and LinkedIn. The convenience of real-time sharing of mostly personal information comes with the price of a plethora of issues [3]. Among them, privacy and intellectual property are two major infamous issues that frequently emerge in the discussion of appropriate design and use of social networks.

Privacy conventionally denotes the limited authorized access to users' data. Intellectual property is concerned with the ownership of published digital information, including the photos and videos uploaded to social networks. Currently most SON services retain a copy of every photo or video uploaded onto their servers. Many SON services further claim the copyrights of the user's data, and are therefore self-empowered to publish, sell, or use any work of the users' without users' consent [4,5,6].

Some research proposed a few adjustments to the current website framework in order to provide better protection of users' content [7,8,9]. However, these adjustments provided little relief to the issues of privacy and intellectual property. Other research tackled the privacy issues with web-based SONs, but they demonstrated very limited control over the web-based SONs [10]. Another approach, a Decentralized SON, presents a completely different model in which the administrative privilege is shifted from the centralized servers to individual users [11].

Decentralized SON is modeled on peer-to-peer architecture, in which each networked node can function both as a client and as a server. Users store individual information on their own devices, and share via peer-to-peer file transfer. Currently there are several projects attempting to create fully functional Decentralized SON, including Safebook, Persona, MAZE, and MyNet. However, the issue with these services is their ground-up approach. Safebook, for example, is currently experimenting on a local network with limited number of users. While it is a very valuable research project in a tightly controlled environment, it does not appear to be

suitable for large scale general public deployment [12,13,14].

This paper presents an advanced framework: Decentralized BitTorrent Social Online Network (Dbt-SON). Dbt-SON has a server side and a client side. The server side authenticates users before connects clients to each other. And the client side communicates directly with each other to request/send shared information. Dbt-SON is designed to be secure, efficient, and scalable. In a typical web-based SON, users' data is located on the SON servers, which, if compromised, can put users' data at great risks. Dbt-SON on the other hand supports secure registration, login, and sessions. Users will be allowed to share information on if they are on the friends' list. Otherwise access will be denied. Data transferred on the Dbt-SON network will always be validated with a SHA1 hash that is generated by the users. The SHA1 hash is stored in the torrent file that is created by the user. Furthermore, users have the option for end-to-end encryption in order to prevent man-in-the-middle attacks. In section 2, the design framework is discussed. Section 3 presents the development and evaluation. Paper concludes with future work in section 4.

2 DESIGN FRAMEWORK

Decentralized BitTorrent Social Online Network is a Decentralized Social Online Network that incorporates BitTorrent. BitTorrent [15,16] is one of the most popular peer-to-peer protocols for file sharing and distribution. Its clients are available for numerous computing platforms and operating systems. BitTorrent data traffic is secure and is hard to be deciphered. Its users' identifies are typically very well protected. BitTorrent can be extremely fast and cost-effective as the responsibility of uploading files is shared among the users [17]. Section 2.1 describes the BitTorrent. Secion 2.2 discusses the Dbt-SON design.

2.1 BitTorrent

BitTorrent needs the following components to promote is functionality:

- a web server
- a torrent file (metainfo)
- a BitTorrent tracker
- an uploader (original seeder)
- end user BitTorrent downloaders

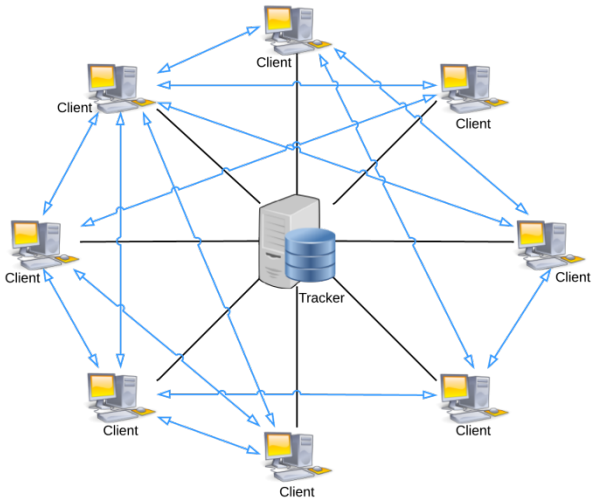

Figure 1. BitTorrent Client and Tracker Interaction Model

Figure 2 depicts the BitTorrent client model. To share a file, user completes the following steps:

