A Secure Session Transfer Protocol for Downloading a Large File across a Cluster of Servers in the Presence of Network Congestion

Natarajan Meghanathan[1] and Bhadrachalam Chitturi[2]

[1] Jackson State University, Jackson, MS 39217, USA
nmeghanathan@jsums.edu
[2] Amrita Vishwavidyaapeetham University, Amritapuri Campus, Kerala, India
bhadrachalam@am.amrita.edu

Abstract. We propose the design of a Session Transfer Protocol (STP) that allows a client to download a large file replicated across several servers. STP runs at the session layer, on the top of the standard Transmission Control Protocol (TCP). A client can sequentially download the entire file from one or more servers, from one server at a time, with just one TCP session. A STP Server, currently sending the contents of a file to a client, can proactively detect congestion in the network and transfer a file download session to another peer STP Server that is located in a different network. At any stage (initial session establishment or session transfer), the STP Client chooses a particular server by executing certain selection tests among the servers in the list sent by the STP Gateway, which is the public face of the cluster of STP Servers in the Internet. Unlike the traditional File Transfer Protocol (FTP) that requires users to repeatedly initiate the entire download process upon the failure of each FTP connection, STP is seamless, incremental and provides improved Quality of Service while downloading a large file. The user working at the STP Client is unaware of the congestion and resulting session transfer to a different STP Server. STP is security-aware and has appropriate encryption, authentication and anti-spoofing features incorporated at different stages of its execution.

Keywords: Session Transfer Protocol, Sequential Download, Large File Download, Quality of Service, Secure Download.

1 Introduction

With the phenomenal growth in the Internet and the diversity of consumer applications, the size of the files being downloaded keep increasing from KB through MB to GB. The traditional File Transfer Protocol (FTP) with a single server that runs on the top of the connection-oriented Transmission Control Protocol (TCP) [10] is often considered unsuitable for downloading larger files over the Internet. A commonly employed strategy to counter the single server bottleneck problem is to employ multiple mirror servers and let the client choose one of these servers for download. Even in this scenario, once a server is chosen, the client has to stay with

N. Meghanathan et al. (Eds.): CCSIT 2012, Part I, LNICST 84, pp. 552–562, 2012.

that server for the entire download process. If a client starts experiencing more delay in the download process and wishes to download from another server, the client has no option other than completely disconnecting from the first server and opening a new TCP connection with the second server and starting the download all over again from the first byte of the file. For example, if a client is downloading a huge file (such as an .iso file for virtual machine operating systems) that is in the order of GB and if network congestion sets in after half of the file has been downloaded and the client apparently sees no appreciable progress in the download, it becomes quite exasperating for a client to start the process all over again with a new server. There is no guarantee that the client will not again experience the same problem with the new server after a while.

To counter the problem of relying on a single client - single server model, downloading in parallel has been considered as a viable alternative (e.g. [1][2]). Here, the distinct segments of a file are downloaded in parallel from multiple servers and the downloaded contents are merged at the client to reconstruct the original file. However, parallel downloading has several drawbacks. A critical drawback is the requirement to maintain multiple TCP connections at the client side, with each of the parallel servers from which the file is being downloaded. It becomes tedious for thin clients (client machines with very limited resources) to maintain multiple TCP connections and the associated memory buffers for a download session. The client is overloaded until the download is completed. In addition, proper security features need to be embedded in the parallel downloading schemes.

Another strategy that is gaining prominence in recent times is peer-to-peer file sharing with technologies such as the BitTorrent protocol [9]. Here, files are no longer hosted at a particular server or a mirror of servers. A file is broken into pieces and distributed among several machines across the Internet; the information about these machines is stored as part of a metadata for the file. An interested client wishing to download a file contacts the machines listed in the metadata of the file. As the different pieces of the file get downloaded, the client itself becomes a host from which other interested peer clients can download. Peer-to-peer file sharing again requires the client to re-order the downloaded pieces of the file before being delivered to the application and it is highly prone to out-of-order packet arrival. Hence, peer-to-peer file sharing systems are not typically suitable for streaming applications that require progressive or contiguous downloading.

