Towards Optimal Resource Utilization in Heterogeneous P2P Streaming

Dongyu Liu   Fei Li   Songqing Chen
Department of Computer Science, George Mason University
Email: {dliu1, lifei, sqchen}@cs.gmu.edu

Abstract

Though plenty of research has been conducted to improve Internet P2P streaming quality perceived by end-users, little has been known about the upper bounds of achievable performance with available resources so that different designs could compare against. On the other hand, the current practice has shown increasing demand of server capacities in P2P-assisted streaming systems in order to maintain high-quality streaming to end-users. Both research and practice call for a design that can optimally utilize available peer resources. In the paper, we first present a new design, aiming to reveal the best achievable throughput for heterogeneous P2P streaming systems. We measure the performance gaps between various designs and this optimal resource allocation. Through extensive simulations, we find out that several typical existing designs have not fully exploited the potential of system resources. However, the control overhead prohibits the adoption of this optimal approach. Then, we design a hybrid system in trading off the cost of assignment and utilization of resources. This hybrid approach has a proved theoretical bound on efficiency of utilization. Simulation results show that compared with the optimal resource allocation, our proposed hybrid design can achieve near-optimal (up to 90%) utilization while only use much less (below 4%) control overhead. Our results provide a basis for both server capacity planning in current P2P-assisted streaming practice and future protocol designs.

1. Introduction

With the increasing number of end-users subscribing to the broadband Internet access services, such as cable and DSL, the improved last-mile bandwidth have made Internet media streaming possible. As a result, recent years have witnessed the rapid increase of Internet media traffic. For example, according to ComScore [1], Internet users viewed more than 12 billion online videos in May 2008, representing an increase of 45 percent from the prior year. A number of different P2P/overlay streaming systems, such as PPLive [3], PPStream [4], SopCast [5], TVAnts [6], ESM [9], and Cool-Streaming [22], have contributed to this significant media

traffic increase. For example, as a typical P2P streaming system, PPLive [3] provides daily accesses to more than 400 channels for millions of Internet users.

Plenty of research has been conducted to study and improve existing systems, focusing on issues such as topologies, free riders and incentive mechanisms, peer churns, file dissemination, and asymmetric bandwidth of peers. Though existing protocol designs for P2P streaming have provided a rich set of solutions to system stability, scalability, fairness, and/or peer perceived streaming quality, prior research has paid little attention to optimal resource utilization achievable with available peer resources. However, such an optimal result is important since it gives insights on how good existing systems have been and it provides a performance target that future design/improvement could aim at. On the other hand, current practice on P2P-assisted live and on-demand streaming systems have shown that although the P2P approach can significantly reduce server capacity demand on average [13], a calling for more server capacities accompanies with the increasing number of users [12], [19]. This indicates that optimal utilization of available peer resources could potentially mitigate the bottleneck of server capacity and thus support more users.

While both research and practice call for exploring all potentials of utilizing available peer resource, the problem is different from and orthogonal with incentive studies. Although incentive mechanisms [20], [15], [23] could help improve the system throughput, fundamentally, they aim to increase the available resources in the system. On the contrary, we seek to fully and optimally utilize available peer resources in the system in order to provider better streaming quality or reduce server capacity demand.

In this paper, aiming to unveil the best achievable resource utilization, we first provide a theoretical bound for optimizing resource allocation in heterogeneous environments. In our design, the least quality of any individual peer in the system is maximized. The crux of our design is a new admission control and resource allocation policy, in which each physical peer is divided into two entities, a *supply* and a *demand*. Accordingly, a bipartite graph is constructed and a linear program formulation returns a `max-min` resource allocation solution. Unfortunately, this optimal resource assignment invites overhead if peers frequently join in and depart from the system. Then, we propose a hybrid archi-

tecture to increase the system scalability (which inherently reduces the overhead) while achieving a provable near-optimal performance in heterogeneous streaming systems. Our Simulation results show that our proposed hybrid design can achieve near-optimal (up to 90%) utilization while only incurs less than 4% control overhead. We believe our results can shed light on future protocol designs as well as server capacity planning.

In the remaining part of this paper, we first introduce the background of resource allocation in P2P streaming systems. Then we discuss the core of our algorithm design as well as its approximated solution to reduce computation and control overhead in Section 3. We present the protocol design in Section 4. Extensive simulation results are presented and discussed in Section 5. Concluding remarks are in Section 6.

2. Resource Allocation in Streaming Systems

While many P2P/overlay based streaming systems have been deployed in practice and lots of research has been actively conducted on various issues in these systems, resource allocation and utilization have mainly been addressed from the fairness perspective. These schemes themselves have different resource allocation objectives and optimize the utilization of available resources mainly through admission control. The admission control strategies in existing systems can be roughly classified into the following four categories: `Best-fit`, `Random-selection`, `Preemption-based`, and `Reputation-based` algorithms.