- start a BitTorrent tracker or use one that is already running
- start a web server or use one that is already running
- generate the metainfo (.bittorrent) file and the URL address of the tracker
- upload the .torrent file to the web server
- start the BitTorent downloader, which is directed to the original data file

For other users to download shared files, they do the following:

- install a BitTorrent downloader
- download the metainfo .torrent file from the web server
- select local folder to save the download
- wait for the download to complete

- change user role from a leecher (a simultaneous downloader and uploader) to a seeder (an uploader)

The metainfo .torrent file is UTF-8 encoded. It uses Bencode [18] to encode the metadata. Bencode has the following BNF syntax:

- dictionary = "d" 1*(string anytype) "e" ; non-empty dictionary
- list = "l" 1*anytype "e" ; non-empty list
- integer = "i" signumber "e"
- string = number ":" <number long sequence of any CHAR>
- anytype = dictionary / list / integer / string
- signumber = "-" number / number
- number = 1*DIGIT
- CHAR = %x00-FF; any 8-bit character
- DIGIT = "0" / "1" / "2" / "3" / "4" / "5" / "6" / "7" / "8" / "9"

A typical metainfo .torrent contains the following fields:

- announce: the tracker address
- announce list: an optional field which contains a list of different trackers instead of a single tracker
- comment: an optional field that can be edited by the author of the metainfo file
- created by: an optional field that can contain the name and version of the editor of the metainfo file
- creation date: an optional field that contains the creation time, in Unix epoch format, of the metainfo file
- encoding: an optional field on the format of metainfo file encoding
- info: description of the file, structure, and sharing settings

Contents of the info field can vary depending on whether the metainfo file is a single file or a collection of files. In both cases the info field contains:

- piece length: size of each piece (chunk) in bytes
- pieces: 20-byte SHA1 hash values of each piece
- private: an optional field, if set to 1, the download client may only connect to users provided by the tracker, if set to 0, the download client may connect to any users

In the case of a single file, the info field contains:

- name: the file name
- length: length of the file in bytes
- md5sum: optional field, MD5 hash of the file

If the metainfo file is collection of files, the info field contains:

- name: name of the default folder where the files will be saved
- files: list of files each contains the following
  - length: file length in bytes
  - md5sum: an optional field, MD5 hash of the file
  - path: the folder, local to the torrent directory, to save the file

The BitTorrent tracker's main purpose is to connect peers to each other. The tracker does not contain any file contents; it receives requests and responds using the http or https protocol. Peers send http requests to the tracker using http get method with the following:

- info_hash: a 20-byte SHA1 hash of the info key's value from the metainfo .torrent file
- peer_id: a 20-byte string that the client assigns itself that is typically generated each time the client starts
- IP: optional field where the actual IP address of the client device
- port: the port (typically 6881-6889 but others can be available) that the client is

listening to and waiting for incoming connections from other peers
- uploaded: the total number of bytes uploaded to other peers
- downloaded: total number of bytes downloaded from other peers
- left: number of bytes left for the client to complete the download
- event: optional field that states an event, *started* (first request to tracker), *completed* (when the peer finishes downloading a file), or *stopped* (when peer exits)
- numwant: optional field containing the number of peers the client wants to get from the tracker
- compact: an optional field whose value is 1 when the peer will accept responses from the tracker
- key: an optional field that is used in private trackers to prove the identity of the peer if the IP changes
- trackerid: an optional field that can be set if a tracker sent its id previously

When the tracker receives a request, it must respond with a bencoded message. If there was an error, the tracker responds with only a human-readable error message. Otherwise, the tracker responds with a bencoded message which contains the following keys:

- warning message: an optional field that is similar to an error message except the request is still processed by the tracker
- interval: the time, in seconds, that the peer should wait to send a new request to the tracker
- min interval: the minimum time the peer should wait before asking for a new connection
- tracker id: a tracker id that the peer should send back for its future requests
- complete: the number of peers who have the whole file (i.e. seeds)
- incomplete: the number of peers which are still downloading (i.e. leeches)
- peers: a list of other peers to connect to