We propose a novel Session Transfer Protocol (STP) for downloading a huge file over the Internet in a sequential fashion using just one TCP session at any given time (between the client and a chosen server) while providing improved Quality of Service (QoS) and a secure (reliable) download. The STP runs at the session layer, on top of TCP at the transport layer. Here, we conceptualize a cluster of cooperating file servers, each of which hosts the entire file. The cluster is publicly identified through a gateway, which is the initial point of contact for an interested client. The gateway, by itself, does not store any file – however, it maintains a database (STP database) that has information about the cooperating servers hosting each of the files. The gateway merely forwards this information to the requesting client in the form of a secure STP ticket, which has to be used by the client to initiate a download session with any of

the cooperating servers of the file. During the download process, as the client sends out Acknowledgments for the last packet that has arrived in-order, the server evaluates the variance in the round-trip times (RTT) of the acknowledgment packets. If the RTT starts to increase beyond a threshold, the server considers this as a sign of impending congestion on the path to the client. As a proactive measure, the server decides that the client has to choose some other server to continue the session and hands-off by sending an encrypted 'Transfer Session' message that includes the session details (such as last byte acknowledged, window size, etc); the client selects the next suitable server from the list of cooperating servers for the file through a ping-request-reply cycle [10] and forwards the encrypted Transfer Session message and the STP ticket originally sent by the gateway. If the chosen server can accommodate the new session with the required QoS, it responds positively. Otherwise, it rejects the connection request.

The STP Client maintains a list of overloaded and unavailable servers and updates this list based on the recent STP sessions it has gone through. After a server positively responds to the session transfer, the STP Client continues to download the remainder of the file from that server. If the session has to be further transferred to another server, the above process is repeated. However, we anticipate that there will not be several session transfers as a STP Server accepts a connection request only if it is able to provide the required QoS in terms of maintaining the same sender window size, etc. The only unknown parameter here is the network bandwidth. The bandwidth on the path between the client and server may be sufficient at the beginning of the session transfer or session initiation. But, after a while, the intermediate networks and the routers on the path between the client and server may be overloaded with traffic, necessitating a session transfer for quick, real-time download. However, at any time, a client has to run only one TCP connection and has to deal with only one server. Hence, STP is perfectly suitable for thin clients. The File Transfer Protocol (FTP) that runs at the application layer, on the top of TCP, can be suitably modified to run STP at the Session layer. We will refer to the modified FTP as STP-aware FTP.

The rest of the paper is organized as follows: Section 2 analyzes related work on parallel downloads and motivates the need for a secure sequential download, especially for thin clients, and at the same time provides the required QoS. Section 3 presents a detailed design of the proposed Session Transfer Protocol (STP) and provides a qualitative comparison with that of the traditional FTP. Section 4 concludes the paper and discusses future work.

2 Related Work

In [1], the authors propose a Parallelized-File Transfer Protocol (P-FTP) that facilitates simultaneous downloads of disjoint file portions from multiple file servers distributed across the Internet. The selection of the set of parallel file servers is done by the P-FTP gateway when contacted by a P-FTP client. The number of bytes to be downloaded from each file server is decided based on the available bandwidth. We observe the following drawbacks with P-FTP: (1) The P-FTP client would be

significantly overloaded in managing multiple TCP sessions, one with each of the parallel file servers. Thus, P-FTP cannot be run on thin clients that are limited in the available memory and resources to run concurrent TCP sessions for downloading a single file. (2) If the path to a particular file server gets congested, the P-FTP client is forced to wait for the congestion to be relieved and continue to download the remaining bytes of the portion of the file allocated for download from the particular file server. The QoS realized during the beginning of the download process may not be available till the end due to the dynamics of the Internet. (3) P-FTP has no security features embedded in it. Hence, it is open for spoofing-based attacks on the availability of the parallel file servers by unauthorized users/clients who simply launch several parallel download sessions that appear to originate from authentic users/IP addresses.

In [2], the authors propose a Dynamic Parallel Access (DPA) scheme that is also based on downloading a file in parallel from multiple servers, but different from P-FTP in the sense that the portion of the file and the number of bytes to be downloaded from a particular file server is not decided a priori; but done dynamically based on the response from the individual servers. In this scheme, the client chooses the set of parallel servers to request for the file. The download is to be done in blocks of equal size. Initially, the client requests one block of the file from every server. After a client has completely received one block from a server, the client requests the particular server for another block that has not yet been requested from any other server. Upon receiving all the blocks, the client reassembles them and reconstructs the whole file. Unlike P-FTP, DPA is less dependent on any particular mirror server as it requests only one block of the file from a server at a time and does not wait for several blocks of the file from any particular server. However, with DPA, the client cannot close its TCP connections with any of the mirror file servers until the entire file is downloaded. This is because, if a client fails to receive a block of the file from a particular mirror server and has waited for a long time, then the client has to request another peer mirror server for the missing block. In order to avoid opening and closing multiple TCP connections with a particular mirror server, the client has to maintain the TCP connection with each of the file servers until the entire download is completed. The client has to keep sending some dummy packets to persist with the TCP connections. On the other hand, a P-FTP client can close the TCP connection with a P-FTP server once the required portions of the file are downloaded as initially allocated from the particular mirror server. DPA also does not have any security features embedded in it.