The `Best-fit` algorithms, for example, used in P2Cast [21] and ESM [10], aim to find several existing peers with the largest available resource (in terms of bandwidth) for this newly arriving peer. At its arrival, each peer selects peers with the largest available resources to maximize its downloading throughput.

Upon the arrival of a new peer, the `Random-selection` algorithm often chooses a number of peers that can fulfill the bandwidth requirement of the new one. In [16], [18], [22], a partial list of existing peers may be built from the centralized control node and the new peer selects the downloading peers randomly from the list.

`Preemption-based` algorithms [17], [10] generally first choose a number of peers as the downloading peers of the joining peer. If these downloading peers do not have sufficient bandwidth and the resource supply of the joining peer is larger than some existing nodes, some of the existing nodes in the system will be preempted and their preempted resource will be reallocated to this newly joining one.

`Reputation-based` algorithms [15], [20] provide different amount of resources to existing peers according to their reputations (= ranks). As all peers' reputation is determined over history, in general, it takes quite long time to reach a stable and precise reputation rank and their rankings are sensitive to the dynamics of the system.

Apparently, these admission control strategies allocate available system resources in a very coarse granularity from an individual peer's interests but not from the system's perspective. While these algorithms have greatly simplified the resource allocation issues and system implementation, a significant portion of available resources have potentially been wasted. In this paper, we aim to provide a theoretical bound, as well as a system design prototype, for future P2P streaming systems such that we could reach the best tradeoff in maximizing utilization of the system resources, maximizing an individual peer's interest, and minimizing overhead due to resource re-assignment.

3. Dynamic Resource Allocation

As aforementioned, it has not been well studied how to maximize throughput of a P2P streaming system in heterogeneous environments with the given resources while the least qualities perceived by peers are maximized simultaneously. In this section, we first formulate such a problem from the system's perspective. Then, we provide a solution maximizing the least quality perceived by a peer based on dynamic resource allocation. An improved algorithm is developed, and it reaches a near-optimal resource utilization of the system.

3.1. Problem Statement

In a heterogeneous P2P streaming system, there are a number of peers demanding as well as providing services. The best quality requested by a peer could be different from others. Thus, we should not use the absolute streaming quality received by a peer to evaluate the system. Instead, we use *relative quality* (which is defined later) to characterize how much a user is satisfied with the streaming service. Such a definition can be generalized to cover homogeneous environments as well.

From the system's perspective, the problem we consider is how the system should allocate resources such that the least quality perceived by the peers is maximized. The importance of this criteria lies that the least quality perceived by the peers reflects the fairness among peers. To initialize the resource allocation, each peer should declare the resource it is willing to contribute when it joins in the system.

To precisely characterize this problem, we have the following notations. At each peer i's arrival, we assume that i has a *resource demand* q_i and a *promised (declared) supply* c_i. Assume peer i arrives at time r_i and leaves at time e_i: q_i and c_i are revealed at time r_i. e_i is not known until the peer really leaves. When peer i arrives, it also actively or passively declares its device type. The visual streaming

quality for a peer is proportional to the resource that it gets with respect to the maximum quality that can be achieved for that type. To allocate appropriate resources to each peer, we view each peer as 2 non-overlapping entities: *resource demand* q_i and *resource supply* c_i. q_i is regarded as the best quality that the peer i would like to have. We may adjust q_i to reduce or increase the service quality that this peer i actually receives, based on the actual supply this peer provides to the system. Therefore, for peer i, in contrast to what this peer has requested (q_i) and promised (c_i), we use q_i' (which is always $\leq q_i$) and c_i' to denote the estimated actual resource assigned and the estimated actual resource contributed, respectively.

3.2. Maximize the Least Perceived Quality

With above problem formulation, we illustrate our resource allocation algorithm to the utilization problem in the following.

Without loss of generality, at time t, we assume that there are n peers in the system. We build up a bipartite graph: the left column consists of n peers' estimated actual supply c_i' and the right column consists of n peers' resource demand q_i', $i = 1, 2, \ldots, n$. We periodically measure the systems' resource requests and resource demands. For those peers j who join in the system after our last time of measuring resource requests/demands, we simply set their q'-values and c'-values as their q-values and c-values, respectively.

