Maximizing the Bandwidth Multiplier Effect for Hybrid Cloud-P2P Content Distribution

Zhenhua Li [1,2], Tieying Zhang [3], Yan Huang [4], Zhi-Li Zhang [2], Yafei Dai [1]

[1] *Peking University*, *Beijing, China*, lizhenhua1983@gmail.com
[2] *University of Minnesota*, *Minneapolis, MN, US*, zhzhang@cs.umn.edu
[3] *ICT, CAS*, *Beijing, China*, zhangtiey@gmail.com
[4] *Tencent Research*, *Shanghai, China*, galehuang@tencent.com

***Abstract*—*Hybrid cloud-P2P* content distribution ("CloudP2P") provides a promising alternative to the conventional cloud-based or peer-to-peer (P2P)-based large-scale content distribution. It addresses the potential limitations of these two conventional approaches while inheriting their advantages. A key strength of CloudP2P lies in the so-called *bandwidth multiplier effect*: by appropriately allocating a small portion of cloud (server) bandwidth S_i to a *peer swarm* i (consisting of users interested in the same content) to seed the content, the users in the peer swarm – with an aggregate download bandwidth D_i – can then distribute the content among themselves; we refer to the ratio D_i/S_i as the bandwidth multiplier (for peer swarm i). A major problem in the design of a CloudP2P content distribution system is therefore how to allocate cloud (server) bandwidth to peer swarms so as to maximize the overall bandwidth multiplier effect of the system. In this paper, using real-world measurements, we identify the key factors that affect the bandwidth multipliers of peer swarms and thus construct a *fine-grained* performance model for addressing the *optimal bandwidth allocation problem* (OBAP). Then we develop a *fast-convergent* iterative algorithm to solve OBAP. Both trace-driven simulations and prototype implementation confirm the efficacy of our solution.**

I. Introduction

Large-scale content distribution has become increasingly prevalent and contributes to a significant portion of the Internet traffic. Today's large content providers (e.g., YouTube and Netflix) typically employ a *cloud-based* approach which relies on huge data centers for computing and storage and utilizes geographically dispersed CDNs (content distribution networks) to further meet users' demand on content delivery performance. Such an approach requires a massive and costly computing, storage and delivery infrastructure. For instance, YouTube, as a subsidiary of Google, utilizes Google's own massive delivery infrastructure [1], whereas Netflix employs Amazon's cloud services and third-party CDNs such as Akamai and Limelight [2]. In contrast, the *peer-to-peer* (P2P)-based content distribution [3], [4] incurs little infrastructure cost and can scale with the user scale, as it utilizes individual users' machines (and their ISPs) for replicating and delivering content to each other. However, P2P also suffers several technical shortcomings such as end users' high dynamics and heterogeneity, difficulty to find "seeds" or other peers with the content in which an end user is interested. As a result, the working efficacy of P2P can be quite poor and unpredictable.

978-1-4673-1298-1/12/$31.00

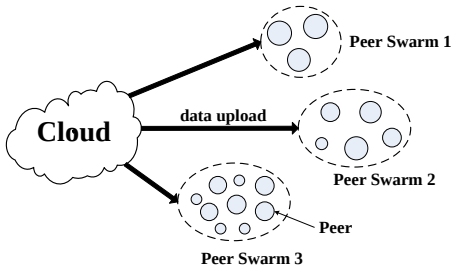
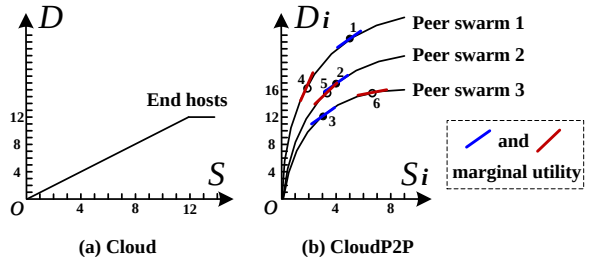


Fig. 1. Hybrid cloud-P2P content distribution. Inside each swarm, peers exchange data with others; meanwhile, they get data from the cloud. Different peer swarms share different contents.

A third approach – *hybrid cloud-P2P* ("CloudP2P") content distribution – has recently emerged [5], [6], [7], [8], [9], [10], [11], [12] as a promising alternative. It addresses the potential limitations of these two conventional approaches while inheriting their advantages. As depicted in Fig. 1, CloudP2P comprises of a cloud component as well as a number of peer swarms. The cloud not only provides content "seeds" but also assists end users to find other peers who are interested in the same content. A peer swarm starts by obtaining a content seed from the cloud, and subsequently, peers within the swarm can exchange data among themselves. Compared to the purely cloud-based approach, CloudP2P incurs far lower infrastructure and network bandwidth costs (especially there is no need for a large-scale CDN infrastructure). Take Youku [13], the biggest video sharing site in China, as an example. Since its startup, the cloud bandwidth expense of Youku has consumed more than half of its total income. In the past two years, Youku has been encouraging its users to install the iKu accelerator [5], which changes its purely cloud-based content distribution to a hybrid CloudP2P architecture. Meanwhile, CloudP2P also effectively circumvents the key limitations of P2P by providing extra cloud bandwidth to those peer swarms who do not work well for lack of seed peers.

