# Live Streaming With Receiver-Based Peer-Division Multiplexing

K.Lakshmi Devi[1] and Ravipati.Venkat[2]
[1]*PG Student, GDMM College of Engineering, Nandigama, India*
[2] HOD, GDMM College of Engineering, Nandigama, India

*Abstract*— **A number of commercial peer-to-peer systems for live streaming have been introduced in recent years. The behavior of these popular systems has been extensively studied in several measurement papers. Due to the proprietary nature of these commercial systems, however, these studies have to rely on a "black-box" approach, where packet traces are collected from a single or a limited number of measurement points, to infer various properties of traffic on the control and data planes. Although such studies are useful to compare different systems from end-user's perspective, it is difficult to intuitively understand the observed properties without fully reverse-engineering the underlying systems. In this paper we describe the network architecture of Zattoo, one of the largest production live streaming providers in Europe at the time of writing, and present a large-scale measurement study of Zattoo using data collected by the provider. To highlight, we found that even when the Zattoo system was heavily loaded with as high as 20,000 concurrent users on a single overlay, the median channel join delay remained less than 2 to 5 seconds, and that, for a majority of users, the streamed signal lags over-the-air broadcast signal by no more than 3 seconds.**

***Index Terms*—** Live streaming, network architecture, peer- to-peer (P2P) system.

## I. INTRODUCTION

**T**here is an emerging market for IPTV. Numerous commercial systems now offer services over the Internet that is similar to traditional over-the-air, cable, or satellite TV. Live television, time-shifted programming, and content-on-demand are all presently available over the Internet. Increased broadband speed, growth of broadband subscription base, and improved video compression technologies have contributed to the emergence of these IPTV services. To draw a distinction between three uses of peer-to-peer (P2P) networks: delay-tolerant file download of archival material, delay-sensitive progressive download (or streaming) of archival material, and real-time live streaming. In the first case, the completion of download is elastic, depending on available bandwidth in the P2P network. The application buffer receives data as it trickles in and informs the user upon the completion of download. The user can then start playing back the file for viewing in the case of a video file. Bit torrent and variants are examples of delay-tolerant file download systems. In the second case, video playback starts as soon as the application assesses it has sufficient data buffered that, given the estimated download rate and the playback rate, it will not deplete the buffer before the end of file. If this assessment is wrong, the application would have to either pause playback or rebuffed or slow down playback. While users would like playback to start as soon as possible, the application has some degree of freedom in trading off playback start time against estimated network capacity. Most video-on-demand systems are examples of delay-sensitive progressive-download application. The third case, real-time live streaming has the most stringent delay requirement. While progressive download may tolerate initial buffering of tens of seconds or even minutes, live streaming generally cannot tolerate more than a few seconds of buffering. Taking into account the delay introduced by signal ingest and encoding, and network transmission and propagation, the live streaming system can introduce only a few seconds of buffering time end-to-end and still be considered "live".

The Zattoo delivery network architecture is shown in figure 1. The Zattoo peer-to-peer live streaming system was a free-to-use network serving over 3 million registered users in eight European countries at the time of study, with a maximum of over 60 000 concurrent users on a single channel. The system delivers live streams using a receiver-based, peer-division multiplexing scheme. To ensure real-time performance when peer uplink capacity is below requirement, Zattoo subsidizes the network's bandwidth requirement, as described in. After delving into Zattoo's architecture in detail, large-scale measurements collected during the live broadcast of the UEFA European Football Championship, one of the most popular one-time events in Europe, in June 2008. During the course of that month, Zattoo served more than 35 million sessions to more than 1 million distinct users. Drawing from these measurements, it will report on the operational scalability of Zattoo's live streaming system along several key issues.