In order to establish connection, peers first *handshake* where each peer listens to a port for new incoming connections. A handshake is a string of 68 bytes containing the following items in order:

- name length: one byte for the length of protocol name
- protocol name: a string of the protocol name
- reserved: a field reserved for future expansions.
- info hash: the 20-byte SHA1 hash of the info key in the metainfo file; also used to contact the tracker
- peer ID: a 20-byte string that is also sent to the tracker

After the handshake, both peers can send messages to each other without further delay. There are two types of messages: state-oriented messages and data-oriented messages. The purpose of a state-oriented message is to inform each other of the current states. Following are the possible states:

- keep-alive: no ID, no payload, peer asks to keep the connection alive
- choke: state ID 0, no payload, peer is choked when it cannot send any data request
- unchoke: state ID 1, no payload, peer is no longer choked
- interested: state ID 2, no payload, peer indicates interest in receiving new data pieces
- uninterested: state ID 3, no payload, peer indicates no interest in receiving new data pieces
- have: state ID 4, payload of 4 bytes, the index of a data piece that the peer has successfully downloaded and validated
- bitfield; state ID 5, variable payload, indication of which data pieces already received and which data pieces needed

There are three formats of data-oriented messages which deal with sending and requesting of data pieces:

- request: state ID 6, payload of 12 bytes, denoting the data piece of interest
- piece: state ID 7, variable payload, containing a block of data.
- cancel: state ID 8, payload of 12 bytes, sent after a request message when the corresponding data piece is no longer needed

2.2 Dbt-SON design

Dbt-SON is designed with the control flow as shown in Figure 2. Dbt-SON contains a client side and a server side where the client side is developed in Java and the server side is implemented with PHP and MySQL. Dbt-SON is a closed BitTorrent system in which only registered users can upload .torrent files and only friends can download each other's .torrent files.

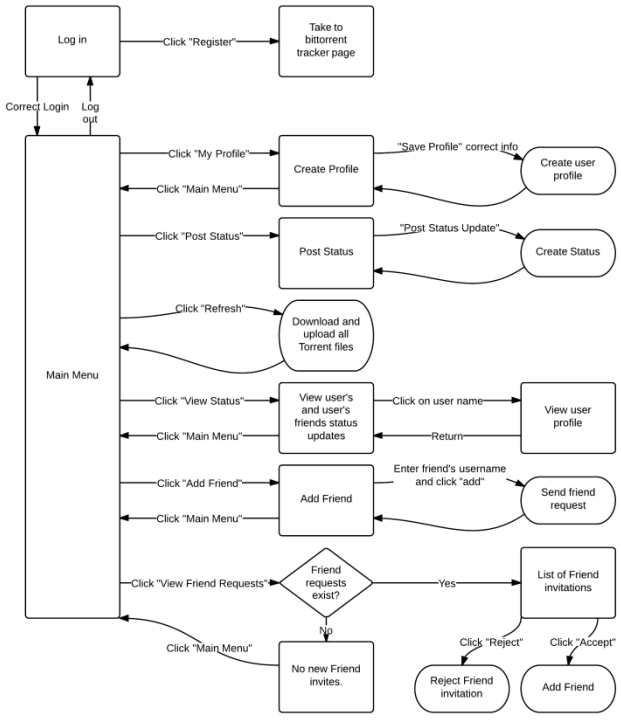

Figure 2. Dbt-SON Control Flow

The following services are added to the existing ones in BitTorrent:

- registration at the tracker: users register for a user name and password
- client services gateway: a PHP page that takes client requests and responds to them, for example, profile creation, profile update, addition of friend, etc.

On the client side, the following functions are included:

- creation, view, and update of profile
- post and deletion of status update
- view of a single friend's status and profile
- view of all friends' statuses
- auto download and upload of new .torrent files
- addition and removal of friends
- support of public and private keys

Figure 3 depicts the interactions between server and client. The primaries correspondences are:

- Login: User name and password are sent to javaLogin.php which responds with the user's ID in the system if the user name and password are correct, or with an error message if otherwise.
- Upload a torrent file: A torrent file, title, and description is sent to javaUpload.php. The server responds with "File uploaded successfully" if no issues occurred, or "Error uploading file" if error is encountered.
- Download a torrent file: Client sends torrent file ID to javaDownload.php. If the user is allowed to download the torrent file, the server replies with the torrent file. If the client is not allowed, server replies with an empty message.
- Get a list of friends' torrent files: Client sends a request to javaTorrentsList.php. The server then replies with a list of the last created friends' torrent files.
- Get a list of new friend requests: Client send a request to

javaFriendRequests.php. The server replies with a list of the new friend requests, or an empty list if no friend requests exist.