Many other related works (e.g., [3][4][5]) on simultaneous partial download have also been proposed in the literature for better QoS. All of these schemes use parallel downloading to fasten the throughput and minimize the delay. But, this will be a significant overhead on the part of the client. Also, as mentioned above, the parallel download schemes rarely take into account incorporating modules that will address the security issues. In [6], the authors analyzed (through simulations) the impact of large-scale deployment of parallel downloading on the Internet as well for network dimensioning and content distribution service provisioning. They show that with

proper admission control and dimensioning, single-server downloading can perform just as well as parallel downloading, without the complexity and overhead incurred by the latter. The above observation forms the motivation for our work in this paper. Ours is the first novel approach to expedite file download in a sequential fashion by incorporating the idea of a secure session transfer protocol that can be run on thin clients, with just one TCP connection for the entire download process, and is also adaptive to the congestion in the Internet.

3 Design of the Session Transfer Protocol

The Session Transfer Protocol (STP) will run at the session layer on the top of TCP. To use STP at the application layer, the traditional FTP Protocol has to be modified to run on the top of STP. The modified FTP can be referred to as the STP-aware FTP and it needs to run on a separate port number. In other words, the STP-aware FTP would be an alternate to the standard FTP. If a client does not want to go through the file transfer that could potentially involve more than one server, then the client can use the standard FTP; if the client wants to use STP in order to get better QoS and be able to successfully transfer the files even in the presence of network congestion, then the client can use the STP-aware FTP. Figure 1 illustrates the TCP/IP protocol stack for the standard FTP and the STP-aware FTP.

There are three entities involved in the STP protocol: (i) STP Server Cluster – A group of servers, each located in different networks, one or more of which are involved in the file download session with a client. Note that, only a subset of the cluster might carry a specific file and this information resides in the STP Gateway Server. (ii) STP Client – A client machine that runs the STP protocol and is involved in downloading a file from the STP Server Cluster. (iii) STP Gateway Server – The public face of the STP Cluster. The STP Client first contact the STP Gateway Server to initiate the file downloading process. The STP Server Cluster and STP Gateway Server are organization-specific. There could be multiple STP Server Clusters and an appropriate STP Gateway Server (one for each organization) running in the Internet.

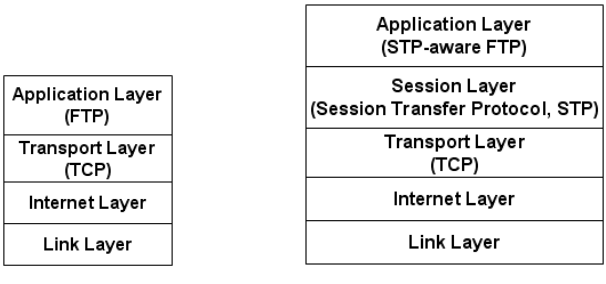

Fig. 1. TCP/IP Protocol Stack for the Traditional FTP and the STP-aware FTP

3.1 STP – Details

1. The STP Client initiates the download session by contacting the STP Gateway Server. The client passes the username and password to first get authenticated by the Gateway Server. Once authenticated, the client sends the path and the name of the file to download. We assume the file hierarchy for a particular user is maintained the same across all STP Servers. The Gateway server resolves the tuple <username, path> in its database and extracts the list of STP Servers that store the file. The STP Servers are ranked in the order of the number of hops from the client network.
2. The Gateway server creates a STP Ticket that contains the username, path of the file requested, filename, IP address of the client machine, the byte number in the file (set to 0) and the time of contact information. The time of contact information is included to avoid any replay attack. STP Tickets lose their validity beyond a certain time after their creation. All of the above information in the STP Ticket is encrypted using a secret key that is shared by all the STP Servers and the Gateway Server. Along with this information, the Gateway also includes the set of IP addresses of the STP Servers in the increasing order of the hop count from the client network. For security purposes, the IP address list of the candidate STP Servers is encrypted through a key that is derived (using a Key Derivation Function agreed upon by the user while creating an account at the Gateway Server) based on the user password. Figure 2 illustrates the contents of the STP Ticket along with the STP Server IP address list. We show only the payload portion of the Ticket message; we do not show the standard IP header (containing the STP Gateway IP address as the sender address and the STP Client address as the destination address) that is part of the message.