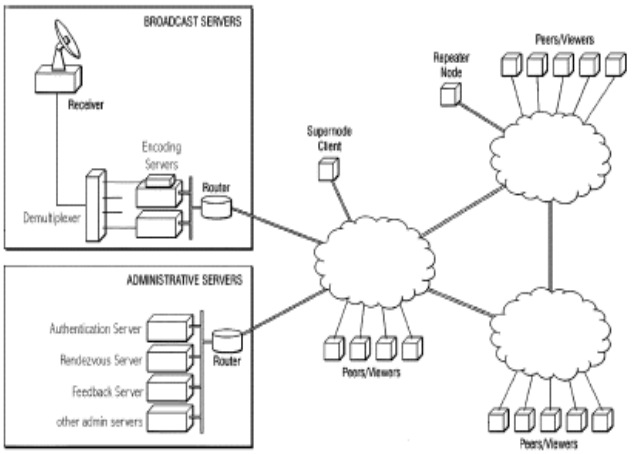


Fig 1 Zattoo Delivery Network Architecture

## II. OVERVIEW OF PPLIVE

### 2.1 Cool streaming: Design, Theory, and Practice:

**T**here have been significant studies and numerous technical innovations for live video streaming, in which the recent development has been focusing on application layer multicast or overlay multicast, in which the key idea is to push multicast functionalities to the edges of networks. It will argue that the native IP multicast encountered practically unsolvable problems and the adoption of some forms of end application multicast to bypass the need from infrastructure support has been proposed. In a typical overlay multicast architecture, end systems self-organize themselves into an overlay topology and data forwarding follows the links of the overlay. On the other hand, Cool streaming, a *data-driven* scheme, is a completely different approach. The cool streaming system diagram is shown in figure 2.The key novelties are

1) Peers gossip with one another for content availability information and

2) Peers use a swarm-like technique, somewhat similar to the technique used in Bit Torrent, for content delivery. To make a distinction in this paper by referring this as a peer-to-peer (P2P) live video streaming system, in which there is no explicit overlay topology construction. This was referred to as a treeless approach or swarming in. Refer to the other approaches as overlay multicast, given the explicit construction of multicast tree(s).

Cool streaming represented one of the earliest large-scale P2P video streaming experiments , which has been widely referenced in the community as the benchmark (Google entries tops 300 000) that a P2P-based live streaming system has the potential to scale. Since its first release, while keeping the random partner selection, we have enhanced the system in nearly all aspects, specifically

1) The initial system adopted a simple pull based scheme for content delivery based on content availability information exchange using buffer-map. This incurs per block overhead, and more importantly, it results in a longer delay in retrieving the video content. We have now implemented a hybrid pull and push mechanism, in which the video content is pushed by a parent node to a child node except for the first block. This lowers the overhead associated with each video block transmission, reduces the initial delay and increases the video playback quality;

2) A novel multiple sub stream scheme is implemented, which essentially enables multisource and multipath delivery for video streams. Observed from the results, this not only enhances the video playback quality and but also improves the effectiveness against system dynamics;

3) The gossip protocol was enhanced to handle the push function;

4) The buffer management and scheduling schemes are redesigned to deal with the dissemination of multiple sub streams. There are many issues relevant to the design of live video streaming systems such as system dynamics, heterogeneity, churn, network congestion, stream synchronization, multipath routing, topology stability and convergence. There have been investigations on optimal resource allocation, scaling factors, incentive-based mechanism, fine-granularity control, priority based multicast, integration with network encoding technique. However, there seems to be several misconceptions in the most basic understanding of a live video streaming system.

This paper takes a different approach by contributing to this basic understanding, we will:

1) Describe the key issues in a real working system;

2) Closely examine a set of existing practical problems and how they could be handled in a real system;

3) Focus the discussions on system dynamics and how it affects the overall performance. Recall one of the fundamental principles in the Internet is the simplicity in its core functionalities. Keeping this simplicity in mind, our discussions in this paper focus on a reasonably scale no optimal working system. The purpose of this paper is to demonstrate concretely how a working system resolves some of these issues and a set of realistic engineering problems that need to be addressed. We leverage a large set of traces

We recently obtained from a cool streaming experiment conducted on Oct 26, 2006, and attempt to develop some theoretical basis to provide the intuitions and our understanding on how and why such a system works. Specifically,