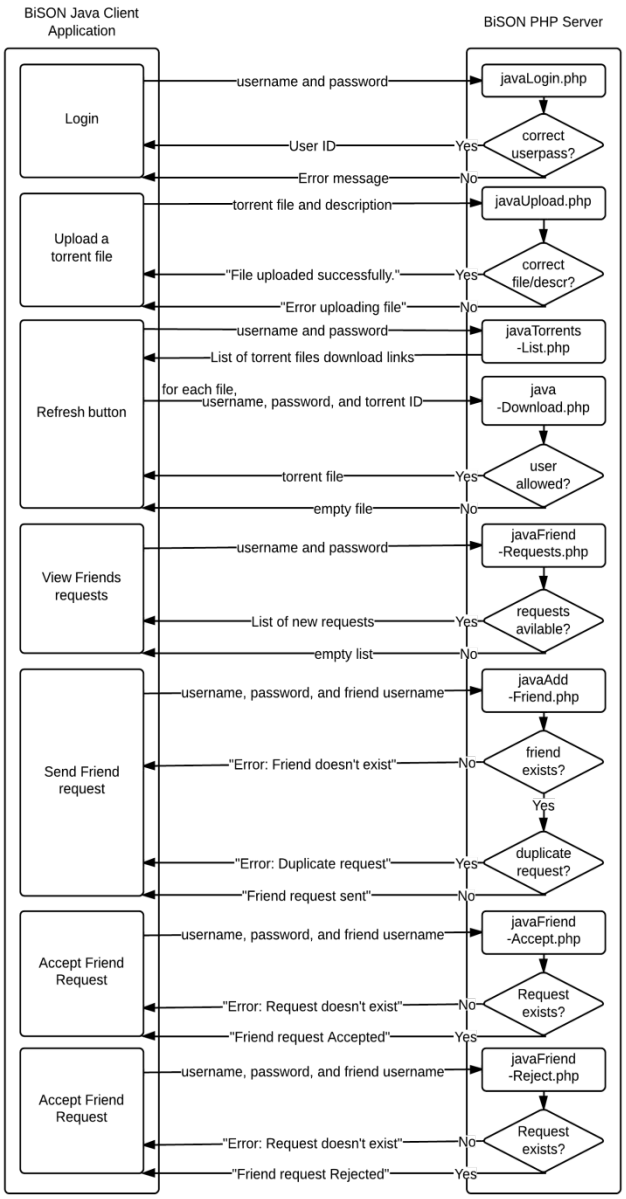

Figure 3. Client Server Interactions

- Send a friend request: Client sends the friend's user name to javaAddFriend.php. If the user name does not exist, the server responds with "User does not exist". If the users are already friends, the server replies with "Already friends". If the user exists and they are not friends, the server adds the friend request to the database and responds with "Friend request sent".
- Accept a friend request: Client sends the user name of the friend who previously sent an add friend request to javaFriendAccept.php. The server respond with "Friend request accepted" if there was a friend request from that user, or "No friend request from this user" if there was no request.
- Reject a friend request: Client sends the user name of the friend who previously sent an add friend request to javaFriendReject.php. The server responds with "Friend request rejected" if there was a friend request from that user, or "No friend request from this user" if there was no request.

3 IMPLEMENTATION AND EVALUATION

The development source code is available at [19]. phpMyBittorent is built to support closed online communities. Only registered users are allowed to upload and download files. Each user gets a special key in order to download the torrent files from the tracker. SSL can be used to encrypt and protect the data from any sniffing or man-in-the-middle attacks. Server-side data is validated for all incoming data. Correct formatting and SQL or HTML injection are verified. The minimalistic and lightweight design yields fast data communication. According to Google's developers PageSpeed Insights Test [20], the tracker URL *phpmbt.010flm.com/announce.php* scored 100/100 on both mobile tests and desktop tests.

