Paxson
Spring 2017

CS 161
Computer Security

Discussion 9

Week of April 3, 2017

Question 1 ***Lists and Trees of Hashes*** **(20 min)**

BitTorrent splits large files into small file chunks which are then transmitted between peers in such a way that each peer eventually ends up with the whole file. Commonly, chunks are of size 2^8 KiB = 256 KiB.

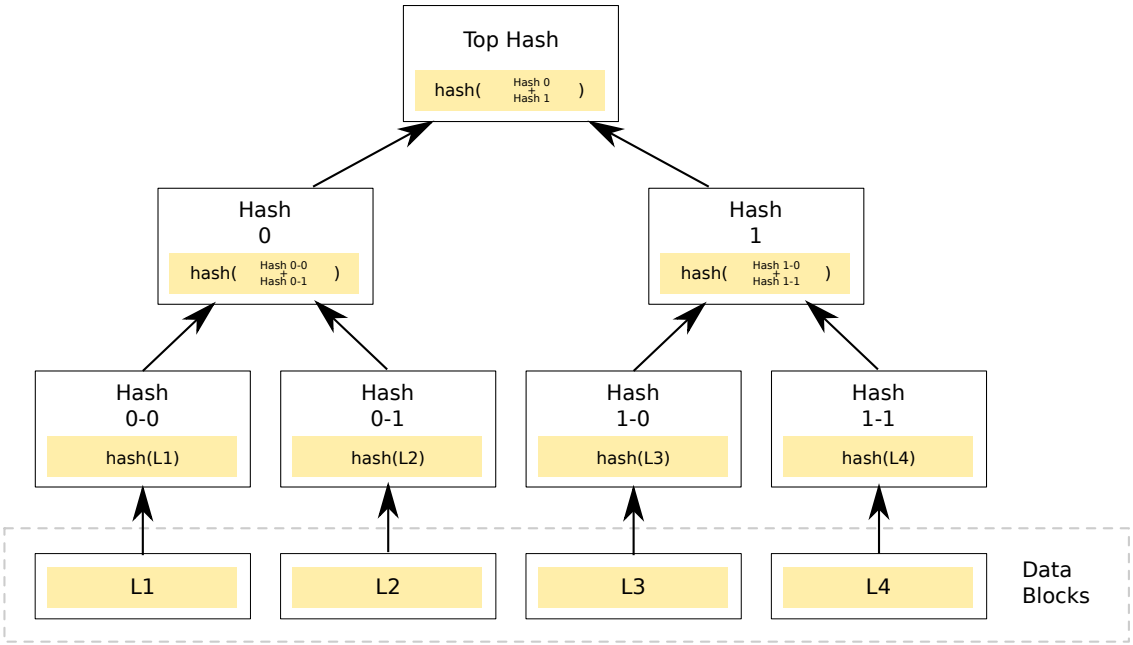
Because you cannot trust peers, you have to verify each chunk as you download them from a peer before you start providing them to other peers. Furthermore, you want to be able to do this as soon as possible and not wait for the whole file to be downloaded. You also want to be able to know which part of the file got potentially corrupted so that you do not have to re-download the whole file.

To achieve the above properties, BitTorrent uses a Torrent file. The file contains information describing the file (or files) to be transmitted, and their chunks. You must obtain this file from a trusted source.

(a) Initially, a Torrent file contained a list of SHA-1 hashes for each chunk. How large is such a list for a 10 GiB large file, if one SHA-1 hash takes 160 bits?

(b) One way to make Torrent files smaller is to instead store only a hash of the hash list (top hash, or root hash) in the file and retrieve the hash list itself from a peer. Why would we want to make a Torrent file smaller? What is a downside of this approach?

(c) One approach to address the issue of the size of the hash list is to split it into chunks. However, you would then need a hash list of those chunks. A better approach is to generalize this idea and use a data structure called a hash tree or Merkle tree:



Now you do not need the whole hash list in advance to verify one chunk. Instead, you can ask your peer to provide you with some hashes along with the chunk just received.

Suppose you just received chunk L2 from a peer. Which and how many hashes do you need to verify if you correctly received chunk L2? How would you generalize which and how many hashes you need for each chunk?

(d) You do not trust your peer with the contents of a chunk, but why you can use hashes provided by the peer for verifying the file?

Question 2 ***Circumventing Network Policy Controls*** **(15 min)**

You are stuck at the Chicago O'Hare airport for at least 7 more hours, and the airport does not have free WiFi. How boring! Since you are unwilling to pay the offensive fees the hotspot provider demands to access the Internet, you start to explore what you can do with the limited connectivity. Quickly you find out that all connections to Internet hosts are blocked. You also discover that the DNS server of the hotspot answers queries for any hostname in the Internet.

As an exceptionally studious and forward-looking CS 161 student, you were prepared for this scenario of Internet deprivation and in fact already set up your own DNS infrastructure prior to leaving on your trip. How is it possible to get free WiFi access in this scenario? What needed to happen prior to the trip? Sketch a diagram and describe any protocol you come up with.

Question 3 ***Project 2 Rollback Prevention*** **(15 min)**

In Project 2, you were not required to prevent rollback attacks where the server reverted the state of a file to a previous value. In this problem you will design a scheme to prevent *partial rollback* attacks. (A *partial rollback* is one where the server rolls back the contents of one file, but not another.) Clients in your scheme can cooperate to work together against a malicious server, but may not keep state on the client.

(a) The simplest scheme has each user write to a shared *hash chain* that exists on the server. After an upload operation, the user who performed that operation adds a new entry to the chain and uploads it to the storage server. What should each entry contain so that users can verify the state of the server by examining the hash chain?

(b) Can we create a more secure scheme if we allow clients to maintain state? What additional guarantees can we provide?