1) Based on a simple topology model, we illustrate how random peer selection can lead to topology convergence;

2) Using a set of real traces and analysis, we quantitatively provide the insights on how the buffering technique can resolve the problems associated with dynamics and heterogeneity;
3) We show how sub stream and path diversity can help to alleviate the impact from network congestion and peer churns;
4) we discuss the scalability and limitations of a P2P based live video streaming system. The existing works on P2P live video streaming can generally be classified into two categories: overlay multicast and data driven approaches. In an overlay multicast approach, the key idea is to build a multicast tree at the application end. The end system multicast is one of the first protocols [2], in which it examines the key difficulties encountered in the native IP multicast and proposes a new mechanism by moving the multicast functionalities to the edge of the networks. A single-tree overlay suffers two drawbacks:
1) unfair contributions in that leaf nodes do not contribute upload capacity, which leads to poor resource utilization;
2) Given the dynamic nature of nodes, the departure or failure of a high-level node can cause significant disruption and requires the reconstruction of the overlay topology.

The multi tree approach has been proposed to cope with this problem. This usually requires that the source encodes a video stream into multiple sub streams and each sub stream is distributed over a separate overlay tree. These result in two improvements,
1) Better resilience against churn and
2) fair bandwidth utilization from all peer nodes. However, multi tree based systems demand the use of special multi rate or/and multilayer encoding algorithms, which has not been demonstrated feasible in real systems over the Internet. Further, those systems still have to deal with the reconstruction of the multi trees in the presence of network dynamics and potentially restrictive contributions from leaf nodes. Perhaps most significantly, there has been no large scale real systems demonstrated using such an approach.

There have been several recent works, such as Chunky spread, which focused on the efficiency associated with multi tree construction and also explored the simplicity and scalability adopted from unstructured networks. Specifically, a randomized multi tree was constructed to spread the slices of video stream, and the topology could be improved iteratively by swapping parents with respect to load and delay measurement. Rae *et al.* evaluated the multi tree framework and proposed a new concept called *contribution awareness* as an incentive to enable better contribution from peer nodes. Reza *et al.* proposed to use the swarming technique to enable leaf nodes to contribute in uploading capacity. Within the framework of P2P live streaming, there are several fundamental issues that need to be resolved such as how an overlay topology evolves under a random peer selection scheme, under what condition an overlay can be stabilized, how can the system be scalable and what are the fundamental limitations and tradeoffs. None of the existing works have provided a comprehensive study on these important issues. To the best of our knowledge, this paper represents the first attempt to address these problems from a practical point of view. In this study, we demonstrate the real engineering problems we have encountered; concretely describe how those problems are resolved with various mechanisms; and how different components interact with one another. Further, we provide analysis whenever necessary and our intuitions behind the system design.