There are three types of communication types in this system, server-to-client, client-to-server, and client-to-client. Server-to-client communication is when the server checks the client's integrity by validating the client's user id and password. The client-to-server

communication is when the client checks the server's integrity via SSL signed certificate. The client-to-client communication is when clients verify the integrity of the transferred data.

Several measures are put in place to ensure authenticity for user credentials. User name and IP address will be locked after five unsuccessful login attempts. This can help prevent any risks that jeopardize authenticity. Users of different system roles and privileges are supported. Strong passwords are required. For example, passwords should be at least 8 characters long that contain at least one number, one letter, and one symbol. Passwords have preset expiration time. Passwords are hashed before being stored in the database. Further, to prevent false identity in profile creation or status update, asymmetric cryptography is incorporated [21]. With asymmetric cryptography, a public key and a private key are generated. The public key is made known to the general. The private key is kept secretly to the owner.

There are three factors of the .torrent files accessibility: tracker availability, distributed hash table (DHT), and seeder availability. Tracker is the key to the success of the overall data communication. As a redundancy, DHTs serve as the backup for the trackers. DHTs are decimalized databases where the IPs of different peers, as well as the hashinfo from the metainfo .torrent file, are stored under each torrent file. Seeders availability is an issue that users on Dbt-SON network need to be aware of; the more seeders the faster data exchange. As for the scalability, the key is the seeders to leechers ratio. The file transfer speed is usually bottle-necked with the seeder's speed. However, that changes as a soon as there is a matching number of seeders and leechers [22,23].

The major issues regarding integrity and availability are the Distributed Denial of Service (DDoS) attacks, and selfish peers [24]. DDoS attacks usually target the trackers. A DDoS attack can potentially put a stop to the entire data exchange. Fortunately DHTs can provide a counter measure to this risk. Selfish peers are peers who participate mostly in download but not upload. Such peers can cause great delay in the overall swarm sharing speed. Closed trackers are typically a good solution for this problem because the tracker would stop such peers from further download until they start sharing what they have downloaded.

Dbt-SON is evaluated against the Facebook. Facebook stores a copy of user's data on its own servers and use those supposedly private data at Facebook's discretion. In Dbt-SON, users save keep the files on personal storage. Facebook uses the traditional user id and password authentication, which can be compromised by the man-in-the-middle attack. Dbt-SON on the other hand verifies user's true authenticity before participation of any information sharing. Facebook is a web-based SON and therefore has the upper hand in data availability. Currently, Facebook uses an estimated 180,000 servers to support its extensive online community. It is understandably very costly to support such service [25]. Dbt-SON is potentially a much more economical solution for social online networking. As for the speed in accessing shared information, Facebook is more efficient in sharing photos, but Dbt-SON is shown to be more responsive in sharing videos.

4 CONCLUSION

The Decentralized BitTorrent Social Online Network, which adopts the open source BitTorrent, is shown to be a viable option for social online networking. Its scalability and availability is a double-edged sword since a lot of the responsibility is on the shoulders of the users and their computers. A fully cryptography-based decentralized system would greatly enhance the system.

Following directions are considered for future improvements:

- asymmetric cryptography based server-to-client integrity check
  Server sends user an encrypted random message with user's public key, validation is considered successful if user can successfully decrypt the message with its private key.
- inclusion of an expiration date to every status update
  This action would prevent overloading the system with user updates. It should also improve the system availability and scalability.
- development of a server-to-server collaboration protocol
  This protocol can be enabled on any user who wishes to participate the Dbt-SON as a server, and therefore makes the overall Dbt-SON more scalable and cost effective.

REFERENCES

[1] D. Boyd, and N. Ellison, Social Network Sites: Definition, History and Scholarship, *Journal of Computer-Mediated Communication 13(1)*, pp. 210-230, 2008.

[2] H. Gao, J. Hu, T. Huang, J. Wang, and Y. Chen, Security Issues in Online Social Networks, *IEEE Internet Computing, vol. 15, no. 4*, pp. 56-63, 2011.