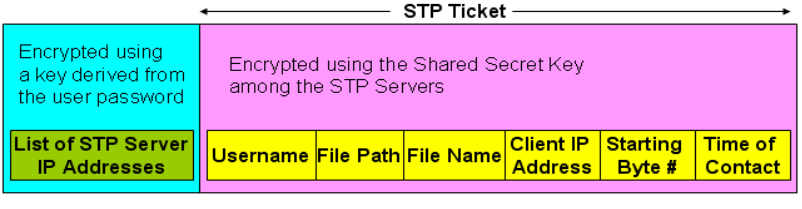

Fig. 2. Structure of STP Ticket along with the List of Server IP Addresses

3. The client decrypts STP Server List and pings the top three servers in the list by sending four short “Echo Request” messages to each of these servers. The client measures the Round Trip Time (RTT) of the “Echo Reply” ping messages. The STP Server that returns the Reply message at the earliest (i.e., incurred the lowest RTT) is selected. Ties are broken by the lowest hop count and other predefined criteria.

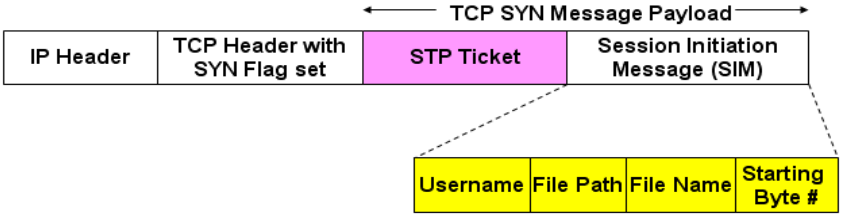

Fig. 3. TCP SYN Message with the Payload STP Ticket and SIM

4. The STP Client attempts to establish a TCP Session with the chosen STP Server and sends a TCP SYN message (structure shown in Figure 3) – the payload of which includes the STP Ticket and a Session Initiation Message (SIM) containing the username, path, filename and the byte number, starting from which the download is requested. The STP Server first decrypts the STP Ticket using the secret key shared among the servers in the STP Cluster as well as the Gateway Server. If the extracted contents of the Ticket matches with the username and file path (sent by the client) as well as the IP address of the client machine, then the STP Sever accepts the TCP connection request (sends a TCP SYN/ACK message) if it can allocate the required resources for the file download session. Otherwise, the STP Server sends a 'Connection Request Reject' message. Once the STP Server has accepted for the TCP session, the STP Client begins to download the contents of the requested file using TCP. In order to avoid any IP-spoofing triggered session transfers, we recommend the STP Client and STP Server to form an IPSec security association (SA) before establishing a TCP session. One of the pre-requisite steps for establishing an IPSec SA is to run an Internet Key protocol Exchange (IKE) session between the concerned Client and Server machines and exchange their public-key certificates. All subsequent communications, including the TCP session establishment messages, packets of the file being downloaded and the Transfer Session message – all of these could be encrypted at the sender using the public key of the receiver and decrypted at the receiver using its private key.
5. If the STP Server denies the TCP connection request, the STP Client includes the STP Server to the 'Overloaded List of STP Servers' and then tries to establish a TCP Session with the STP Server that responded with the next lowest RTT. If all the three first-choice STP Servers deny the connection request, the STP Client chooses the next three STP Servers in the list sent by the Gateway Server and pings them. This procedure is repeated until the STP Client manages to successfully find a STP Server; otherwise, the STP Client returns an error message to the user indicating that the file cannot be downloaded.
6. After receiving a packet in-order, the STP Client acknowledges for all the packets that have been received in-order and not acknowledged yet. The STP Server measures the RTT for the acknowledgment packets received from the STP Client. If the RTTs start increasing significantly for every acknowledgment received (the actual rate of increase of the RTT is an implementation issue), then the STP Server decides to handoff the session to another peer STP Server.