We build up an undirected edge connecting each q_j' and each c_i', for any $i \neq j$. Let the weight over this edge be w_{ij}, denoting the resource (such as uploading bandwidth) provided from peer i to peer j. Since we do not know yet whether the supplied resources are enough to satisfy the peer demands, we further construct one super node S to reflect the deficiency of the resources in the system. We build up edges over each q_j' with S. The weight of the edge over q_j' and S is w_{Sj}. w_{Sj} denotes the extra *virtual resource* that should be assigned to j, in addition to what has been provided by other peers in the system. In general, this value should be 0 to ensure that peer j actually gets its satisfactory quality q_j'. Introducing the super node S guarantees that we always have a solution to our problem. In order to make each peer j receive an acceptable quality of streaming service, i.e., q_j', we have

$$\sum_i w_{ij} + w_{Sj} \geq q_j', \quad \forall i, j = 1, 2, \ldots, n, \text{ and } i \neq j.$$

Since generally a peer i contributes no more than what it promises to contribute, we have

$$\sum_i w_{ij} \leq c_i', \quad \forall i, j = 1, 2, \ldots, n, \text{ and } i \neq j.$$

Considering that a peer cannot serve itself, for each single pair peer i and peer j, $i \neq j$, we constrain

$$w_{ij} \geq 0, \ w_{Sj} \geq 0, \ w_{ii} = 0, \quad \forall i, j = 1, 2, \ldots, n.$$

Notice that if $w_{Sj} > 0$, peer j may not be getting satisfied service quality in the system. (As S is the super node we introduce, S cannot contribute any real resource to j.) Consider the heterogeneity of the streaming quality demanded by a peer, we define the service quality as the *relative quality* as follows:

Definition 1: **Relative quality.** For each peer i, its *relative quality* is defined as $(q_j' - w_{Sj})/q_j' (= 1 - w_{Sj}/q_j')$.

Thus, our objective is

$$\max \min (1 - \frac{w_{Sj}}{q_j'}), \quad \forall j = 1, 2, \ldots, n.$$

Equivalently, our objective can also be presented as

$$\min \max v_j,$$

where $v_j = w_{Sj}/q_j'$ ($0 \leq v_j \leq 1$).

Here, we have a linear program $\mathcal{LP}$. We start from some value v (an option is to initialize it as 0). Then, we use the binary search approach to approximate v as much close as the optimal solution by identifying whether the following set of linear constraints has a feasible region or not.

$$
\begin{aligned}
\mathcal{LP}: \quad & \text{is } v \text{ feasible?} \\
\text{subject to} \\
\sum_i w_{ij} + w_{Sj} &\geq q_j', \quad \forall i, j = 1, 2, \ldots, n \\
\sum_i w_{ij} &\leq c_i', \quad \forall i, j = 1, 2, \ldots, n \\
w_{Si} &\leq v \cdot q_i', \quad \forall i = 1, 2, \ldots, n \\
w_{ij} &\geq 0, \quad \forall i, j = 1, 2, \ldots, n, \ i \neq j \\
w_{ii} &= 0, \quad \forall i = 1, 2, \ldots, n \\
c_i', \ q_i', \ w_{Si} &\geq 0, \quad \forall i = 1, 2, \ldots, n
\end{aligned}
$$

The constraints are self-explained. If there exists a feasible region when $v = v_1$ and there is not a feasible region when $v = v_2$, we know that the optimal value falls in the range $[v_1, v_2]$. Using binary search, we reach a value v such that the solution of $\mathcal{LP}$ returns the optimal solution of maximizing the least quality of peers. Note that if we would like to keep a peer j to receive a quality no worse than q_j', we just simply set $w_{Sj} = 0$ in above $\mathcal{LP}$. However, in this way, $\mathcal{LP}$ may not have a feasible solution. We extend this kind of discussion in the protocol design.

Before we proceed to the protocol implementation based on $\mathcal{LP}$, we realize that in the worst-case scenarios above $\mathcal{LP}$ should be invoked upon each peer's arrival or departure, which is intolerable in practice. For example, the computation time to solve $\mathcal{LP}$ with 100 (resp. 800) peers using

`Mosek` [2] (a third-party Linear Programming solver package) is 0.39 (resp. 379.40) seconds. The experiment is run on machines of Pentium D CPU 2.8 GHz with 1 GB memory and the algorithm for linear programming is the *interior-point method*. When the number of peers is increased, the computation overhead is increased significantly.

The overhead due to solving $\mathcal{LP}$ involves in each peer's join and departure. Frequently updating the connections among peers but not improving their qualities is unacceptable. Hence, we would call for a much less time-consuming, more stable solution, which does not run a linear programming over all peers in the system but results with near optimal resource utilization. Moreover, to reduce the cost of centralized management and to increase the scalability of the P2P streaming system, we prefer to design a distributed system, or at least a hybrid system without losing the utilization of resources much. There are two ways to reduce the cost of running a linear program: (1) reduce the number of variables, or (2) reduce the number of *rounds* or *phases* when running the algorithms for convergence. In our paper, we apply a mixed approach to reduce the overhead of solving $\mathcal{LP}$.

Conceptually, we employ a few *representative entities* and each of them represents a *cluster* of demands/supplies with size of the cluster proportional to $\sqrt{n}$. Here, each peer has two entities: *resource supply* and *resource demand*. Each entity of a peer (but not the peer itself) appears in the clusters, but a peer may have its supply entity and demand entity in two different clusters. Based on this data structure, we allow resource utilization calculation run in a parallel manner to reduce the number of variables involved in the linear program $\mathcal{LP}$. We take the approach developed by Luby and Nisan [14] to run $\mathcal{LP}$ on a parallel machine using $O(\sqrt{n})$ processors where $c \cdot \sqrt{n}$ ($1/2 \leq c \leq 2$) is the number of entities in each cluster. The algorithm reaches a running time of $\log(\sqrt{n})$ with the performance bounded by at least $\sqrt{n}$ of the optimal one. Details of our algorithms follow.

