Chameleon: Adaptive Peer-to-Peer Streaming with Network Coding

Anh Tuan Nguyen †, Baochun Li ††, and Frank Eliassen †
† *Department of Informatics, University of Oslo, Oslo, Norway*
{*anhtn,frank*}*@ifi.uio.no*
†† *Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada*
bli@eecg.toronto.edu

***Abstract*—Layered streaming can be used to adapt to the available download capacity of an end-user, and such adaptation is very much required in real world HTTP media streaming. The multiple layer codec has become more refined, as SVC (the scalable extension of the H.264/AVC standard) has been standardized with a bit rate overhead of around 10% and an indistinguishable visual quality, compared to the state of the art single layer codec. Peer-to-peer streaming systems have also become the reality. The important question is how such layered coding can be used in real world peer-to-peer streaming systems. This paper tries to explore the feasibility of using network coding to make layered peer-to-peer streaming much more realistic, by combining network coding and SVC in a fine granularity manner. We present *Chameleon*, our new peer-to-peer streaming algorithm designed to incorporate network coding seamlessly with SVC. Key components with different design options of Chameleon are presented and experimentally evaluated, with the objective of investigating benefits of network coding in combination with SVC. We carry out extensive experiments on real stream data to (i) evaluate the performance of Chameleon in terms of playback skips and delivered video quality, and (ii) understand its insights. Our results demonstrate the feasibility of the approach and bring us one step closer to real adaptive peer-to-peer streaming.**

I. Introduction

As video streams are transferred across best-effort IP networks, no Quality of Service guarantees can be made. Therefore, it is much required to have some means for video streaming to be able to adapt to network fluctuations. Layered coding has emerged as one of the most promising solutions, in which scalable video coding has been shown advantageous in terms of coding efficiency over other alternatives, such as Multiple Description Coding (MDC). The layered codec has become more refined and practical. Recently, the scalable extension of the H.264/AVC standard, referred to as SVC, has been standardized as Amendment 3 of H.264/AVC [1]. Performance evaluations of SVC have shown that, a reasonable degree of scalability can be supported with a bit rate overhead of less than 10% and an indistinguishable visual quality, compared to the state of the art single layer coding [2], [3].

The peer-to-peer (P2P) paradigm has been successfully used in live multimedia streaming over the Internet [4], [5]. The essential advantage of P2P systems is that the system capacity scales up when more peers join, as peer upload capacity is utilized. Although P2P streaming takes advantage of the P2P paradigm to mitigate server load, peers still suffer degraded video quality when bandwidth variations occur. The important question here is how layered coding can be used in P2P streaming to take advantage of both emerging technologies. Many approaches have been proposed to use layered coding in P2P streaming. However, a complete solution that can be applied to real world systems has never existed for the following reasons:

- ▷ Peer coordination, while critical to utilize available upload bandwidth from each peer to maximize the delivered quality under network fluctuations, is a complex problem. For example, the layer allocation problem is proven to be NP-hard [6], [7].
- ▷ Generic layered data models are often used. Previous studies [6]–[9] often do experiments with synthetic layered data. The use of unreal stream structures limits the applicability of designed protocols in the real world.

Network coding [10] has been shown beneficial in P2P streaming [5], [11]. Wang and Li present the current state-of-the-art live P2P streaming using random network coding in single layer streaming [5]. The key feature of network coding is that it makes all pieces of data equally important, and every coded packet is innovative to receivers with high probability. This feature maximizes the potential of peer collaboration (referred to as *perfect collaboration* in [5]).

Convinced that network coding is beneficial, we explore the feasibility of using network coding to make layered P2P streaming much more realistic, by combining network coding and SVC in a fine-granularity manner. To fully evaluate the approach, a complete adaptive P2P streaming protocol, *Chameleon*, is designed. We expect to see benefits of network coding in mitigating peer coordination problems. To be closer to reality, we take into account the scalability structure specified by the SVC standard from the design to the evaluation phase. We carry out extensive experiments with different design options of the protocol on real stream data (a two-hour video sequence) generated from the latest SVC reference software, JSVM Software 9.17 [12]. Our results show that Chameleon can adapt to bandwidth variations to provide the best possible quality, while maintaining efficiency and scalability of a P2P system. In summary, our main contributions include:

- ▷ An effective and complete P2P streaming protocol. The core of Chameleon is studied, including neighbor selection, quality adaptation, receiver-driven peer coordination, and sender selection with different design options. Our

This work is partially funded by the Research Council of Norway, grant number 176756/S10.

investigation brings interesting findings that are useful for building an adaptive P2P streaming system.

- ▷ A segmentation method to use SVC in P2P streaming in combination with network coding.

The remainder of this paper is organized as follows. Section II discusses related work. The motivation for using network coding is mentioned in Section III. Section IV presents the combination of network coding and SVC. The design of Chameleon is described in Section V. Performance evaluation results are discussed in Section VI. We conclude the paper in Section VII.

II. Related Work

The work of Cui *et al.* [6] and Rejaie *et al.* [8] can be considered as two of the first efforts in layered P2P streaming. Cui *et al.* prove that the layer allocation problem is NP-hard, and propose a greedy approach to assign layers to senders. Later, Liu *et al.* [7] present another approach to the problem. They formulate it as an optimization problem with the constraints of available bandwidth capacity and layers, and use an approximation algorithm, FABALAM, to simplify the problem. Although FABALAM has been demonstrated to be able to achieve better performance than the approach of Cui *et al.*, they both rely on static layer-to-sender mapping, and a layer is provided by only one sender.

Rejaie *et al.* introduce PALS, a receiver-driven approach for quality adaptive playback. In PALS, a layer is divided into packets and provided by multiple senders. Therefore, it better utilizes the available bandwidth capacity of senders. In addition, PALS addresses bandwidth variations and peer dynamics in a timely manner. The receiver peer periodically sends an ordered list of packets to each of its senders, and each sender delivers the requested packets to the receiver in the given order. Recently, Magharei and Rejaie present an extended version of PALS with extensive simulations in ns-2 [9]. However, although PALS is motivated for P2P streaming, its performance is currently evaluated for the case of streaming from multiple senders to one receiver. Its performance in P2P scenarios has not yet been shown. This is the reason that we can not compare Chameleon with PALS in this paper. In another direction, Magharei and Rejaie introduce PRIME with the goal of minimizing content and bandwidth bottlenecks in mesh-based streaming by deriving proper peer connectivity and an efficient pattern of delivery [13]. However, PRIME uses MDC while Chameleon uses SVC.