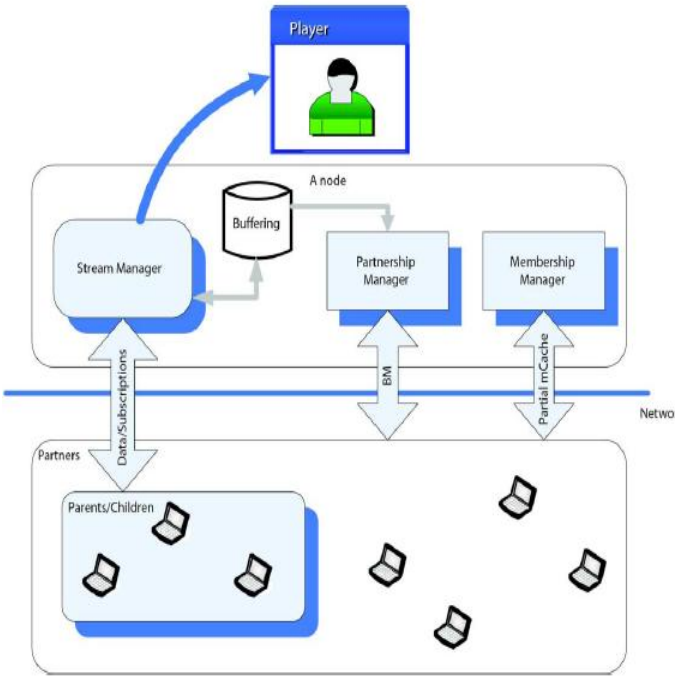


Fig. **2. Cool streaming System Diagram**

### 2.2 Impacts of Peer Characteristics on P2PTV Networks Scalability

A typical P2PTV network consists of a source and multiple clients. Clients can connect directly to the source to view the video stream, or connect to other already connected clients to get the stream, thus creating a P2P network. The video is a live television stream, typically an analog or digital stream trans coded in H.264. An H.264 video stream is typically cut into several slices (i.e., amplitude division) and into chunks (i.e., time division). The stream is transported by UDP or TCP flows over unicast IP forwarding between participating peers. In this paper, we assume that no IP multicast is used as it is not currently available Internet-wide. P2P broadcasting is different from traditional P2P data sharing

or A/V asynchronous streaming (e.g., video on demand). Download rates above the stream bitrate are not necessary (except for error recovery). No data storing is needed and buffering is limited only to what is needed (through the use of a relevant time window). Upload capacity and session length are, however, important parameters for the overlay network as they determine its scalability and churn. Finally, P2PTV applications are very sensitive to delay and, to a lesser extent, loss rate.

The concept of media streaming and especially live media streaming is not new. Since 2003, many research prototypes have been created and evaluated. To name a few in order of publication date: Zigzag by Tran et al., PRO by Rejoice and Stafford, Anise by Liao et al., Cool Streaming by Zhang et al. and PULSE by Piranesi et al. A number of fully operational P2PTV systems then followed circa 2005. Aside from Tattoo, the most popular systems have been developed in China (e.g., PP Live, PP Stream, Sop Cast, TV Ants, and Unused), but some notable initiatives have emerged from Europe as well (e.g., Joust, Live station, and Raw flow). As a result of the P2PTV's growing popularity, a large number of measurement papers have recently been written on these P2PTV systems, tackling workloads, P2P topologies, or specific systems Being one of the most popular systems, PP Live has been extensively studied by Hei et al, Vu et al

**2.3 Insights into PP Live: a Measurement Study of large-scale P2P IPTV System**

IPTV is expected to be the next disruptive IP communication technology, potentially reshaping our media and entertainment culture. However, provisioning the IPTV service brings forth significant new challenges. IPTV systems can be broadly classified into two categories: infrastructure-based and peer-to-peer based. In infrastructure-based systems, video servers and application-level multicast nodes are strategically placed in the Internet, and video is streamed from servers to clients via the multicast nodes. The infrastructure-based IPTV systems are expensive to build and difficult to maintain. On the other hand, peer-to-peer IPTV systems do not rely on dedicated application-level multicast servers. Instead, each IPTV client is potentially a server, multicasting received content to other IPTV clients. The IPTV clients and the connections between them thus form an overlay network, cooperatively exchanging video content by leveraging the uploading capacity of the peers. Several P2P IPTV systems have been developed and deployed to date. Among them, Cool Streaming and PP Live have been two of the most popular P2P-based IPTV applications. Zhang et al. reported that more than 4, 000 Cool Streaming users were simultaneously online at some peak time. Using a crawler (to be discussed in a subsequent paper), we observed in our PP Live measurements more than 100, 000 simultaneously online users for a live broadcast of a popular TV program. To the best of our knowledge, PP Live is by far the most popular P2P IPTV application on the Internet today. Given its success and its low-cost P2P architecture, it is our position that the designers of future IPTV systems should understand PP Live design, performance, traffic characteristics and, more broadly, its strengths and its flaws. Nevertheless, PP Live employs proprietary signaling and video delivery protocols. Details about its performance, streaming workload and overlay characteristics are still largely unknown. In this paper we present results from a preliminary measurement study of PP Live. We have been measuring PP Live with passive packet sniffing as well as with an active crawler. In this paper, we only present the sniffing results. Our measurement study focuses on three important aspects of PP Live streaming: streaming performance, workload characteristics, and overlay properties. Quantitative results obtained in our study bring to light important performance and design issues of live streaming over the public Internet

