ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 4, Issue 2, pp: (9-14), Month: April - June 2016, Available at: www.researchpublish.com

Video Streaming: A Case Study

[1]Abin Sebastian, [2]Vipin Mohan, [3]Sneha Thankachan

[1,2] IVth year student MBCCET Peermade, Idukki, Kerala, India
[3]Asst. Professor Dept. of CSE, MBCCET Peermade, Idukki, Kerala, India

***Abstract*: Video streaming is the sending of contents in a compressed form to the internet and it is displayed by a viewer in real time, with video streaming a web user does not have to wait to download a file and to play it. instead of this the media is sent in a continues stream of data and played as it arrives, nowadays tremendous studies are conducted according to the video streaming such as Gnutella, Bit torrent, Bit Tyrant, peer to peer streaming, Random push etc. This paper also compares each streaming techniques.**

***Keywords:* Video streaming, Random push, Gnutella.**

I. INTRODUCTION

A great comprehensive analysis of user traffic on Gnutella [1] shows a significant quantity of free riding found in the system. By testing messages on the Gnutella network over a 24-hour period, It established that practically 70% of Gnutella users share no documents, and practically 50% of all responses are delivered by the top 1% of sharing hosts. Furthermore, It found out that free riding is given away evenly between domains, in order that no-one group contributes substantially a lot more than others, and that peers that volunteer to share files are not really necessarily those who have got desirable ones. It claim that free riding brings about degradation in the system. The achievements of peer-to-peer contribution for live multicast loading are determined by their ability to maintain low delays and a low ratio details loss end-to-end. However, info distribution over a contribution consisting of unreliable friends is definitely inherently subject to disturbances. Resilience is so inevitably the requirement intended for peer-to peer live-streaming architectures [2]. Found in this article, it present a survey of the media division methods, contribution structures, and error-control alternatives proposed for peer-to-peer live streaming. It discuss the tradeoff between power and overhead and believe that efficient architectures can always be defined only through complete overall performance analysis.

A fundamental problem with many peer-to-peer systems is the tendency for users to “free ride”—to consume resources without contributing to the system. The popular file distribution tool Bit Torrent was explicitly designed to address this problem, using a tit-for-tat reciprocity strategy to provide positive incentives for nodes to contribute resources to the swarm [3]. While Bit Torrent has been extremely successful, it show that its incentive mechanism is not robust to strategic clients. Through performance modeling parameterized by real world traces, it demonstrates that all peers contribute resources that do not directly improve their performance. It use these results to drive the design and implementation of BitTyrant [3], a strategic BitTorrent client that provides a median 70% performance gain for a 1 Mbit client on live Internet swarms.

The dynamics of peer participation, or churn [4], are an inherent property of Peer-to-Peer (P2P) systems and critical for design and evaluation. Accurately characterizing churn re- quires precise and unbiased information about the arrival and departure of peers, which is challenging to acquire? Prior studies show that peer participation is highly dynamic but with conflicting characteristics. Therefore, churn remains poorly understood, despite its significance. The field of peer-to-peer reputation systems has exploded in the last few years. Our goal is to organize existing ideas and work to facilitate system design. It present a taxonomy of reputation system components, their properties, and discuss how user behavior and technical constraints can conflict. In our discussion, it describe research that examples compromises made to deliver a useable, implementable system.

II. FREE RIDING ON GNUTELLA

The sudden appearance of new forms of network applications such as Gnutella [Gn00a] and Free Net [Fr00], holds promise for the emergence of fully distributed information sharing systems [1]. These systems, inspired by Napster [Na00], will allow users worldwide access and provision of information while enjoying a level of privacy not possible in the present client-server architecture of the web. While a lot of attention has been focused on the issue of free access to music and the violation of copyright laws through these systems, there remains an additional problem of securing enough cooperation in such large and anonymous systems so they become truly useful. Since users are not monitored as to who makes their files available to the rest of the network (produce) or downloads remote files (consume), nor are statistics maintained, the possibility exist that as the user community in such networks gets large, users will stop producing and only consume. This free riding behavior is the result of a social dilemma that all users of such systems confront, even though they may not be aware of its existence. In a general social dilemma, a group of people attempts to utilize a common good in the absence of central authority. In the case of a system like Gnutella, one common good is the provision of a very large library of files, music and other documents to the user community. Another might be the shared bandwidth in the system. The dilemma for each individual is then to either contribute to the common good, or to shirk and free ride on the work of others.