7. To handoff the session, the STP Server sends a 'Transfer Session' message to the STP Client and includes the sequence number of the last byte whose acknowledgment has been received by the Server and the position of this byte (i.e., the byte number) in the actual file being downloaded. The STP Server also updates the STP Ticket with the byte number that was last sent to the Client and acknowledged by the latter. The STP Server encrypts the updated STP Ticket using the secret key shared among all the servers in the STP cluster. The updated STP Ticket along with the Transfer Session message is sent to the STP Client.
8. After receiving the Transfer Session message, the STP Client confirms about the last byte number that was received in-order from the previous STP Server (which is now added to the Overloaded list). The STP Client now goes through the original Server List sent by the STP Gateway Server. Unlike the previous procedure adopted (i.e., to look for potential STP Servers in the increasing order of the number of hops), the STP Client randomly permutes the list and pings all the Servers in the Cluster, except those in the locally maintained Overloaded list.

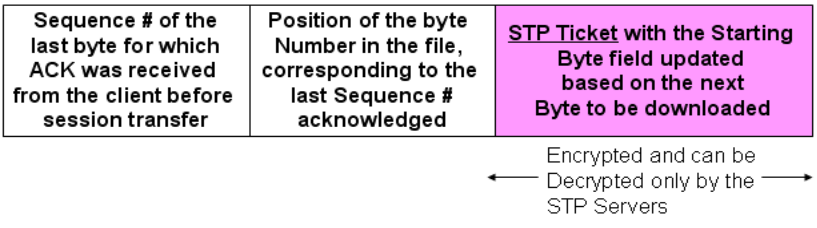

Fig. 4. Contents of the Transfer Session Message Sent by an STP Server

9. The STP Server that responds back with an "Echo Reply" at the earliest is chosen as the next Server to transfer the session. The STP Client attempts to open a TCP session with the new chosen server by sending a TCP SYN message – the payload of which includes the STP Ticket received from the previous STP Server as well as a 'Transfer Request' message containing those forwarded to the first STP Server: username, path, filename and the byte number (one more than the previous value), starting from which the download is requested.

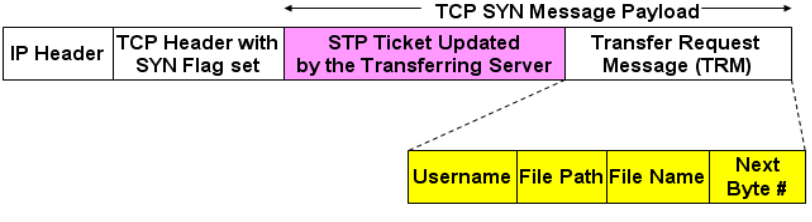

Fig. 5. TCP SYN Message with the Payload – Updated STP Ticket and Transfer Request Message

10. Once the newly chosen STP Server receives the STP Ticket along with the Transfer Request message, it decrypts the STP Ticket using the secret key for the STP Server cluster and compares the contents of the STP Ticket with those in the Transfer Request message. If everything matches and it is ready to allocate the required buffer space for this session and offer the requested window size, the new STP Server agrees to continue with the download session and sends a TCP SYN/ACK message; otherwise, it sends a Connection Request Reject message.
11. The STP Client adds the last chosen STP server that sent the Connection Request Reject message to the Overloaded list. Another STP Server that is not in the Overloaded list is contacted and this procedure is repeated until a new STP Server to transfer the session is found. If unsuccessful over the entire STP Server List, the STP Client quits and reports an error message to the user.
12. Once the new STP Server has accepted the TCP connection request and to continue with the transferred session, the STP Client begins to download the subsequent contents of the file. A secure-TCP session established on the top of IPSec is recommended.
13. After a while, if the new STP Server decides to handoff the file download session, then Steps 7 through 12 are again followed.

3.2 Qualitative Comparison with Standard FTP

FTP does not support session transfer during the middle of a file download. If a client or server experiences frequent timeouts and/or packet loss due to network congestion, the TCP session running as part of FTP has to be discontinued and a new TCP session has to be established. Nevertheless, we cannot be sure whether the new TCP session would be of any remedy to the network congestion problem as packets are more likely to be again routed through the same set of congested routers (and networks) as long as the server and client remain the same. STP handles the network congestion problem by initiating the transfer of a session to another server. This transfer is done in a secure fashion, through the encrypted session transfer ticket, in order to avoid the scenarios wherein an attacker initiates the transfer without the consent or knowledge of the actual server or the client. There could be some delay involved in transferring a session from one server to another server. However, the transfer delay is expected to be smaller enough to offset the delay incurred if the packets are continued to be sent on a congested route without any session transfer.