Annapureddy *et al.* [11] show that network coding helps to provide high quality Video-on-Demand (VoD) services. Network coding is applied over small time-windows (*e.g.*, a segment with a few seconds worth of video frames) of a single-layer stream. The coding prevents the occurrence of rare blocks within a segment. In addition, it ensures that bandwidth is not wasted in distributing the same block multiple times, *i.e.*, it minimizes the risk of making incorrect upload decisions.

In single layer P2P streaming, the work of Wang and Li [5] is considered as the current state-of-the-art. They show that random network coding with a simple data delivery scheme not only reduces bandwidth costs and initial buffering delays, but also makes the system resilient to peer dynamics and bandwidth variations during streaming sessions. Their work is evaluated by experiments on an actual implementation, real network traffic, and emulated peer upload capacities.

Zhao *et al.* [14] propose LION, a layered overlay multicast system. Our approach is similar to theirs in the way that they also explore the combination of layered coding and network coding. However, they focus on overlay construction while we focus on a complete streaming protocol. In addition, LION has a well-structured overlay and is aimed to support small-scale application scenarios in quite stable environments, while Chameleon is proposed for unstructured overlays and geared towards the goal of architecting a live, adaptive, and scalable P2P streaming system.

III. Motivation for Using Network Coding

In addition to conventional challenges in P2P streaming, *e.g.*, peer selection and packet scheduling, layered P2P streaming poses unique and challenging problems, of which two of the most important issues are *peer coordination* and *quality adaptation*.

With respect to *peer coordination*, the bandwidth and data availability of each peer are constrained and varied, which further limit the data availability (content bottleneck) and bandwidth (bandwidth bottleneck) of downstream peers. Peer coordination is critical to the system performance because it controls the collaboration of sending peers to utilize available bandwidth from each sender to maximize the delivered quality at the receiving peer. Two important questions are:

- ▷ *Is a layer supplied by one or more than one sender?* If a layer is delivered by only one sender, coordination may be simpler, but the residual bandwidth of each peer may not be fully utilized. On the other hand, assigning partial layers to senders requires a proper division of a layer across multiple senders.
- ▷ *How do we map packets to senders appropriately?* Given an ordered list of required packets and potential senders, the problem is to determine which packets are to be delivered from each sender.

The purpose of *quality adaptation* is to avoid playback skips and to maximize the video quality when bandwidth variations occur. Challenges in quality adaptation are:

- ▷ *How does a peer choose layers to be requested at a point of time?* The selection should be based not only on playback deadlines, but also on streaming quality (the number of layers) the peer aims to deliver.
- ▷ *How and when is quality adaptation invoked?*

We believe that network coding can help to solve the above problems with ease. With network coding, a peer only needs to check if it has received a sufficient number of linearly independent coded blocks, without being concerned with who has been sending them. The probability of receiving "duplicate" blocks is so low that multiple senders can serve blocks to the same receiver without the need of any explicit coordination. In addition, since coded blocks are equally useful to the receiver, the responsibilities of a particular sender can be easily transferred to other senders if it leaves the system. Even the computational complexity of network coding is no longer a

concern: Wang and Li [15] have implemented a decoding process using Gauss-Jordan elimination, such that it can be performed while coded blocks are progressively received. Shojania *et al.* [16] propose an implementation that efficiently takes advantage of multiple CPU cores and SIMD instruction sets in modern CPUs. In a nutshell, network coding can be efficiently implemented, and it maximizes the collaboration potential among peers.

However, combining network coding with SVC is not straightforward. SVC prioritizes video data to provide different quality levels by allowing the extraction of substreams. Meanwhile, network coding makes data packets equally important to ease the data delivery, and the original data is only recovered when enough linearly independent blocks are received (the *all-or-nothing* property). How do we combine network coding and SVC? How much can network coding help? In the remainder of the paper, we present the design of *Chameleon* to address these challenges with a seamless combination of network coding and SVC.

IV. The Combination of Network Coding and SVC

Since SVC was included in the H.264/AVC Standard Version 8 in July 2007 [17], we are not aware of any existing work in the P2P streaming literature that focuses on using SVC as the layered codec of choice. In this section, we take the first step by presenting a segmentation method to use SVC, seamlessly combining with network coding at a fine granularity. We start with a brief description of important features in SVC. For more details about SVC, interested readers are referred to [1], [17]–[19].

A. Background: An Overview of SVC

SVC supports three modes of scalability: *temporal*, *spatial*, and *quality* scalability. In spatial scalability and temporal scalability, subsets of the bit stream represent the source content with a reduced picture size (spatial resolution) or frame rate (temporal resolution). With quality scalability, the substream provides the same spatial-temporal resolution as the complete bit stream, but with a lower fidelity (Signal-to-Noise Ratio). The scalability structure is characterized by three syntax elements: D_ID, T_ID, and Q_ID for spatial, temporal, and quality scalability respectively.

1) SVC Structure: An SVC video is organized into Network Abstraction Layer (NAL) units, which are packets that each contains an integer number of bytes. NAL units are grouped into logical entities. A *layer representation* (LR) consists of all NAL units representing an original picture and pertaining to a combination of D_ID and Q_ID. An *access unit* (AU) consists of all LRs that represent an original picture. The decoding of an AU results in exactly one decoded picture. A *group of pictures* (GOP) is a group of successive pictures. The GOP structure specified by picture types (I-, B-, P- pictures) determines the temporal scalability of the stream. A *coded video sequence* represents an independently decodable part of a NAL unit bit stream. It starts with an instantaneous decoding refresh (IDR) AU, and following AUs can be decoded without decoding any previous pictures of the bit stream. It ends before the next IDR AU or at the end of the bit stream, whichever is earlier. Figure 1 shows the structure.