Definition 2: **Balanced cluster.** Assume we have k clusters with sizes of $x_1, x_2, \ldots, x_k$, we say these clusters are *balanced* if

$$\frac{\sqrt{n}}{2} \leq k \leq 2 \cdot \sqrt{n} \text{ and } \max_{\forall i \neq j} \frac{x_i}{x_j} \leq 2. \quad (1)$$

Assume at time t, the system consists of n peers. Also, we construct a set of clusters such that the number of clusters could be up to $c \cdot \sqrt{n}$, where c is a constant. At any time, we keep these clusters balanced. There are two layers of clusters: *uploading layer* of resource supply entities and *downloading layer* of resource demand entities. A peer's resource supplier entity has its uploading "neighbors" in the same downloading cluster and a peer's resource demand entity has its uploading "neighbors" in the same uploading cluster. Constituting clusters involves two actions: *merging* and *splitting*.

Merging. When a peer's leaving a cluster results the number of peers in this cluster is $o(\sqrt{n})$ (i.e., if the second inequality in Eq. 1 is violated), the cluster will merge with another one with the smallest size to be into a larger cluster with size proportional to $\sqrt{n}$. Assume we have the two smallest clusters with sizes x_i and x_j, the largest cluster is x_l. Then when we assume the leaving peer is from the cluster with size x_i, the new cluster has a size of $x_i + x_j$ and it satisfies Eq. 1 (see below). That means whenever a peer leaves, at most one merging operation is needed. The assignment in this new (bigger cluster) will be adjusted using $\mathcal{LP}$.

$$
\begin{aligned}
x_i + x_j &\leq \min\{x_j + x_j,\ x_l/2 + x_l\} \\
&= \min\{2 \cdot x_j,\ 1.5 \cdot x_l\} \\
x_i + x_j &\geq x_j + 1 \geq x_l/2.
\end{aligned}
$$

Splitting. When a peer arrives, we investigate the relation between the number of existing clusters k and $n+1$ (assume before this arrival, there are n peers).

- If $\sqrt{n+1}/2 \leq k \leq 2 \cdot \sqrt{n+1}$, the new arriving peer distributes its demand entity and supply entity into two clusters with the most remaining supply and demand, respectively. For each new arriving peer, the splitting action is taken once.
- If $k \leq \sqrt{n+1}/2$, we split the largest cluster into two smaller ones: sort all demand and supply in increasing order of their values and after aligning them, split the set into two by halves.

  Assume the largest cluster (in terms of sizes) is with a size of x_l before this new arrival. Then assume it is separated into two smaller clusters with sizes of x_l^1 and x_l^2, then these clusters are balanced again.

$$x_l^1 + x_l^2 = x_l \text{ and } \min\{x_l^1,\ x_l^2\} \geq \frac{x_l}{2} \geq \frac{x_i}{2}, \forall i.$$

$$k \geq \frac{\sqrt{n}}{2} \text{ implies } k + 1 \geq \frac{\sqrt{n+1}}{2}.$$

Upon a new peer i's arrival, this peer has two disjunct entities: supply entity c_i' and demand entity q_i'. After contacting with each cluster and getting the information from that single cluster, then the peer puts its supply entity into the cluster deficient of resource most and puts its demand entity into the cluster with the most remaining resource (these two clusters are different). Since we have $c\sqrt{n}$ clusters for downloading and uploading, the maximum number of entities per cluster in the P2P system is up to $2 \cdot n/(c \cdot \sqrt{n} - 1)$ ($\leq 2 \cdot (\sqrt{n}/c + 3), \forall n \geq 4$) under the constraint of Eq. 1. We reduce the overhead of computation of running a linear program to find out the best quality for the peers. Upon each arrival, at most $c \cdot \sqrt{n}$ are affected and the $\mathcal{LP}$ runs over at most two clusters with size of $(2/c) \cdot \sqrt{n} + d$, where $d \leq 3$ is constant. Upon each departure, a merging procedure may be involved. The distributed version of $\mathcal{LP}$ significantly

reduces the computation overhead. The remaining part of our work is to prove that the resource utilization is not affected remarkably.

Theorem 1: [7] Let $w = w_1 + w_2$. If x is an r-approximate solution for w_1 and w_2 then x is r-approximate with respect to w as well.

Theorem 2: Above algorithm results a solution in which at most $\sqrt{n}$ peers get the optimal quality or all peers get quality at least $1/\sqrt{n}$ of the optimal ones.