STP sincerely attempts to avoid session thrashing wherein a newly transferred session to a server *I* does not get immediately transferred to some other server *J*. Note that in Step 10, the STP Server receiving the Transfer Request message accepts the message only if it can allocate resources and offer the download service as requested. However, from a network congestion point of view, we cannot guarantee that session thrashing will be totally avoidable. As IP works on a per-packet basis, it is possible that after the session transfer is implemented, one or more networks on the route between the client and the server start to get congested and the session has to be again transferred to some other server within the set of clusters.

4 Conclusions and Future Work

The high-level contribution of this paper is the design of a secure Session Transfer Protocol (STP) that can be used even by thin clients to download a large file, distributed across several servers that constitute the STP Server Cluster. If there is an impending congestion on the path between a STP Client and the STP Server, the latter proactively initiates a session handoff by sending a Session Transfer message with the details on the last byte acknowledged and etc., updated in the STP Ticket. The STP Client contacts the other STP Servers in the list originally sent by the STP Gateway (during the authentication phase) and chooses the best alternate STP Server that agrees to continue with the download session. The user working at the STP Client is totally unaware of this session transfer process among the servers in the STP Cluster. The entire session transfer will occur in a secure manner with no scope for any denial of service or spoofing attacks, if the TCP session is run on the top of IPSec. Throughout the download session, an STP Client is required to maintain only one TCP connection – a feature that suits thin clients, unlike the protocols for parallel download that require a client to simultaneously run/maintain multiple TCP connections. Compared to the parallel and peer-to-peer download schemes, STP can be the preferred choice for streaming applications, of course with some jitter experienced during session transfer. Faster the session transfer, smaller is the delay. Our strategy to let the STP Client randomly choose STP Servers (from a list of putative servers) to contact for session transfer helps to minimize the session transfer delay. Also, because of sequential download, data packets of the file are highly likely to arrive in-order at the client. In the near future, we plan to implement STP, first as a prototype in a laboratory scale, simulating with client-server programs and then implement in a larger network with traffic actually sent over the Internet. Through simulations, we plan to compare the performance of STP with that of the P-FTP and DPA parallel download protocols as well as the BitTorrent peer-to-peer protocol.

Acknowledgments. The authors made equal contributions. The work of Natarajan Meghanathan leading to this paper has been partly funded through the U. S. National Science Foundation (NSF) CCLI/TUES grant (DUE-0941959) on “Incorporating Systems Security and Software Security in Senior Projects.” The views and conclusions contained in this document are those of the authors and do not represent the official policies, either expressed or implied, of the funding agency.

References

1. Sohail, S., Jha, S.K., Kanhere, S.S.: QoS Driven Parallelization of Resources to Reduce File Download Delay. IEEE Transactions on Parallel and Distributed Systems 17(10), 1204–1215 (2006)
2. Rodriguez, P., Biersack, E.W.: Dynamic Parallel Access to Replicated Content in the Internet. IEEE Transactions on Networking 10(4), 455–465 (2002)

3. Karrer, R.P., Knightly, E.W.: TCP-PARIS: A Parallel Download Protocol for Replicas. In: The 10th International Workshop on Web Content Caching and Distribution, Sophia Antipolis, France, pp. 15–25 (2005)
4. Brock, M., Goscinski, A.: A Parallel Download Protocol for Internet-based Distributed Systems. In: International Conference on Internet Computing, Las Vegas, pp. 3–9 (2008)
5. Chang, R.-S., Guo, M.-H., Lin, H.-C.: A Multiple Parallel Download Scheme with Server Throughput and Client Bandwidth Considerations for Data Grids. Future Generation Computer Systems 24(8), 798–805 (2008)
6. Koo, S.G.M., Rosenberg, C., Xu, D.: Analysis of Parallel Downloading for Large File Distribution. In: The 9th Workshop on Future Trends of Distributed Computing. IEEE, San Juan (2003)
7. Neglia, G., Reina, G., Zhang, H., Towsley, D., Venkataramani, A., Danaher, J.: Availability in BitTorrent Systems. In: International Conference on Computer Communications, pp. 2216–2224. IEEE, Anchorage (2007)
8. Measche, D.S., Rocha, A.A.A., Li, B., Towsley, D., Venkataramani, A.: Content Availability and Bundling in Swarming Systems. In: The 5th International Conference on Emerging Networking Experiments and Technologies, pp. 121–132. ACM (2009)
9. BitTorrent, `http://www.bittorrent.org/beps/bep_0003.html` (last accessed: July 26, 2011)
10. Peterson, L.L., Davie, B.S.: Computer Networks: A Systems Approach, 5th edn. Morgan Kaufmann (2011)