PP Live is a free P2P-based IPTV application. According to the PP Live web site in January 2006, the PP Live network provides 200+ channels with 400, 000 daily users on average. The bit rates of video programs mainly range from 250 Kbps to 400 Kbps with a few channels as high as 800 Kbps. The PP Live network does not own video content. The video content is mostly feeds from TV channels in Mandarin. The channels are encoded in two video formats: Window Media Video (WMV) or Real Video (RMVB). The encoded video content is divided into chunks and distributed to users through the PP Live P2P network. The PP Live web site provides limited information about its video content distribution mechanism. Nevertheless, various web sites and message boards provide additional information, which we collected from different sources and confirmed by our own measurement results. The PP Live software implements two major application level protocols: a gossip-based protocol for peer management and channel discovery; and a P2P-based video distribution protocol for high quality video streaming. Figure 3 depicts an overview of the PP Live network.

When an end-user starts the PP Live software, it joins the PP Live network and becomes a PP Live peer node. It then sends out a query message to the PP Live channel server to obtain an updated channel list (step 1). Before a peer actually starts to watch a channel, it does not exchange data with other PP Live peers. After a peer selects one channel to watch, it sends out multiple query messages (step 2) to some root servers to retrieve an online peer list for this channel. Peers are identified by their IP addresses and port numbers on the list.

Upon receiving a peer list, the PP Live client sends out probes to peers on the list to find active peers for the channel of interest (step 3). Some active peers may also return their own peer lists, helping the initial peer to find more peers. Peers then share video chunks with each other, as described below.

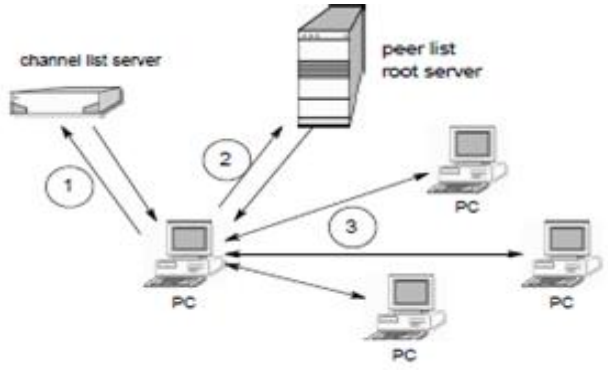


Fig.3 Channel and peer discovery

The major software component of PP Live is its TV engine. This TV engine is responsible for downloading video chunks from the PP Live network and streaming the downloaded video to a local media player. The streaming process in the PP Live traverses two buffers in local memory: the PP Live TV engine buffer and the media player buffer, as shown in Figure 4. This double buffering mechanism is designed with two goals. One is to pre-cache media content to combat download rate variations from the PP Live network; the other is for efficient content distribution between peers.

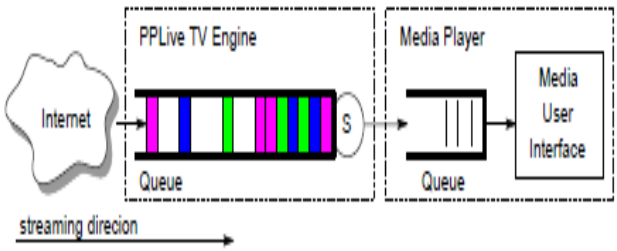


Fig.4. PPLive streaming process

The cached contents can be uploaded to other peers that are watching the same channel. Specifically, the peer client contacts multiple active peers to download media content of the channel. At the same time, this peer client may also upload cached video chunks to multiple peers. Received video chunks are reassembled in order and buffered in the queue of the PP Live TV engine, forming a local streaming file in memory. When the streaming file length crosses a predefined threshold, the PP Live TV engine launches a media player, which downloads video content from the local HTTP streaming server. Most media players, such as windows media player, have built-in video buffering mechanisms. After the buffer of the media player fills up to the required level, the actual video playback starts. When PP Live starts, the PP Live TV engine downloads media content from peers aggressively to minimize the playback start-up delay. When the media player receives enough content and starts to play the media, the streaming process gradually stabilizes. The PP Live TV engine streams data to the media player at the media playback rate.