Proof: Based on Theorem 1, we only consider the case when only one peer arrives or one peer leaves. Assume at time t, the clusters of uploading streams (supplies) are C_1, C_2, ..., C_i and the clusters of downloading streams (demands) are L_1, L_2, ..., L_j, where i, $j = c \cdot \sqrt{n}$.

Assume a peer arrives. We note that this peer gets the maximum remaining resource available for the clusters, and thus, its quality is at least $1/\sqrt{n}$ of the optimal ones. (The upper bound of the quality is to collect all remaining resource from $c \cdot \sqrt{n}$ clusters.) If the splitting procedure happens, at least $\sqrt{n}$ demand entities get their optimal quality, assuming the entities in the newly generated clusters are deficient at resources from the supply entities in the original cluster.

Assume a peer leaves. If the *merging* procedure has not happened, we claim that a peer, if its quality is not satisfied, will connect the remaining entities in the merged clusters to find more neighbors. Otherwise, with more entities, resource distribution will be more balanced (this is the convex property of $\mathcal{LP}$).

The new algorithm can be regarded as a sparse matrix. We limit the number of non-zero variables in each row and column bounded by $c \cdot \sqrt{n}$ and $\sqrt{n}/c$, respectively. □

4. Protocol Design

Based on what we have discussed in Section 3, we develop a set of protocols for peers in a P2P streaming system. In this protocol design, we also introduce mechanisms to avoid cheating from peers by associating a reputation system with them. Our protocol consists of three parts, including *peer arrivals*, *peer goodness adjustment*, and *peer departure*. In this section, we present the details of these parts one by one in the order.

Peer Arrival. When a peer j arrives, it sends a joining request to the bootstrapping peer. The bootstrapping peer examines the list of the peers with unused resources and sends back such connection information to the joining peer. When a peer j receives the response from the bootstrapping node, it specifies an inquiry, in a format of (c'_j, q'_j), and delivers to the set of nodes capable of accepting new peers. Among all the possible acceptable quality levels, q'_j is the best desirable quality for the peer, while c'_j is the declared resource supply from this peer. Here, the modified $\mathcal{LP}$

is run to identify the resource allocation with tolerable overhead of computation. We use $\mathbf{C}$ and $\mathbf{Q}$ to denote the sets of estimated supplies and estimated demands for the peers existing in the system, respectively. Let p be a new-coming peer. We initialize $c'_p = c_p$ and $q'_p = q_p$. The *Allocate_System_Resource* works as follows:

Algorithm 1 `Allocate_System_Resource`$(\mathbf{C}, \mathbf{Q}, p)$

1: Consider $\mathcal{LP}$ with all peers in the group and this new coming one p.
2: **for** a peer j is with goodness value `good` **do**
3:     $w_{Sj} = 0$.
4: **end for**
5: Solve $\mathcal{LP}$.
6: **if** $\mathcal{LP}$ does not have a solution with $v \geq 0$ **then**
7:     Flush all peers' goodness values to be `bad`.
      {Note that if p's goodness is `bad`, $\mathcal{LP}$ always have a solution since this convex polyhedron is not empty.}
8: **end if**
9: Accept p and update/notify peers the flow change among peers based on the value w_{ij}.

Peer Goodness Adjustment. The routine is used by the centralized system to keep track of peer's behavior in terms of whether they make enough supply as what they have promised when joining the system. As aforementioned, each peer has a goodness value, which is `bad` or `good`, depending on whether the peer has committed the supply it declared or not. Therefore, if a peer makes supplies the same as what it promised, it has a higher priority to be served with the desirable quality. Otherwise, the quality that this peer will receive depends on how much resource available in the system (in a best-effort manner).

To correctly monitor the behavior of different peers in the system, each peer needs to record who has been uploaded to it in the past as well as the amount of supply made by these peers periodically, and whether the service quality it receives is satisfying or not. For this purpose, each peer maintains a list of at most $\sqrt{n}$ peers that have uploaded to it with the corresponding service amount of traffic. Such information is collected passively or actively when it is time to perform dynamic resource allocation. Based on this principle, the *Peer Goodness Adjustment* is implemented with the following *Adjust_Peer_Goodness* for updating the goodness of peer j and its up-loader peer k.

Peer Departure. After a peer leaves, we invoke new resource allocation, which reflects the possible change of resource re-allocation caused by the departure of the peer. At a peer's departure, it sends a departing signal to its neighbors and the bootstrap node for possible merging operation. The bootstrap node also updates the goodness values of those neighbors of the leaving peer, if necessary. We run the modified $\mathcal{LP}$. Detailed of the procedure of peer departure

Algorithm 2 `Adjust_Peer_Goodness(`j, k`)`

1: Set a counter for each peer in the system in the recent l time.
2: Set a threshold for each peer's goodness: if in the last l time, a peer j is reported badly in uploading for more than s times, its goodness value is set to `bad`.
3: **if** j reports its poor quality due to k and k's counter $\geq s$ in the last l time **then**
4:     Set k's goodness `bad`.
5: **end if**
6: **if** A peer i's goodness is `bad` and its record is clean in the last l time **then**
7:     Set i's goodness `good`.
8: **end if**

---

has been discussed in Section 3.2.