The second problem caused by free riding is to create vulnerabilities for a system in which there is risk to individuals. If only a few individuals contribute to the public good, these few peers effectively act as centralized server. Users in such an environment thus become vulnerable to lawsuits, denial of service attacks, and potential loss of privacy. This is relevant in light of the fact that systems such as Gnutella, Napster, and Free Net are depicted as a means for

Individuals to rally around certain community goals and to "hide" among others with the same goals. These may include providing a forum for free speech, changing copyright laws, and providing privacy to individuals.

A. Gnutella:

People who wish to use the Gnutella network will download [Gn00a] or develop [Gn00b] an application that adheres to the Gnutella protocol. This application acts as either a client (a consumer of information) or a server (a supplier of information), as well as a high-level network, connecting and routing information between clients and servers. Each instance of an application is called a peer

Gnutella [1] boasts a number of features that make it attractive to certain users. For example, Gnutella provides for anonymity by masking the identity of the peer that generated a query. Additionally, Gnutella provides the mechanism by which ad hoc networks can be formed without central control. Since there are no central servers in the Gnutella network, in order to join the system a user initially connects to one of several known hosts that are almost always available (although these generally do not provide shared files). These hosts then forward the IP and port address information to other Gnutella peers.

Once attached to the network, peers interact with each other by means of messages. Peers will create and initiate a broadcast of messages as well as *rebroadcasting* others (receiving and transmitting to neighbors). The messages allowed in the network are:

III. RESILIENCE IN LIVE PEER-TO-PEER STREAMING

The success of peer-to-peer overlays for live multicast streaming depends on their ability to maintain low delays and a low ratio of information loss end-to-end [2]. However, data distribution over an overlay consisting of unreliable peers is inherently subject to disturbances. Resilience is thus inevitably a key requirement for peer-to peer [6] live-streaming architectures. Live multicast of streaming media over the Internet presents several challenges. First of all, high bit rate streams must be delivered to a potentially large population of users. This requires either high transmission capacity at the streaming server, if using the traditional client-server approach, or multicast support in the network. The peer to- peer approach aims to alleviate these demands by utilizing the upload bandwidth of the participating peers to distribute the media stream. However, peer-to-peer streaming faces several challenges. The number of peers participating in the overlay may change rapidly; the streams must be transmitted with end-to-end delays that are acceptable for live applications, in the range of a couple of tens of seconds; and to keep the perceived media quality acceptable, the packet loss rate must be low.

A. System requirement and data transmission methods:

For peer-to-peer streaming to be successful, it must provide low end-to-end packet loss rate, delay, and delay jitter, similarly to point-to-point live streaming [2]. To satisfy these requirements, the transmission bandwidths must be utilized effectively in bandwidth-scarce environments, and the system must adapt to changes in network conditions, for example, to increasing congestion on some of the peer-to-peer transmission paths. The solution must be resilient to churn, that is, when peer nodes join and leave the overlay during the streaming session, the cohesion of the overlay structure must be maintained, and the information loss due to node departures must be minimized. The architecture proposed for peer-to-peer

Streaming generally falls into one of two categories: push based or pull-based. Solutions in both categories utilize multi path transmission to ensure graceful quality degradation in dynamic overlays. With multi-path transmission, parts of the stream reach the peers through independent overlay paths, and consequently a large part of the streaming data can be received, even if some of the peers stop forwarding. The push method follows the traditional approach of IP multicast. The streaming data is forwarded along multiple disjoint transmission trees with the streaming server as the root [2]. The pull method (also called swarming) follows the approach of off-line peer-to-peer content distribution and aims at efficient streaming in highly dynamic environments. There is no global structure maintained, and parent-child relations are determined locally on the fly as the overlay membership changes.