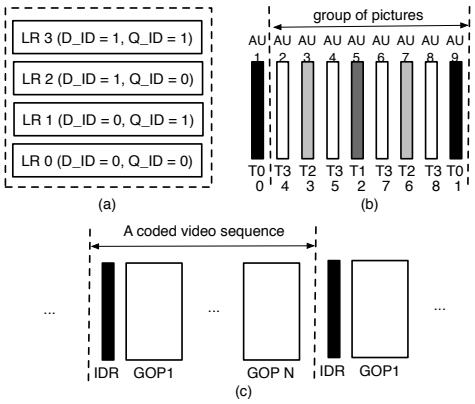


Fig. 1. An example of the SVC structure. (a) An AU consisting of four LRs. (b) A GOP consisting of eight pictures (AUs) and coded with hierarchical B-pictures. The symbols *Tk* specify the temporal layers with *k* representing the corresponding T_ID. The numbers below specify the coding order. (c) A coded video sequence.

2) Decoding Dependency: At any AU, an LR of a smaller D_ID may be used for decoding an LR with a greater D_ID; and for a particular D_ID, an LR with Q_ID always uses the LR with $Q\_ID - 1$. Finally, a given T_ID depends on the smaller T_ID.

3) BitStream Switching: SVC allows switching between different scalable levels during streaming to provide adaptability. However, switching operations can only occur at specific points of a stream:

- ▷ Switching between spatial layers can only occur at IDR AUs.
- ▷ Switching between quality layers within a spatial layer can occur at *any* AU.
- ▷ Switching between temporal layers within a spatial layer can occur at *any* AU or only at temporal layer switching points depending on encoding parameters.

B. Proposed Segmentation Method

To be transmitted across IP networks, an SVC stream needs to be divided into segments. The segmentation method should preserve the scalability of the stream so that (i) adaptation can be operated on segments (*adaptability*), and (ii) the regenerated stream is a valid stream (*validity*). In an SVC video file, the video entities are arranged in the specific order: from one GOP to another. Within a GOP, AUs are sorted on the decoding order; and within an AU, LRs are sorted on (D_ID, Q_ID). Figure 2 depicts this order.

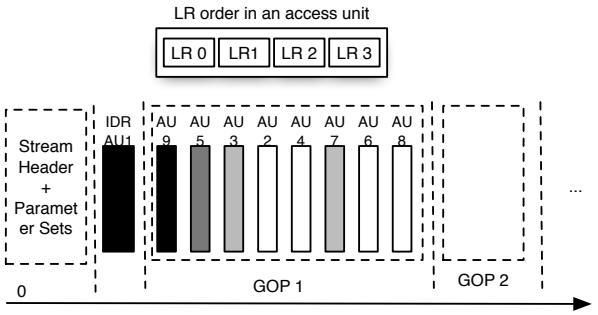


Fig. 2. The store order of the entities in the video file

In P2P streaming, a peer does not receive a complete stream to extract valid substreams for other peers; it, instead, receives

only pieces of video data to constitute a stream according to its download capacity. Therefore, to maintain the adaptability and validity, the video entities should be re-arranged, and the segmentation method should be based on layer switching enabled points to support particular scalability modes. We segment an SVC stream based on the boundary of GOPs, because switching between temporal and quality levels within a GOP is independent from other GOPs, and spatial switching is only allowed at IDR AUs (outside of any GOP).

In the following, we describe our proposed segmentation for SVC with quality scalability, but it is clear that the method can be applied to other scalability modes. An SVC stream is first divided into segments, each of which consists of an integer number of GOPs. Then, within each segment, NAL units are grouped into packets based on Q_ID from the lowest to the highest value. Since each NAL unit header contains (D_ID, Q_ID, T_ID) of the scalable layer it belongs to, it is always possible to recover the original order [19]. Figure 3 illustrates the segmentation method.

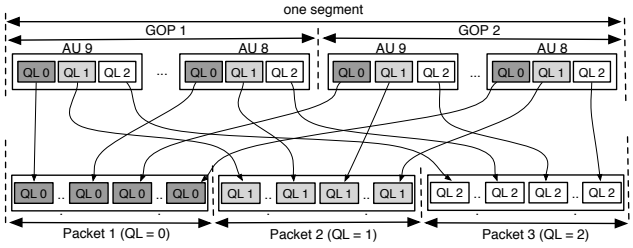


Fig. 3. An example of the segmentation method where the stream has three quality levels and is divided into segments of two GOPs. The symbols *QL k* specify $Q\_ID = k$.

During streaming, packets are exchanged among peers. Packet 1 in each segment contains the base layer and is necessary for every peer, while other packets can be received or not depending on download capacity. Since each packet contains all NAL units of one quality layer in the segment, streams that are generated by dropping one or more packets (except the base packet) are valid streams, *i.e.*, the segmentation method guarantees the adaptability and validity requirement.

C. SVC with Random Network Coding

Our approach in *Chameleon* is to apply random network coding to scalable layers, based on which scalability mode the system aims to support. For example, if an SVC stream with a certain number, N_s, of scalable layers is divided into segments as previously proposed, each segment would now contain N_s packets. Each packet is further divided into N blocks $[b_1, b_2, ..., b_N]$, all blocks of a packet have the same number of bytes k (referred to as the *block size* of that packet). When a peer performs network coding for layer l, it randomly chooses a set of coding coefficients $[c_1, c_2, ..., c_M]$ ($M \leq N$) in GF(2^8). It then randomly chooses M blocks of layer l — $[b_1^l, b_2^l, ..., b_M^l]$ — out of all the blocks of the layer it has received so far, and produces the coded block x of k bytes:

$$x = \sum_{i=1}^{M} c_i \cdot b_i^l$$

With Gauss-Jordan elimination implemented in the decoding process, a peer starts to progressively decode a packet, as soon as it receives the first coded block of this packet. As a total of N linearly independent coded blocks $X = [x_1, x_2, ..., x_N]$ has been received, the original blocks can be immediately recovered as Gauss-Jordan elimination computes $B = A^{-1}X^T$, where A is the matrix formed by coding coefficients of X. Figure 4 shows the combination of network coding and SVC for the stream in Figure 3.