We conducted a measurement study on a popular IPTV application, PPLive. Our measurement results show that the PPLive deploys the P2P principles for efficient resource discovery and video distribution. Utilizing the best-effort Internet infrastructure, the PPLive streaming maintains satisfactory IPTV performance. This demonstrates that the current Internet infrastructure is capable to provide economic-viable IPTV services while meeting the performance requirements of IPTV. Nevertheless, the emerging IPTV applications exhibit different characteristics from other applications, which may change Internet traffic pattern significantly. This brings forth new challenges and opportunities for network service providers.

## III. SYSTEM ANALYSIS

The **Systems Development Life Cycle (SDLC)**, or *Software Development Life Cycle* in systems engineering, information systems and software engineering, is the process of creating or altering systems, and the models and methodologies that people use to develop these systems.

In software engineering the SDLC concept underpins many kinds of software development methodologies. These methodologies form the framework for planning and controlling the creation of an information system the software development process.

### 3.1. SOFTWARE DEVELOPMENT LIFE CYCLE

The ideas about the software development life cycle (SDLC) have been around for a long time and many variations exist, such as the waterfall, and the V-model. These variations have many versions varying from those which are just guiding principles, to rigid systems of development complete with processes, paperwork, and people roles. However underlying all these are a set of common principles.

#### *A. SDLC Common Principles*

The common principles behind the SDLC are:

The process of developing software consists of a number of phases. These phases are arranged in a precedence sequence of when they start. The sequence of phases represents the passage through time of the software development. Phases can and do overlap as previous phases are revisited, when more information becomes available. The software becomes more

complex and useful as the phases are followed. These principles apply to whichever particular variation of the SDLC is looked at, with emphasis being placed on particular principles in each variation

### *B. SDLC Common Phases*

As you look at the various methods they all have a number of particular phases they recommend. On the proposed model there will be a specific set of phases. This set is practical, but is not the definitive answer for all software development. The V-Model example is just to demonstrate the issues to be considered in development and how they affect each other.

The proposed the "4D" model as a generic model for understanding the larger issues. This has four phases:

- **Decide** – What is it you want to build in software?
- **Design** – How will you map these decisions to a software environment?
- **Develop** – Build the software according to the designs.
- **Demonstrate** – Prove that the software delivers what was required.

#### *1) Decide Phase*

The Decide phase covers all those activities involved in deciding what it is that you want to build. The products from this phase typically include:

- Business cases to justify what is wanted in terms of business benefit.
- Feature lists of what is wanted to be included to deliver that benefit.
- Use Cases to explore how the features would work together.
- Non-functional requirements are the performance and development constraints placed on the system.
- System Specification which maps between what is wanted in the real world and what is possible in a computer system.

This phase is frustrating one – it is necessary but it can be perceived as delaying the eventual system. Also it requires a large commitment of user's time to decide this, often while they are involved in doing their normal day job. As a result many projects have a poor set of deliverables from the Decide phase before the Design phase starts. If however there is a process to allow modification of these products throughout the project then the Decide products will improve.

#### *2) Design Phase*

The Design phase takes the products from the Decide phase and creates a design of the architecture and detail working of how the software system will deliver what is wanted. The key thing to note is that despite many efforts over the years there is no automatic way of deriving the Design products from the Decide products. Where this has been claimed before inevitably it has been by restricting the way the Decide products are written and forcing them to be expressed in the form of a computer design. However it is possible, and necessary, to compare the Design products with the Decide products to see if what is wanted has been included. The Design phase is done by the analyst and design team who should work closely with developers in the Develop team.