B. Streaming with the pull method:

The overlay constructed for pull-based streaming is often referred to as directed mesh, where directed edges represent potential parent-child relations [2]. The mesh is reconstructed periodically as peers search for new parents after the transmission of each data block. Overlay membership information is managed by a central entity or spread by gossiping with message exchange between child and parent peers. Overlays employing the pull method are dynamic and follow the changes of membership and network conditions in the overlay easily, compared to the push-based approach, where a tree structure must be maintained. The cost of this simplified overlay management is twofold. First, the data transmission requires the frequent exchange of control messages. Second, the pulling process introduces additional delay at each overlay hop: first the list of available packets is transmitted from the parent to the child node, then the child node sends a request message to the parent, and finally, the packets are transmitted.

IV. ROBUSTNESS IN BIT TORRENT

A fundamental problem with many peer-to-peer systems is the tendency of users to “free ride” consumes resources without contributing to the system [3]. File distribution tool Bit Torrent was explicitly designed to address this problem, using a tit-for-tat reciprocity strategy to provide positive incentives for nodes to contribute resources to the swarm. While Bit Torrent has been extremely successful, its incentive mechanism is not robust to strategic clients. These results to drive the design and implementation of BitTyrant [3].

A. BitTorrent:

BitTorrent, a popular file distribution tool based on a swarming protocol, proposed a tit-for-tat (TFT) strategy aimed at incenting peers to contribute resources to the system and discouraging free riders. Bit Torrent focuses on bulk data transfer [3]. All users in a particular swarm are interested in obtaining the same file or set of files. In order to initially connect to a swarm, peers download a metadata file, called a torrent, from a content provider, usually via a normal HTTP request. This metadata specifies the name and size of the file to be downloaded, as well as SHA-1 fingerprints of the data blocks (typically 64–512 KB) that comprise the content to be downloaded. These fingerprints are used to verify data integrity. The metadata file also specifies the address of a tracker server for the torrent, which coordinates interactions between peers participating in the swarm. Peers [8] contact the tracker upon startup and departure as well as periodically as the download progresses, usually with a frequency of 15 minutes. The tracker maintains a list of currently active peers and delivers a random subset of these to clients, upon request.

B. Bit Tyrant:

Altruism in Bit Torrent serves as a kind of progressive tax. As contribution increases, performance improves, but not in direct proportion. Here describes the design and implementation of Bit Tyrant, a client optimized for strategic users. This chose to base BitTyrant [3] on the Azureus client in an attempt to foster adoption, as Azureus is the most popular client in

our traces. To evaluate Bit Tyrant, here explore the performance improvement possible for a single strategic peer in synthetic and current real world swarms as well as the behavior of Bit Tyrant when used by all participants in synthetic swarms. Evaluating altruism in Bit Torrent experimentally and at scale is challenging. Traditional wide-area test beds such as Planet Lab do not exhibit the highly skewed bandwidth distribution. Here observe in measurements, a crucial factor in determining the amount of altruism. Alternatively, fully configurable local network test beds such as Emu lab are limited in scale and do not incorporate the myriad of performance events typical of operation in the wide-area. Further, Bit Torrent implementations are diverse, as shown in Table 1.

To address these issues, it perform two separate evaluations. First, evaluate Bit Tyrant on real swarms drawn from popular aggregation sites to measure real world performance for a single strategic client. This provides a concrete measure of the performance gains a user can achieve today. To provide more insight into how Bit- Tyrant functions, It then revisit these results on Planet- Lab where evaluate sensitivity to various upload rates and evaluate what would happen if Bit Tyrant is universally deployed.