[3] M. Hansen, A. Schwartz, and A. Cooper, Privacy and Identity Management, *IEEE Security & Privacy 6(2)*, pp. 38-45, 2008.

[4] J. Bauer, Digital Death - Social Media Owns Your Data, http://technorati.com/social-media/article/digital-death-social-media-owns-your/.

[5] T. Ingram, Does Facebook Own Your Photos?, http://www.tyleringram.com/blog/does-facebook-own-your-photos.

[6] Who Owns Photos and Videos Posted on Facebook or Twitter?, http://www.nyccounsel.com/business-blogs-websites/who-owns-photos-and-videos-posted-on-facebook-or-twitter/.

[7] E. Aimeur, S. Gambs, and A. Ho, Towards a Privacy Enhanced social Networking Site, *Proc. of the 2010 IEEE Conference on Availability, Reliability and Security (ARES)*, pp. 172-179, 2010.

[8] J. Bonneau, and S. Preibusch, The Privacy Jungle: On the Market for Data Protection in Social Networks, *Proc. of The 8th Workshop on the Economics of Information Security*, pp.121-167, 2009.

[9] B. van den Berg and R. Leenes, Audience Segregation in Social Network Sites, *Proc. of the IEEE International Conference on Social Computing*, pp. 1111-1116, 2010.

[10] L. Cutillo, R. Molva, and T. Strufe, Privacy Preserving Social Networking through Decentralization, *Proc. of the 6th International Conference on Wireless On-Demand Network Systems and Services*, pp. 145-152, 2009.

[11] B. Furht, Decentralized Online Social Networks, *Handbook of Social Network Technologies and Applications, Springer*, pp. 349-378, 2010.

[12] R. Baden, A. Bender, N. Spring, and B. Bhattacharjee, Persona: An Online Social Network with User-Defined Privacy, SIGCOMM, pp. 135-146, 2009.

[13] L. Cutillo, R. Molva, T. Strufe, and S. Antipolis, Safebook: a Privacy Preserving Online Social Network Leveraging on Real-Life Trust, Communications Magazine, IEEE, vol 47, issue 12, pp. 94-101, 2009.

[14] D. Kalofonos, Z. Antoniou, F. Reynolds, M. Van-Kleek, J. Strauss, and P. Wisner, *MyNet: A Platform for Secure P2P Personal and Social Networking Services*, Sixth Annual IEEE International Conference on Pervasive Computing and Communications, pp. 135-146, 2008.

[15] B. Cohen, The BitTorrent Protocol Specification, http://bittorrent.org/beps/bep_0003.html.

[16] J. Fonseca, B. Reza, B., and L. Fjeldsted, BitTorrent Protocol Version 1.0, http://jonas.nitro.dk/bittorrent/bittorrent-rfc.html.

[17] F. Bader, A. Radoveneanu, and H. Ragab-Hassen, A New Security Architecture for BitTorrent, *IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 451-455, 2011.

[18] Bencode, http://en.wikipedia.org/wiki/Bencode

[19] Distributed BiTorrent Social Online Network Source Code, https://drive.google.com/file/d/0B5TOGbkpt0cVM0JoOGtOR2t6U3M/view?usp=sharing.

[20] Google's developers PageSpeed Insights Test, http://developers.google.com/speed/pagespeed/insights/.

[21] Asymmetric Cryptography, http://en.wikipedia.org/wiki/Public-key_cryptography.

[22] L. D'Acunto, T. Vinko, and J. Pouwelse, Do BitTorrent-like VoD Systems Scale under Flash-Crowds?, http://www.inf.u-

szeged.hu/~tvinko/vod_scalability_analysis.pdf.

[23] G. Neglia, G. Reina, H. Zhang, D. Towsley, A. Venkataramani, and J. Danaher, Availability in BitTorrent Systems, *http://people.cs.umass.edu/~arun/papers/BT_availability.pdf*.

[24] R. Guha and D. Purandare, Security Issues in BitTorrent like P2P Streaming Systems, *http://www.academia.edu/664476/Security_Issues_in_BitTorrent_like_P2P_Streaming_System*.

[25] J. Hamilton, James Hamilton's Blog: Fun with Energy Consumption Data, *http://perspectives.mvdirona.com/2012/08/13/FunWithEnergyConsumptionData.aspx*.