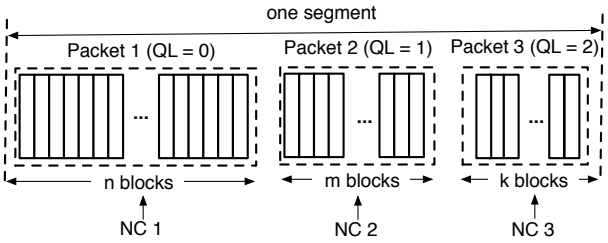


Fig. 4. An example of the combination of network coding and SVC. Packet 1, 2, and 3 are divided into n, m, and k blocks, respectively. Network coding with different number of unknowns (n, m, and k) is used for different quality levels.

V. Chameleon: Adaptive P2P Streaming with Network Coding

In this section, we present the design of *Chameleon*, our new adaptive P2P streaming protocol that seamlessly integrates SVC with network coding. We consider a typical P2P streaming session with a number of dedicated streaming servers, and a large number of peers. Peers participate in and depart from a session in unpredictable ways, and they are heterogeneous with different bandwidth capacities.

A. System Overview

The primary design goal of Chameleon is to effectively utilize available bandwidth capacity of each peer to maximize delivered quality under bandwidth variations. Figure 5 shows the internal architecture of Chameleon with key components and their relations. When a peer joins the system or when it needs to update the neighbor list for better quality, it creates neighboring relationships with other peers. A list of available peers can be provided by a rendezvous peer or by exchanging membership information, *e.g.*, using SCAMP [20]. The *neighbor selection* component chooses a number of peers to be neighbors. Information about each neighbor, *e.g.*, IP address, average experienced quality, current number of neighbors, etc. is stored in the neighbor list. During streaming, a peer needs to decide how many quality levels it aims to receive according to its current bandwidth capacity. The *quality adaptation* component will make decisions on keeping, increasing, or decreasing the current quality level, based on the status of the playback buffer and the available download capacity. When adaptation occurs, the *sender selection* component will select potential senders from the neighbor list, based on the decision from the adaptation process. The *peer coordination* component will send layer requests to the selected senders. The senders are expected to collaboratively send coded blocks of the requested layer to the receiver. When the playback deadline is reached, one segment in the playback buffer is sent to the player.

In addition to the above components, *buffer maps* are used to exchange necessary information among peers. In traditional

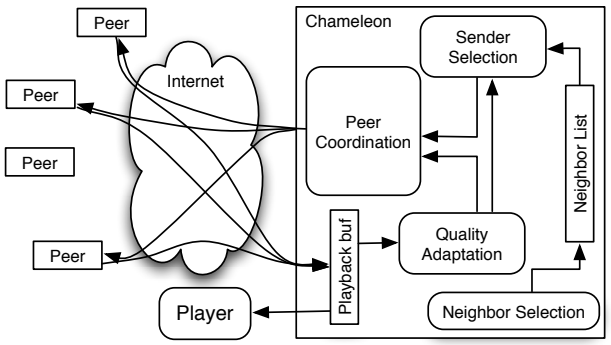


Fig. 5. Architecture of Chameleon with key components.

systems, only one bit is used to represent the availability information of video data. With Chameleon, we use 2 bits to represent the following four meanings: (1) the peer has received enough linearly independent blocks to decode the packet; (2) the peer has not received enough linearly independent blocks; (3) the peer has not received enough linearly independent blocks to decode but enough to serve other peers; and (4) the peer does not need to receive the packet (quality adaptation). For a layer, a *ready-to-serve* peer is the peer that has received $\alpha \cdot N$ coded blocks from other peers ($0 < \alpha \leq 1$) for that layer, in which the tunable parameter α is referred to as *aggressiveness* [5]. A peer sends out its buffer map to its neighbors when the status of the buffer map changes. Using one more bit causes slightly more overhead. However, as pointed out for R^2 [5], this overhead is still acceptable and less than traditional protocols, since much larger segments are used with network coding.

B. Design Space of Key Components

We now turn our attention to the details of each key component. Different design options for each component are presented and experimentally compared. For the simulation setting used in this section, please refer to Section VI.

1) Class-based vs. Quality-based Neighbor Selection: It is likely that peers with similar capacity should be connected to each other to maximize collaboration potential because they are supposed to receive the same quality. As in some previous studies, we first try a class-based selection method. Peers can be classified into classes based on the highest quality level they can achieve. We say that a peer belongs to class C when its best possible quality level according to its download capacity is C. A peer connects to other peers in the following priority: peers of the same class, peers of higher classes, and peers of lower classes. If there are more peers than needed, choose randomly. However, as we will see, class-based selection does not work very well. In Chameleon, we propose quality-based selection as follows. Each peer calculates the average quality level it has perceived so far. When a peer selects a neighbor, it will choose the candidate whose average quality level is closest to its class. If there are more than one peer, choose one randomly.

To experimentally evaluate the neighbor selection methods, consider a generic method as follows. We denote C_i and AQ_i as the class and the average quality level of peer P_i, respectively. When a peer P_k chooses a new neighbor from its candidate list L_k, it will choose peer P_q that satisfies the following condition:

$$|C_k - AQ_q| - \min_{j \in L_k}(|C_k - AQ_j|) \leq \tau$$

If more than one peer satisfies the condition, class-based selection is applied. The above condition is designed to choose peers whose average quality level is closest to the peer class of P_k within the range τ. If τ is equal to 0, we have pure quality-based selection. If τ is equal to the highest quality level of the stream, we have pure class-based selection because all peers in L_k satisfy the condition. Otherwise, we have a hybrid approach. By experimenting with different values of τ, we explore how different neighbor selection methods affect Chameleon. Figure 6 plots the skip rates and the average quality satisfaction with τ ranging from 0 to 2.6 (the skip rates and the average quality satisfaction with $\tau \geq 2.6$ are the same). Note that the stream has four quality layers.

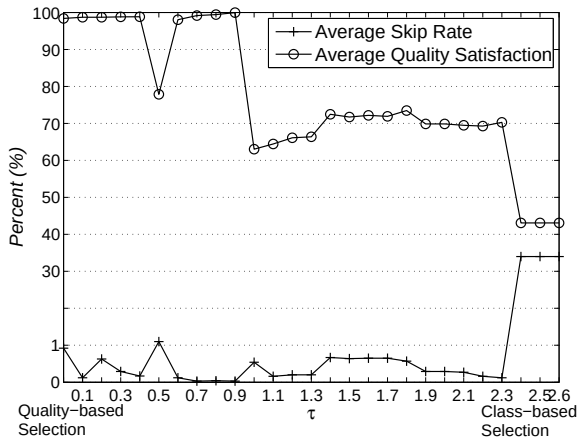


Fig. 6. The effect of the neighbor selection methods on Chameleon.