Table 1: Bit Torrent implementation usage as drawn from measurement data

Implementation	*Percentage share*
Azureus	47%
BitComet	20%
Mtorrent	15%
BitLord	6%
Unknown	3%
Reference	2%
Remaining	7%

V. CHURN IN PEER TO PEER NETWORKS

The dynamics of peer participation, or churn [4], are an inherent property of Peer-to-Peer (P2P) systems and critical for design and evaluation. Accurately characterizing churn requires precise and unbiased information about the arrival and departure of peers, which is challenging to acquire. Prior studies show that peer participation is highly dynamic but with conflicting characteristics. Therefore, churn remains poorly understood, despite its significance.

Here identify several common pitfalls that lead to measurement error and carefully address these difficulties and present a detailed study using three widely- deployed P2P systems: an unstructured file-sharing system (Gnutella) [1], a content-distribution system (Bit Torrent), and a Distributed Hash Table (Kad). This analysis reveals several properties of churn: (i) overall dynamics are surprisingly similar across different systems, (ii) session lengths are not exponential, (iii) a large portion of active peers are highly stable while the remaining peers turn over quickly, and (iv) peer session lengths across consecutive appearances are correlated. In summary, this paper advances our understanding of churn by improving accuracy, comparing different P2P [6] file sharing/distribution systems, and exploring new aspects of churn.

During recent years, the Internet has witnessed a significant increase in the popularity of Peer-to-Peer (P2P) applications ranging from file-sharing (e.g., Gnutella [1] and Fast Track) to conferencing (e.g., End System Multicast) and content distribution (e.g., BitTorrent [3]). A peer joins the system when a user starts the application, contributes some resources while making use of the resources provided by others, and leaves the system when the user exits the application. This define one such join-participate-leave cycle as a session. The independent arrival and departure by thousands—or millions—of peers creates the collective effect it call churn.

VI. RANDOM PUSH WITH RANDOM NETWORK CODING

In information theory, it has been shown that network coding can effectively improve the throughput of multicast communication sessions in directed acyclic graphs. More practically, random network coding [5] is also instrumental towards improving the downloading performance in Bit Torrent-like peer to- peer content distribution sessions. Live peer-

to-peer streaming, however, poses unique challenges to the use of network coding, due to its strict timing and bandwidth constraints. In this paper, it revisit the complete spectrum in the design space of live peer-to-peer streaming protocols, with a sole objective of taking full advantage of random network coding. This present R2 [5], our new streaming algorithm designed from scratch to incorporate random network coding with a randomized push algorithm. R2 is designed to improve the performance of live streaming in terms of initial buffering delays, resilience to peer dynamics, as well as reduced bandwidth costs on dedicated streaming servers, all of which are beyond the basic requirement of stable streaming playback. On an experimental test bed consisting of dozens of dual-CPU cluster servers, It thoroughly evaluate R2 with an actual implementation, real network traffic, and emulated peer upload capacities, in comparisons with a typical live streaming protocol (both without and with network coding), representing the current state-of-the-art in real-world streaming applications.

A. Random network coding:

Random network coding serves as the cornerstone of R2, and is instrumental towards most of the advantages of R2. In traditional P2P streaming protocols, the live stream to be served is divided into segments, such that they can be better exchanged among peers. In R2 [5], each segment is further divided into n blocks [b1, b2, . . . , bn], each bi has a fixed number of bytes k (referred to as the block size). If the segment duration (for example, four seconds) and the streaming rate is predetermined, the block size k can be directly computed from n.

B. Random push:

In R2, whenever a seed sends a coded block to a downstream peer, it needs to use the "random push" [5] mechanism by randomly selecting a segment to code, among segments that the downstream peer has not yet completely received. A seed randomize such a segment selection process for each outgoing coded block as Naturally, an important concern at the downstream peer is that it should expedite the downloading process of "urgent" but missing segments, i.e., those missing segments that are close to their playback time. This range of urgent segments may be seconds after the playback point, and is referred to as the priority region, as shown in Fig. 1. Since there are no explicit requests made by the downstream peer (no "pull" required), seeds should give strict priority to the segments within the priority region. In this randomized segment selection, it stipulate that a seed should randomize within the priority region using a uniform distribution, whenever segments in this region are still missing in the downstream peer. From the viewpoint of a receiving peer, as playback progresses, if a few missing segments eventually fall into the priority region, their urgency guarantees that all of its seeds will serve these segments. If the receiving peer has sufficient download bandwidth, it should be able to completely receive these missing segments before playback.