#### *3) Develop Phase*

The Develop phase is what most users consider to be what software development is about. Paradoxically in many ways it is the least important of the phases, even though it will consume a lot of the resources. The reason is that most systems are constructed from a set of standard parts with some configuration, some customization, and some bespoke parts to make the system unique. These decisions would have been taken in the Design phase, and in the Develop phase the work of converting them into a software system takes place. The products from this phase then have to be shown to work.

#### *4) Demonstrate Phase*

The Demonstrate phase is about proving that the delivered system works and is what was wanted. This phase is not just about testing but contains activities such as document reviews and code walkthroughs. It has a high degree of overlap with the other phases as the earlier you can catch a problem results in higher quality in the final product. This phase is done by both the developers and the users.

We have called these four phases a "4D" model as it implies it is multi-dimensional including a time dimension. However these models are always depicted as a two-dimensional diagram and as a result some of the universal features of a SDLC are obscured or lost. Figure 5 - 4D SDLC - is a non standard way of showing a SDLC showing several of the principles including:

### *C. SDLC Diagrams*

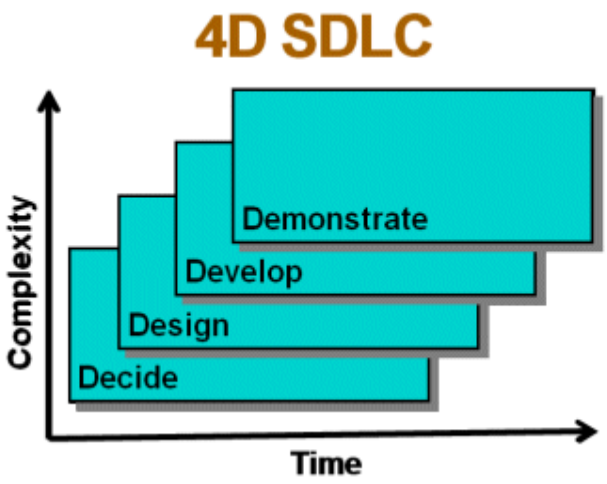


Fig. 5 – "4D" SDLC model

1. overlap and moving up the complexity and time scales.
2. They move along the time scale.
3. Phases overlap implying that there is no fixed finish between the start of one phase and the start of the next. It also implies that previous phases are revisited when further information is found.
4. The phases rising up imply both increasing complexity and also the amount of effort required to reach a final product.
   This figure is different compared with the normal waterfall approach such as in figure 6 - Traditional SDLC.

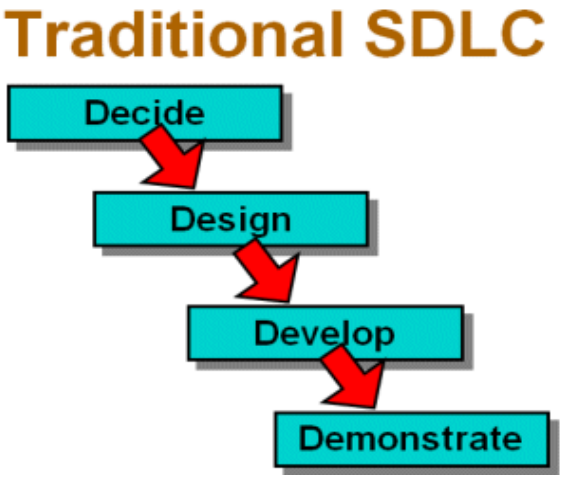


Fig. 3.2 – Waterfall SDLC model

Comparing this against the 5 principles shows:

1. 1. It also clearly shows the four phases.
2. 2. It also shows the precedence order.
3. 3. They also move along the time scale.
4. 4. However the diagram implies that there is a clear finish of each phase before the next starts. Very few Software Development Life Cycles recommend this, but this is the impression the diagram gives.
5. 5.The products falling down give the visual impression of little effort being required, nor does it give an impression of the greater complexity being built up.