Figure 6 shows two important insights regarding the selection methods. *First*, quality-based selection is better than class-based selection, and the hybrid approach is the best. The reason is that the quality-based method reflects the peer situation better than the class-based method. A peer who has average quality Q likely belongs to class $\lceil Q \rceil$ or above, while a peer who belongs to class C may not perceive an average quality level up to C, due to the content or bandwidth bottleneck of its neighbors. However, the average quality level only reflects the quality level a peer has experienced so far, and it may be very low compared to the peer class. Consequently, a strict quality-based selection with very small value of τ may "trap" a high capacity peer within an area of low capacity peers. By using a larger τ, high capacity peers that are currently experiencing low quality have opportunities to connect to other high capacity peers, thanks to the class-based selection. Therefore, the hybrid approach offers better performance. *Second*, there is a sweet spot for the value of τ that should be set appropriately to achieve best performance, *e.g.*, $\tau = 0.7, 0.8,$ or 0.9 in Figure 6.

Finally, in an unstructured overlay, the topology is formed by the neighbor selection at each peer. There are no global mechanisms to create and maintain the overlay structure. An interesting question here is: *what does the topology look like*

under the neighbor selection method? To answer the question, for every peer of class C, we calculate the percentage of its neighbors of class C_k, $k = 1, 2, 3, 4$. In this experiment, the network size is 700, every peer has an average of 50 neighbors. The hybrid neighbor selection with $\tau = 0.7$ is used. Table I shows the neighboring relationships between peer classes in Chameleon. The value at element (i, j), $T(i, j)$, is the average percentage of peers of class j in the neighbor lists of peers of class i. The value in the parentheses at (i, i) is the percentage of peers of class i in the network. As shown in Table I, the neighbor selection method creates clusters of peers that belong to the same class: $T(i, i) \geq T(i, j), \forall i, j$. This can be considered as an *emergent property* of Chameleon, because each peer selects its neighbors with partial knowledge about the network.

TABLE I
PEER CLUSTERING IN CHAMELEON.

Peer class	1	2	3	4
1	**74**(21)	23	2.5	0.5
2	24	**31**(20)	28	17
3	2	26	**48**(24)	24
4	0.5	10	17	**72.5**(35)

2) Quality Adaptation: In Chameleon, adaptation is mainly based on the current status of the playback buffer. In our first design, we divide the playback buffer into two regions by a threshold drop_threshold. The adaptation process is invoked when the status of the buffer changes, *i.e.*, when the downloading of one segment is finished, or when a segment is played. A peer updates the current quality level, which is the target level for the next segment as follows. If the number of playable segments in the playback buffer (the buffer level) is below drop_threshold, the current quality level is decreased by one. Otherwise it is increased by one, but limited by the highest quality level for that peer's class. The intuition of increasing the quality level here is that we expect a peer will achieve its best possible quality as fast as possible. However, we have experienced fluctuations of the perceived quality because the number of segments in the buffer may vary around the threshold. To stabilize the perceived quality, we use another threshold, add_threshold. If the number of segments is greater than add_threshold, the current quality level is increased by one. Otherwise, it is unchanged. The algorithm of the adaptation process is shown in Algorithm 1, and the playback buffer with the two thresholds is illustrated in Figure 7.

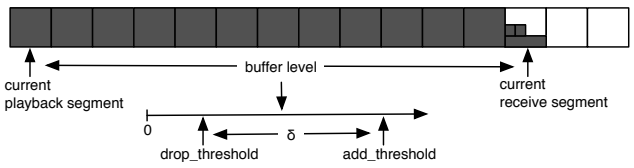


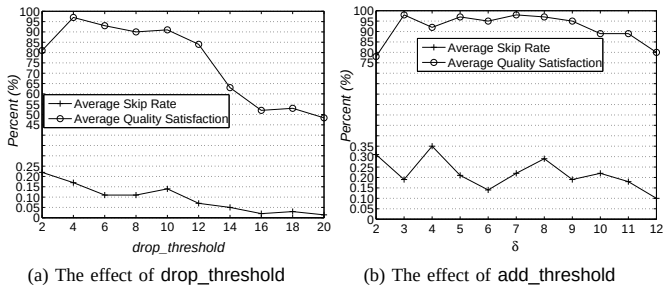
Fig. 7. The playback buffer in Chameleon: The dark shade indicates the receiving status of each segment.

To understand the effect of the two thresholds on Chameleon, we carry out two experiments with different values of drop_threshold and add_threshold. In both cases,

Algorithm 1 Quality Adaptation

```
CL: current quality level.
plb_seg: the segment ID of the current playback segment.
rec_seg: the segment ID of the segment being downloaded.
if (rec_seg − plb_seg < drop_threshold) then
  if (received(QL0, rec_seg)) then
    CL ← CL − 1;
    rec_seg ← rec_seg + 1;
  end if
else if (rec_seg − plb_seg > add_threshold) then
  if (received(CL, rec_seg) ∧ CL < QL_MAX) then
    CL ← CL + 1;
    rec_seg ← rec_seg + 1;
  end if
end if
```

the buffer size is set to 20. *First*, we vary drop_threshold from 2 to 20, and add_threshold is set to (drop_threshold+6). Figure 8(a) shows the performance of Chameleon with different values of drop_threshold. We observe a tradeoff between skip rates and quality satisfaction when increasing drop_threshold: both skip rates and quality satisfaction are reduced. This can be explained as follows. On one hand, a peer may try to maintain/increase the current video quality by using a low drop_threshold. However, the risk is that the skip rate may be increased because the playback buffer is exhausted rapidly when the buffer level reaches the low drop_threshold. On the other hand, if the peer is more "*conservative*" by using a higher drop_threshold, it is willing to drop the current layer and moves to the next segment to minimize the skip rate. As a result, the experienced quality may be lower than expected. When drop_threshold $= 20$ (the buffer size), all peers are very conservative, and they receive only the base layer.