5. Performance Evaluation

In this section, we evaluate the strength and weakness of the hybrid resource allocation scheme via measuring and comparing its performance with optimal allocation scheme and other four other typical protocols that employ representative admission control strategies.

We have implemented a simulator with a linear programming solver provided by a third-party linear programming package (Mosek 5.0) [2]. In the simulation, 1000 peers dynamically join into and leave from the system. These peers join in the system following a Poisson distribution with the mean arrival rate as 1 request per three second and the maximum arrival interval lasting 15 seconds. We assume every peer requires for a full-quality 500-second long video. A peer leaves the system either it gets the video or the cumulative relative quality (defined later) is below a threshold, which happens first.

When a node arrives, it declares its promised supply to the system and the demand for streaming quality. The resource supply and demand are termed in bandwidth. The peer quality demands are shown in Table 1, represented in heterogeneous bandwidth consumption. That distribution is based on what has been reported in [13], [8].

Table 1. Bandwidth Distribution

Upload (Kbps)	Percentage	Download (Kbps)	Percentage
[0, 200)	35.94%	[0, 360)	25.47%
[200, 360)	27.96%	[360, 600)	64.78%
[360, 600)	24.86%	[600, 1000)	8.01%
[600, 1000)	5.14%	[1000, 2000)	1.67%
[1000, 1500)	4.76%	[2000, 5000)	0.07%
[1500, 10000)	1.34%	-	-

As some peers may not keep their initial promises along their sessions in the system, the 1000 nodes fall into two categories: *truthful peers* who keep their promise, and *un-truthful peers* who do not keep their promise. In the experiments, the ratio of truthful and un-truthful nodes is set to $1:1$. When a node does not keep the promise, it selects a percentage between 40% and 80% randomly and contributes that percentage of its promised bandwidth. To bootstrap the system, a source node is assigned to provide $1000k$bps bandwidth resources.

We evaluate the performance of our design by comparing with systems using `Random-selection` [22], `Best-fit` [21], `Preemption-based` [17], and `Reputation-based` [20] admission algorithms as what we have discussed before, and we abbreviate these algorithms as `Random`, `Best-fit`, `Preemption`, and `Reputation`, respectively. We use `Optimal` and `Hybrid` to represent the optimal and our proposed hybrid protocol, respectively.

System Throughput. The system throughput is defined as the number of peers that the system can support with available resources. The total number of peers along this experiment is 1000. If the relative quality of the peer is less than 0.33 (upon its request quality set at 1), the peer is not counted as a peer the system supports. We observe that `Preemption` generates the lowest throughput among all these four systems. Precisely, 32.8%, 37.4%, 89.3%, and 91.8% of peers can be served by `Random`, `Preemption`, `Best-fit`, and `Reputation`, respectively. Both `Hybrid` and `Optimal` can support all peers in the system.

Relative Quality. While system throughput reflects the system overall throughput, it does not reflect each peer's perceived streaming quality. We measure the least relative quality and average relative quality perceived by all peers in different systems. The relative quality is defined by the ratio of the resource received and resource demanded by each peer. In case a peer is not accepted by the system, its relative quality is defined as 0; and a full-quality peer has relative quality 1 ($=100\%$). The least relative quality of a peer and the average relative quality peers in these six algorithms are listed in Table 2. We can see that the average relative quality of the hybrid scheme is less than that of the optimal algorithm, but the average relative quality of the hybrid scheme is still better than the other four schemes. The relative quality of the peers in `Optimal` is always above 83.9% and the relative quality of the peers in `Hybrid` is 79.7%. But these schemes are much better than those in the other four systems.

Resource Contribution and Utilization. The previous evaluation results show that in general, peers in the `Hybrid` scheme can achieve better quality than in other systems in terms of the streaming quality they receive as well as the overall system throughput. According to our design, `Hybrid` achieves a provable performance compared with

Table 2. Least and Average Relative Quality

	Least relative quality (%)	Average relative quality (%)
Best-fit	5.1	58.8
Random	8.5	41.2
Preemption	10.9	43.8
Reputation	2.9	60.2
Optimal	53.9	83.9
Hybrid	47.2	79.7

the optimal resource allocation. In the following, we study how efficiently resources are used in different systems.

In our measurement, the average supply in the `Hybrid` approach is similar to that in `Optimal` and they are better than that in other systems. The average values of peer supply in `Optimal`, `Hybrid`, `Reputation`, `Best-fit`, `Preemption`, and `Random` are $373.7k$bps, $353.6k$bps, $271.7k$bps, $250.7k$bps, $168.1k$bps, and $159.2k$bps, respectively. This partially explains why `Hybrid` and `Optimal` can serve more peers with higher relative qualities than the other four systems.