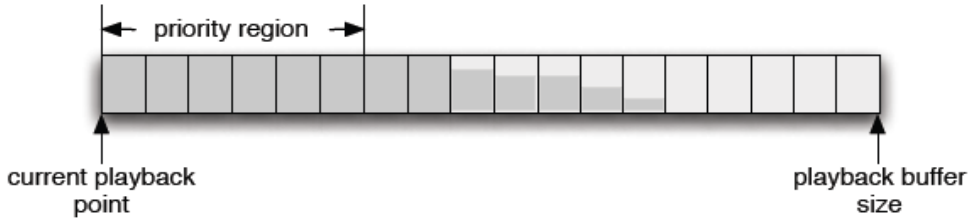

Fig.1. The playback buffer in R^2

VII. SQUIRREL

A decentralized, peer-to-peer web cache called Squirrel [7]. The key idea is to enable web browsers on desktop machines to share their local caches, to form an efficient and scalable web cache, without the need for dedicated hardware and the associated administrative cost. This propose and evaluate decentralized web caching algorithms for Squirrel[7], and discover that it exhibits performance comparable to a centralized web cache in terms of hit ratio, bandwidth usage and latency. It also achieves the benefits of decentralization, such as being scalable, self-organizing and resilient to node failures, while imposing low overhead on the participating nodes. The key idea in Squirrel is to facilitate mutual sharing of web objects among client nodes. Currently, web browsers on every node maintain a local cache of web objects recently accessed by the browser. Squirrel enables these nodes to export their local caches to other nodes in the corporate network, thus synthesizing a large shared virtual web cache. Each node then performs both web browsing and web caching.

VIII. CONCLUSION

In this paper we have studied about peer to peer video streaming, such as Bit torrent, Bit Tyrant, Churn, Gnutella etc.

These topics describe the quality and performance of each streaming techniques. We can conclude that each methods has its own advantages and disadvantages.

REFERENCES

[1] E. Adar and B. A. Huberman, "Free riding on Gnutella," First Monday, vol. 5, no. 10, October 2000. [Online]. Available: citeseer.ist.psu.edu/article/adar00free.html

[2] V. Fodor and G. D´an, "Resilience in live peer-to-peer streaming," IEEE Communications Magazine, vol. 45, no. 6, pp. 116–123, Jun. 2007.

[3] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "Do incentives build robustness in BitTorrent?" in Proceedings of 4th USENIX Symposium on Networked Systems Design & Implementation (NSDI 2007). Cambridge, MA: USENIX, April 2007.

[4] D. Stutzbach and R. Rejaie, "Understanding churn in peer-to-peer networks," in Proceedings of the 6th ACM SIGCOMM conference on Internet measurement. Rio de Janeriro, Brazil: SIGCOMM, 2006, pp. 189–202.

[5] M. Wang and B. Li, "R2: Random push with random network coding in live peer-to-peer streaming," IEEE Journal on Selected Areas in Communications, vol. 25, no. 9, pp. 1655–1666, December 2007.

[6] S. Marti and H. Garcia-molina, "Taxonomy of trust: Categorizing p2p reputation systems," Computer Networks, vol. 50, pp. 472–484, 2006.

[7] S. Iyer, A. Rowstron, and P. Druschel, "Squirrel: A decentralized peer-to-peer web cache," in 12th ACM Symposium on Principles of Distributed Computing (PODC 2002), Jul. 2002, pp. 1–10.

[8] Y. Yue, C. Lin, and Z. Tan, "Analyzing the performance and fairness of bittorrent-like networks using a general fluid model," Computer Communications, vol. 29, no. 18, pp. 3946 – 3956, 2006