(a) The effect of drop_threshold (b) The effect of add_threshold

Fig. 8. The effect of the quality adaptation parameters on Chameleon.

Second, we set drop_threshold $= 6$, and add_threshold =drop_threshold+δ, $\delta = 2, ..., 12$. In Figure 8(b), we also observe a tradeoff between the skip rate and the quality satisfaction when δ is greater than 8. Intuitively, if the current buffer level is between the two thresholds, the peer keeps its current quality level. In other words, the larger δ is, the more conservative the peer is. Consequently, the skip rate and quality satisfaction are decreased with a large δ. However, the relation between the two performance metrics is not very clear in the range from 2 to 8. The performance fluctuation can be explained as follows. Since high capacity peers often fill up the buffer faster than low capacity peers, it is likely

that δ should be smaller for high capacity peers so that they can quickly achieve their best possible quality. On the other hand, low capacity peers should use larger δ to keep the skip rate low. However, we currently use the same δ for all peers. The performance could be improved if different values of δ are used appropriately for different peer classes. We leave this feature to our future work.

To observe the quality adaptation process, we vary the download capacity of a typical peer P (not connected to the server) and examine its playback graph, which shows the quality level of all video segments that have been played. The download capacity can be varied slightly ($\pm 10\%$ of the streaming rate of the current quality level) or extremely (to another quality level). We randomly generate C points of time when the download capacity is varied extremely within the period of 10 minutes in the middle of the streaming session. The playback graph of peer P for $C = 15$ is shown in Figure 9, which demonstrates that Chameleon adapts to the bandwidth variations well. Minor variations are covered by buffering, while major variations are adapted accordingly. There is only one playback skip, which occurred when the download capacity drops below the streaming rate of the base layer (point 1). The figure also shows that adaptation takes effect few seconds after a significant variation occurs (point 2), *i.e.*, the perceived quality graph is a little shifted to the right of the available download capacity graph at the changing points. This is also because of the buffering effect. When the bandwidth capacity changes, there may be segments in the buffer which have been received before; and the bandwidth variation only takes effect when the buffer level reaches the thresholds.

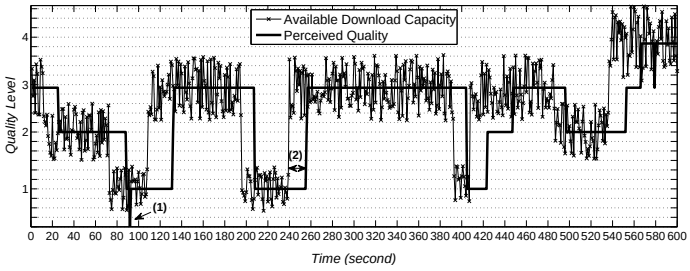


Fig. 9. An example of the playback graph of a typical peer.

3) Receiver-Driven Peer Coordination: Chameleon follows a receiver-driven approach to coordinate peers. A peer actively sends requests to senders. However, the requests are sent at the layer level (not at the block/packet level as in traditional approaches). In addition, all senders receive the same requests, and serve the receiver collaboratively. The receiver does not need to assign packets to each sender separately. The coordination mechanism at the receiver side and the sender side is as follows:

Each receiver:

- ▷ sends requests for the lowest unavailable layer to all senders.
- ▷ progressively decodes arrived blocks.
- ▷ when having received enough linearly independent blocks, sends a stop notification (via buffer maps) to the senders, and finishes the decoding process.

Algorithm 2 Receiver-side

```
PS: list of potential senders.
N: number of NC blocks necessary for decoding.
NumberOfReceivedBlocks ← 0;
L ← getLowestUnavailableLayerID();
newPS ← chooseSenders(PS, L);
sendRequest(newPS, L);
while (NumberOfReceivedBlocks ≤ N − 1) do
  receive(B);
  decode(B);
  if (linearlyIndependent()) then
    NumberOfReceivedBlocks++;
  end if
end while
sendStopNotification(newPS);
```

Each sender:

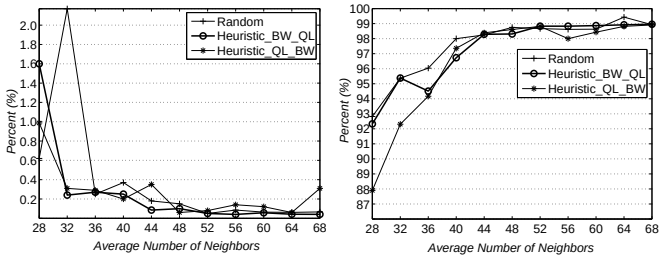
- ▷ on receiving a request, performs network coding on available blocks of the requested layer, and sends newly coded blocks to the requesting peers automatically and continuously as soon as possible.
- ▷ on receiving a stop notification, stops sending.

Algorithm 3 Sender-side

```
R: buffer map message.
RL: requested layer (from R).
while (active) do
  receive(R);
  if (isStopNotification(R)) then
    break;
  else
    encodedBlock ← encodeAvailableBlocks(RL);
    sendToReceiver(encodedBlock);
  end if
end while
```

4) Random-based vs. Heuristic-based Sender Selection: In the above peer coordination mechanism, a peer chooses senders from its neighbors to send layer requests. The first criterion for choosing a sender is that if it can provide coded blocks for the requested layer. This information is available in the buffer maps. If the number of potential peers is greater than the number of connections the peer can create, it needs to choose a subset. The simplest way is to choose a subset randomly. However, choosing senders randomly may not optimally utilize available bandwidth capacity and layers of the senders. For example, the download capacity of a peer is 150 Kbps. Two potential senders $S1$ and $S2$ who are able to create only one more connection have the upload capacity of 150 Kbps and 200 Kbps, respectively. The *random-based* method may choose $S2$ and render 50 Kbps unused, while choosing $S1$ is obviously a better utilization of bandwidth. A similar rationale is also applied to choose senders based on available layers. Intuitively, we use the following heuristics when choosing senders for a peer P: (H1) prefer potential senders whose available upload capacity is closest to the

currently available download capacity of P, and (H2) prefer potential senders who have the smallest number of layers.



(a) Average playback skip rate (b) Average quality satisfaction

Fig. 10. The performance of Chameleon with different sender selection methods.