To better characterize the resource utilization in the systems, we calculate peers' utilization ratios (the ratio between the actually consumed resources and the actually contributed bandwidth along the experiment). Average utilization ratios are sorted in decreasing order as `Optimal`, `Hybrid`, `Reputation`, `Best-fit`, `Preemption`, and `Random` respectively. The corresponding utilization values are 100.0%, 94.0%, 72.3%, 66.7%, 44.7%, and 42.4%, respectively. The utilization rate for these six algorithms are listed in Table 3. We can claim that `Optimal` and `Hybrid` are far better than the other systems in terms of utilization.

Table 3. Comparison of Average Utilization Rate

	Utilization (%)	Min (%)	Max (%)
Best-fit	249.2$k$bps (66.7)	41.1 (15.2)	346.8 (73.1)
Random	158.4$k$bps (42.4)	16.0 (6.0)	210.6 (44.4)
Preemption	167.1$k$bps (44.7)	59.4 (22.0)	207.8 (43.8)
Reputation	270.0$k$bps (72.3)	41.5 (15.4)	396.2 (83.6)
Optimal	373.7$k$bps (100)	269.6 (100)	474.2 (100)
Hybrid	351.3$k$bps (94.0)	253.0 (93.8)	442.7 (93.4)

Peer Duration. In consideration of social welfare, such as system throughput, resource utilization, and supply, `Optimal` is the best and `Hybrid` has near-optimal performance. They perform much better than the other four systems. Now, we study peer duration, which is one main design objective for distributed P2P systems [11]. A peer's duration is the flow time it stays in the system for downloading streaming data.

To evaluation the flow time of peers, we sort peers' duration. In our measurement, if the peer is rejected during admission control or it cannot get the satisfied streaming relative quality, we set its duration to the maximal value: 3600. Our result demonstrates that `Optimal` has less stay duration

than `Hybrid` and `Hybrid` has less stay duration than `Reputation`, `Random`, `Best-fit`, and `Preemption`. Thus, `Hybrid` provides better relative quality to all peers than other four systems. The minimum, maximum, standard deviation, and average of duration for these six algorithms are listed in Table 4.

Table 4. Comparison of Duration (in seconds)

	Min	Max	Standard deviation	Average
Best-fit	500	3600	1460.7	1643.7
Random	500	3600	1260.4	2911.5
Preemption	500	3600	1335.3	2776.0
Reputation	500	3600	1339.4	1515.3
Optimal	500	809	57.4	572.2
Hybrid	500	852	73.5	614.8

Overhead. Though `Hybrid` and `Optimal` schemes have shown their efficiency in utilizing the system resources and minimizing service-time for peers, it comes with some cost of management, which we measure as overhead. In this section, we aims to evaluate the control and computation overheads due to frequent bandwidth re-allocation in terms of bytes and milliseconds.

In our simulation, the control traffic of `Hybrid` only accounts for 0.004% of the streaming traffic. The control overhead at time 1000 second, 2000 second, and 3000 second for these six algorithms are listed in Table 5. We also compare the control overhead in `Optimal` and `Hybrid` with that of other four systems. We list the total control overhead in all systems in Table 5. In comparison of the total control overhead in `Reputation`, `Best-fit`, `Random`, and `Preemption`, the total control overhead in `Reputation` is the largest; while the total control overhead in `Optimal` algorithm is significant larger than the rest four systems. `Hybrid` has much less control overhead compared with the `Optimal` scheme. The result reflects the tradeoff between the performance and the control overhead. We can see that the control overhead of `Hybrid` is reduced to a great extent compared with the optimal scheme, but `Hybrid` has near-optimal resource utilization compared with `Optimal`. By choosing `Hybrid` instead of `Optimal`, the scalability of the system is improved significantly. Furthermore, considering the ratio of the control overhead to the total streaming traffic, that overhead of `Hybrid` is still trivial.

Table 5. Comparison of Overhead (in bytes)

	At time 1000 (%)	At time 2000 (%)	At time 3000 (%)
Best-fit	882234 (16.0)	2296080 (16.3)	4053456 (17.9)
Random	47934 (0.9)	108234 (0.8)	247786 (0.8)
Preemption	69642 (1.3)	150282 (1.1)	172962 (1.1)
Reputation	4709736 (85.3)	10837044 (77.1)	19388718 (85.5)
Optimal	5521176 (100)	14050062 (100)	22669380 (100)
Hybrid	1447326 (26.2)	3562074 (25.4)	5721030 (25.2)

The time overhead in terms of milliseconds is shown in Table 6. Though `Hybrid` has higher overhead than `Best-fit`, `Random`, and `Preemption`, it is better than `Reputation` and its overhead is no more than 4% of that of `Optimal`.