## III. CONCLUSION

The presented a receiver-based, peer-division multiplexing engine to deliver live streaming content on a peer-to-peer network. The same engine can be used to transparently build a hybrid P2P/CDN delivery network by adding Repeater nodes to the network. By analyzing a large amount of usage data collected on the network during one of the largest viewing events in Europe, hence it was shown that the resulting network can scale to a large number of users and can take good advantage of available uplink bandwidth at peers. It was also shown that error-correcting code and packet retransmission can help improve network stability by isolating packet losses and preventing transient congestion from resulting in PDM reconfigurations. Hence it was concluded that the PDM and adaptive PDM schemes presented have small enough overhead to make our system competitive to digital satellite TV in terms of channel switch time, stream synchronization, and signal lag.

## REFERENCES

[1] R. auf der Maur, "Die Weiterverbreitung von TV- und Radioprogrammenüber IP-basierte Netze," in *Entertainment Law*, F. d. Schweiz, Ed., 1st ed. Bern, Switzerland: Stämpfli Verlag, 2006.

[2] S. Lin and D. J. Costello Jr., *Error Control Coding*, 2nd ed. Englewood Cliffs, NJ: Pearson Prentice-Hall, 2004.

[3] S. Xie, B. Li, G. Y. Keung, and X. Zhang, "CoolStreaming: Design, theory, and practice," *IEEE Trans. Multimedia*, vol. 9, no. 8, pp. 1661–1671, Dec. 2007.

[4] K. Shami *et al.*, "Impacts of peer characteristics on P2PTV networks scalability," in *Proc. IEEE INFOCOM*, Apr. 2009, pp. 2736–2740.

[5] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross, "Insights into PPLive: A measurement study of a large-scale P2P IPTV system," in *Proc. IPTV Workshop, Int. World Wide Web Conf.*, May 2006.

[6] B. Li *et al.*, "An empirical study of flash crowd dynamics in a P2Pbased live video streaming system," in *Proc. IEEE GLOBECOM*, 2008, pp. 1–5.

[7] J. Rosenberg *et al.*, "STUN—Simple traversal of User Datagram Protocol (UDP) through network address translators (NATs)," RFC 3489, 1993.

[8] A. Ali, A. Mathur, and H. Zhang, "Measurement of commercial peer-to-peer live video streaming," in *Proc. Workshop Recent Adv. Peer-to-Peer Streaming*, Aug. 2006.

[9] L. Vu *et al.*, "Measurement and modeling of a large-scale overlay for multimedia streaming," in *Proc. Int. Conf. Heterogeneous Netw. Quality, Rel., Security Robustness*, Aug. 2007, pp. 1–7.

[10] T. Silverston and O. Fourmaux, "Measuring P2P IPTV systems," in *Proc. ACM NOSSDAV*, Jun. 2007.

[11] C. Wu, B. Li, and S. Zhao, "Magellan: Charting large-scale peer-topeer live streaming topologies," in *Proc. ICDCS*, 2007, p. 62.

[12] M. Cha *et al.*, "On next-generation telco-managed P2P TV architectures," in *Proc. Int. Workshop Peer-to-Peer Syst.*, Feb. 2008.

[13] D. Ciullo *et al.*, "Understanding P2P-TV systems through real measurements," in *Proc. IEEE GLOBECOM*, Nov. 2008, pp. 1–6.

[14] E. Alessandria *et al.*, "P2P-TV systems under adverse network conditions: A measurement study," in *Proc. IEEE INFOCOM*, Apr. 2009, pp. 100–108.
[15] S. Banerjee *et al.*, "Construction of an efficient overlay multicast infrastructure for real-time applications," in *Proc. IEEE INFOCOM*, 2003, vol. 2, pp. 1521–1531.
[16] D. Tran, K. Hua, and T. Do, "ZIGZAG: An efficient peer-to-peer scheme for media streaming," in *Proc. IEEE INFOCOM*, 2003, vol. 2, pp. 1283–1292.
[17] D. Kostic *et al.*, "Bullet: High bandwidth data dissemination using an overlay mesh," in *Proc. ACM SOSP*, Bolton Landing, NY, Oct. 2003, pp. 282–297.