To investigate benefits of the heuristics, we compare the random-based method (Random) with two heuristic-based methods (Heuristic_BW_QL and Heuristic_QL_BW). Both Heuristic_BW_QL and Heuristic_QL_BW use the aforementioned heuristic rules but in different order. Heuristic_BW_QL uses the priority order H1, H2, and random; while Heuristic_QL_BW uses H2, H1 and random. We plot the playback skip rate and the average quality satisfaction of the selection methods in Figure 10. The average number of neighbors (ν) each peer maintains is varied from 28 to 68, and the network size is set to 400. It is reasonable that the performance gets better when ν increases because it is more likely that a peer can choose good senders in a big neighbor list than in a small one. The performance is stable when ν is greater than a specific value, *e.g.*, 48. However, it is quite surprising that Chameleon achieves impressive performance with the random-based method. There are no significant differences between the three methods, especially when ν varies from 48 to 64. Heuristic_BW_QL seems to be the best one with respect to both the skip rate and quality satisfaction.

VI. Performance Evaluation

In this section, we evaluate the performance of Chameleon by comparing it with FABALAM [7], used as a benchmark. We implement Chameleon and FABALAM in our own discrete-event flow-based simulator developed from scratch. To make the simulator more realistic, we propose and implement an extended version of the max-min fair rate allocation described in [21]. The original version models the network as an undirected graph and only works with generic links without distinguishing uplinks and downlinks. Our extended version calculates the allocated rates for every uplink and downlink of a peer. The latest JSVM Software, Version 9.17, is used to generate a real two-hour video sequence with four quality levels. The average bit rate of the (sub-)stream with quality level up to 1, 2, 3, and 4 is 620, 825, 945, and 1065 Kbps respectively.

The main configuration parameters related to the quality scalability used in this paper are presented in Table II. The download and upload capacity of each peer are determined based on the stream rate at different quality levels of the test sequence. This setting is to reveal the benefit of SVC: different

TABLE II
Main configuration parameters used in the simulation.

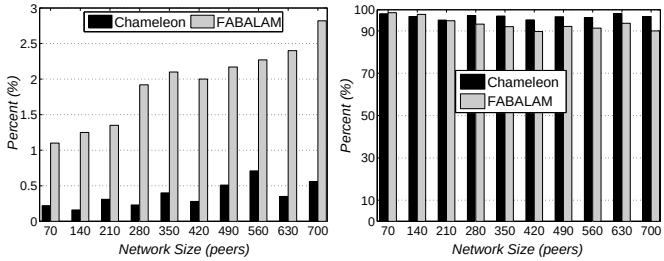
Configuration File	Parameter	Value
main.cfg	BaseLayerMode	2
	MGSControl	2
	NumLayers	2
layer0.cfg	MGSVectorMode	0
	QP	34
	MeQP0-MeQP5	32
layer1.cfg	MGSVectorMode	1
	MGSVector0	4
	MGSVector1	4
	MGSVector2	8
	QP	30
	MeQP0-MeQP5	30

bandwidth capacities perceive different quality levels. With the test sequence above, we use four peer classes (corresponding to the four quality levels) in which the download and upload capacity of each peer of class Q are set to 8-12% and 6-10% higher than the stream rate at quality level Q, respectively. Each peer is randomly assigned to a peer class. The server upload capacity is set so that it can serve 8-10% of the total number of peers, and we use only one server in our experiments. There are no super peers in the system. We use the following metrics to evaluate Chameleon:

- ▷ *Playback skip rate*: the percentage of segments skipped during playback.
- ▷ *Average quality satisfaction*: the average quality satisfaction of the system. The quality satisfaction of a peer is the ratio of the average quality level of played segments to its expected quality level (corresponding to its class).

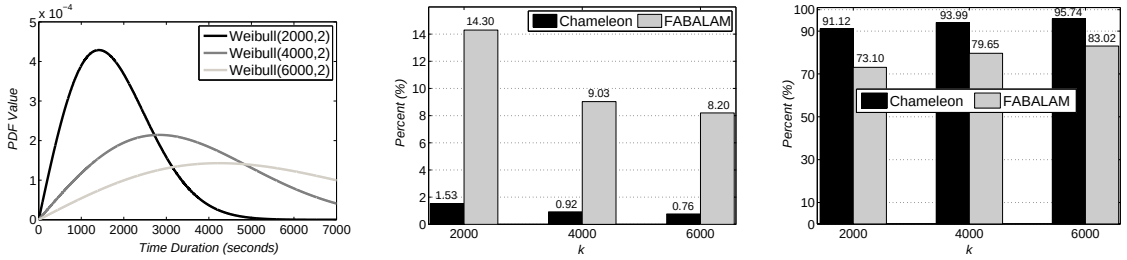
A. Scalability

We first compare Chameleon with FABALAM on the system scalability and performance in stable environments by varying the number of peers from 70 to 700. Peers join the network randomly, and stay connected until the session ends. Figure 11(a) shows that Chameleon achieves very low skip rates: 70-80% lower compared to the benchmark and less than or about 0.5% for various network sizes. Regarding quality satisfaction, Chameleon offers very good and stable quality satisfaction when the network size increases. In Figure 11(b), the quality satisfaction of Chameleon is always greater than 90% which means that peers can enjoy 90% of the best possible quality according to their download capacity. The system scalability is demonstrated by the stable performance when increasing the network size.



(a) Average playback skip rate (b) Average quality satisfaction

Fig. 11. The performance of Chameleon and FABALAM in different network sizes.



(a) Weibull distribution for peer lifetime (b) Average playback skip rate (c) Average quality satisfaction

Fig. 12. The effects of peer dynamics on Chameleon.

B. Coping with Peer Dynamics

To evaluate the performance of the protocols under peer dynamics, we use the Weibull distribution — Weibull$(k, 2)$ — to randomly generate the lifetime of peers, as shown in [22] that the peer session lengths are best captured by the Weibull distribution. With a two-hour streaming session, we use three different values of $k = 2000, 4000,$ and 6000 to generate different mean lifetimes. The lower the value of k is, the more volatile the session becomes. The plot of each distribution is shown in Figure 12 for clarity, together with the skip rate and quality satisfaction of the protocols. In this experiment, the network size is 350.