Table 6. Comparison of Overhead (in milliseconds)

	Running time per peer (%)
Best-fit	138113 (3.0)
Random	127244 (2.8)
Preemption	127990 (2.8)
Reputation	211136 (4.6)
Optimal	4563609 (100)
Hybrid	179705 (3.9)

Our experimental results indisputably show that our hybrid resource allocation scheme can always achieve near-optimal performance in terms of system throughput, peer perceived qualities and resource utilization compared with optimal resource allocation.

6. Conclusion

While many P2P streaming systems have attracted a significant amount of Internet users and researchers, the resource allocation and utilization in existing P2P systems have not been well studied. Potentially, these systems enjoy the super scalability with the cost of poor resource utilization. In this paper, we lay out a theoretical framework that aims to maximize the system resource utilization in a heterogeneous P2P system to improve the system throughput while the least quality perceived by peers in the system is also maximized simultaneously. By comparing with the existing coarse admission control strategies, we propose a hybrid resource allocation approach and show that our proposed hybrid design can achieve near-optimal performance.

References

[1] Comscore. http://www.comscore.com/press/release.asp?press=2324.

[2] Mosek5.0. http://www.mosek.com.

[3] PPlive. http://www.pplive.com.

[4] PPStream. http://www.ppstream.com.

[5] Sopcast - free p2p Internet tv. http://www.sopcast.org.

[6] TVants. http://www.tvants-ppstream.com.

[7] R. Bar-Yehuda, K. Bendel, A. Freund, and D. Rawitz. Local ratio: a unified framework for approximation algorithms. *ACM Computing Surveys*, 36(4):422–463, December 2004.

[8] A. Bharambe, S. Rao, V. Padmanabhan, S. Seshan, and H. Zhang. The impact of heterogeneous bandwidth constraints on dht-based multicast protocols. In *Proceedings of IPTPS*, pages 121–127, February 2005.

[9] Y. Chu, S. Rao, and H. Zhang. A case for end system multicast. In *Proceedings of ACM SIGMETRICS*, pages 1–12, June 2000.

[10] Y. Chu, SG. Rao, S. Seshan, and H. Zhang. Enabling conferencing applications on the Internet using an overlay multicast architecture. In *Proceedings of ACM SIGCOMM*, pages 55–67, August 2001.

[11] C. Courcoubetis and R. Weber. Incentive for large peer-to-peer systems. In *IEEE Journal on Selected Areas in Communications*, volume 24, pages 1034–1050, September 2006.

[12] C. Huang, J. Li, and K. Ross. Can Internet video-on-demand be profitable? In *Proceedings of ACM SIGCOMM*, pages 313–324, 2007.

[13] Y. Huang, T. Fu, D. Chiu, J. Lui, and C. Huang. Challenges, design and analysis of a large-scale P2P-VoD system. In *Proceedings of ACM SIGCOMM*, pages 375–388, August 2008.

[14] M. Luby and N. Nisan. A parallel approximation algorithm for positive linear programming. In *Proceedings of ACM STOC*, pages 448–457, May 1993.

[15] T.B. Ma, Sam C.M. Lee, John C.S. Lui, and David K.Y. Yau. A game theoretic approach to provide incentive and service differentiation in P2P networks. In *Proceedings of ACM SIGMETRICS/PERFORMANCE*, pages 189–198, June 2004.

[16] N. Magharei and R. Rejaie. Prime: Peer-to-peer receiver-driven mesh-based streaming. In *Proceedings of IEEE INFOCOM*, pages 1415–1423, May 2007.

[17] W. Ooi. Dagster: Contributor-aware end-host multicast for media streaming in heterogeneous environment. In *Proceedings of ACM/SPIE MMCN*, pages 77–90, January 2005.

[18] D. Tran, K. Hua, and T. Do. Zigzag: An efficient peer-to-peer scheme for media streaming. In *Proceedings of IEEE INFOCOM*, pages 1283–1292, March 2003.

[19] C. Wu, B. Li, and S. Zhao. Multi-channel live P2P streaming: Refocusing on servers. In *Proceedings of IEEE INFOCOM*, pages 1355–1363, 2008.

[20] Y. Yan, A. El-Atawy, and E. Al-Shaer. Ranking-based optimal resource allocation in peer-to-peer networks. In *Proceedings of IEEE INFOCOM*, pages 1100–1108, May 2007.

[21] G. Yang, K. Suh, J. Kurose, and D. Towsley. P2Cast: peer-to-peer patching scheme for VoD service. In *Proceedings of ACM WWW*, pages 301–309, May 2003.

[22] X. Zhang, J. Liu, B. Li, and T. Yum. Coolstreaming/donet: A data-driven overlay network for efficient live media streaming. In *Proceedings of IEEE INFOCOM*, pages 2102–2111, March 2005.

[23] S. Zhong, J. Chen, and Y. Yang. Sprite: A simple, cheat-proof, credit-based system for mobile ad-hoc networks. In *Proceedings of IEEE INFOCOM*, pages 1987–1997, March 2003.