Figure 12 shows that Chameleon can adapt to peer dynamics well to achieve stable performance, whereas the performance of FABALAM is much impacted by peer dynamics. The reason is that FABALAM suffers the "rarest piece" problem. Without network coding, when a sending peer leaves, the receiver needs to receive exactly the blocks that were assigned to that sender. However, with network coding, the receiver can receive any blocks as long as they are linearly independent with those that have been received so far. In other words, thanks to network coding, Chameleon is robust to peer dynamics. This is demonstrated in the case $k = 2000$ (highly dynamic), the skip rate of Chameleon is only 1.53%, almost ten times lower than that of FABALAM; while the quality satisfaction is still high, up to 91.12%.

VII. Conclusion

In this paper, we present the design and the performance evaluation of Chameleon, our new adaptive P2P streaming protocol that combines the advantages of network coding and SVC. The objective of this paper is to design and preliminarily test a practical adaptive P2P streaming protocol, by taking advantage of network coding and SVC to mitigate the inherent challenges in unstructured layered P2P streaming. With Chameleon, we demonstrate that the combination of network coding and SVC is feasible and beneficial. Network coding helps to simplify the streaming protocol and improve the system performance. Detailed studies in the design space of Chameleon also bring interesting and useful results in building an adaptive P2P streaming system in practice.

References

[1] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the Scalable Video Coding Extension of the H.264/AVC Standard," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 17, no. 9, pp. 1103–1120, Sep. 2007.

[2] M. Wien, H. Schwarz, and T. Oelbaum, "Performance Analysis of SVC," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 17, no. 9, pp. 1194–1203, Sep. 2007.

[3] T. Oelbaum, H. Schwarz, M. Wien, and T. Wiegand, "Subjective Performance Evaluation of the SVC Extension of H.264/AVC," in *Proc. of 15th IEEE International Conference on Image Processing (ICIP)*, Oct. 2008, pp. 2772–2775.

[4] X. Zhang, J. Liu, B. Li, and T.-S. Yum, "CoolStreaming/DONet: a Data-Driven Overlay Network for Peer-to-Peer Live Media Streaming," in *Proc. of IEEE INFOCOM*, vol. 3, March 2005, pp. 2102–2111.

[5] M. Wang and B. Li, "R^2: 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, Dec. 2007.

[6] Y. Cui and K. Nahrstedt, "Layered peer-to-peer streaming," in *Proc. of NOSSDAV*, Jun. 2003, pp. 162–171.

[7] Y. Liu, W. Dou, and Z. Liu, "Layer Allocation Algorithms in Layered Peer-to-Peer Streaming," in *Proc. of IFIP international conference on network and parallel computing (NPC)*, Oct. 2004, pp. 167–174.

[8] R. Rejaie and A. Ortega, "PALS: Peer-to-Peer Adaptive Layered Streaming," in *Proc. of NOSSDAV*, Monterey, CA, USA, jun 2003, pp. 153–161.

[9] N. Magharei and R. Rejaie, "Adaptive Receiver-Driven Streaming from Multiple Senders," *Multimedia Systems*, vol. 11, no. 6, pp. 550–567, Jun. 2006.

[10] S. Y. R. Li, R. W. Yeung, and N. Cai, "Linear Network Coding," *IEEE Trans. Info. Theory*, vol. 49, no. 2, pp. 371–381, Feb. 2003.

[11] S. Annapureddy, S. Guha, C. Gkantsidis, D. Gunawardena, and P. R. Rodriguez, "Is High-Quality VoD Feasible using P2P Swarming?" in *Proc. of the 16th international Conference on World Wide Web (WWW)*, Aug. 2007, pp. 903–912.

[12] ITU-T and I. JTC1. (2008) Jsvm software version jsvm 9.17. [Online]. Available: http://ip.hhi.de/imagecom_G1/savce/downloads/SVC-Reference-Software.htm

[13] N. Magharei and R. Rejaie, "PRIME: Peer-to-Peer Receiver-drIven MEsh-Based Streaming," in *Proc. of IEEE INFOCOM*, May 2007, pp. 1415–1423.

[14] J. Zhao, F. Yang, Q. Zhang, Z. Zhang, and F. Zhang, "LION: Layered Overlay Multicast With Network Coding," *IEEE Trans. on Multimedia*, vol. 8, no. 5, pp. 1021–1032, Oct. 2006.

[15] M. Wang and B. Li, "Lava: A Reality Check of Network Coding in Peer-to-Peer Live Streaming," in *Proc. of IEEE INFOCOM*, May 2007, pp. 1082–1090.

[16] H. Shojania and B. Li, "Parallelized Progressive Network Coding With Hardware Acceleration," in *Proc. of Fifteenth IEEE International Workshop on Quality of Service*, June 2007, pp. 47–55.

[17] I.-T. R. H.264, I.-T. ISO/IEC 14496-10 (MPEG-4 AVC), and V. . ISO/IEC JTC. (2007, July) Advanced Video Coding for Generic Audiovisual Services.

[18] S. Wenger, Y.-K. Wang, and T. Schierl, "Transport and Signaling of SVC in IP Networks," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 17, no. 9, pp. 1164–1173, Sep. 2007.

[19] Y. Wang, M. Hannuksela, S. Pateux, A. Eleftheriadis, and S. Wenger, "System and Transport Interface of SVC," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 17, no. 9, pp. 1149–1163, Sep. 2007.

[20] A. Ganesh, A.-M. Kermarrec, and L. Massoulie, "Peer-to-peer membership management for gossip-based protocols," *IEEE Trans. on Computers*, vol. 52, no. 2, pp. 139–149, Feb. 2003.

[21] F. L. Piccolo, G. Bianchi, and S. Cassella, "QRP03-4: Efficient Simulation of Bandwidth Allocation Dynamics in P2P Networks," in *Proc. of Global Telecommunications Conference (GLOBECOM)*, Dec. 2006, pp. 1–6.

[22] D. Stutzbach and R. Rejaie, "Understanding Churn in Peer-to-Peer Networks," in *Proc. of the 6th ACM SIGCOMM conference on Internet measurement (IMC)*. New York, NY, USA: ACM, 2006, pp. 189–202.