A key strength of CloudP2P lies in the so-called *bandwidth multiplier effect*: by appropriately allocating a small portion of cloud (server) bandwidth S_i to a *peer swarm* i (consisting of users interested in the same content) to seed the content, CloudP2P can attain a higher aggregate content distribution bandwidth (D_i) by letting peers to exchange data and distribute content among themselves. Borrowing a term from economics,



(a) Cloud (b) CloudP2P

Fig. 2. The bandwidth multiplier ($= \frac{D}{S}$) of (a) Cloud and (b) CloudP2P: a simple example. S denotes the invested cloud bandwidth and D denotes the end hosts' aggregate download bandwidth. As to CloudP2P, $S = \sum_i S_i$ and $D = \sum_i D_i$ where i denotes the i-th peer swarm. For each peer swarm, its *marginal utility* at a certain point is the partial derivative: $\frac{\partial D_i}{\partial S_i}$. Here we use $\frac{\partial D_i}{\partial S_i}$ rather than $\frac{dD_i}{dS_i}$ because D_i is not fully dependent on S_i.

we refer to the ratio D_i/S_i as the *bandwidth multiplier* (for peer swarm i). A major problem in the design of a CloudP2P content distribution system is therefore how to allocate cloud (server) bandwidth to peer swarms so as to maximize the overall bandwidth multiplier effect of the system – the *optimal bandwidth allocation problem* (OBAP).

Below we use a simple (artificial) example to illustrate the bandwidth multiplier effect of CloudP2P and argue why in solving OBAP, one must consider the *marginal utility* of cloud bandwidth allocation. As plotted in Fig. 2(a), in the conventional cloud-based content distribution, the bandwidth multiplier is usually 1.0 because each user typically downloads content directly form the cloud content provider. In the case of CloudP2P, the bandwidth multiplier can be much larger than 1.0, since the data exchange among end host peers can "multiply" the upload bandwidth of cloud servers. The *achievable* bandwidth multiplier will hinge critically on the bandwidth allocation scheme as well as individual peer swarms. Depending on the specifics of each peer swarm (e.g., its size, the bandwidth available among the peers, etc.), given the same allocated cloud bandwidth, the bandwidth multiplier can vary significantly from one peer swarm to another peer swarm. Fig. 2(b) plots three hypothetical bandwidth multiplier curves for three different peer swarms. Suppose the total invested cloud bandwidth is $S = 12$ and the bandwidth allocation scheme corresponds to points $\{1, 2, 3\}$ in Fig. 2(b), and then the overall bandwidth multiplier for all three peer swarms is $\frac{D}{S} = \frac{\sum_i D_i}{\sum_i S_i} = \frac{23+17+12}{3+4+5} = 4.33$.

Because of the nonlinear nature of the bandwidth multiplier curves, it is clear that we must take into account the *margianl utility* ($\frac{\partial D_i}{\partial S_i}$) of cloud bandwidth allocation in solving the optimal bandwidth allocation problem. The commonly used bandwidth allocation algorithms in commercial systems, however, do not (well) consider the marginal utility of cloud bandwidth allocation, thus leading to suboptimal or even poor bandwidth multiplier effect. For instance, the *free-competitive* strategy simply lets all the end hosts/peer swarms to compete freely for the cloud bandwidth, whereas the *proportional-allocate* scheme allocates cloud bandwidth proportionally to peer swarms according to their size. As a result, an "over-feeding" peer swarm may be allocated with too much cloud bandwidth resulting in low marginal utility, while a "starving" peer swarm may get little cloud bandwidth resulting in high marginal utility. For example, see the other allocation scheme corresponding to points $\{4, 5, 6\}$ in Fig. 2(b) where the cloud bandwidth ($S = 12$) is proportionally allocated into each peer swarm. The overall bandwidth multiplier corresponding to $\{4, 5, 6\}$ is $\frac{16+15+15}{2+3.3+6.7} = 3.83 < 4.33$ (the overall bandwidth multiplier of $\{1, 2, 3\}$). Intuitively, we find larger bandwidth multiplier implies more balanced marginal utilities among peer swarms (which will be formally proved later).

In this paper, using real-world measurements, we identify the key factors that affect the bandwidth multipliers of peer swarms and thus construct a *fine-grained* performance model for addressing the optimal bandwidth allocation problem (OBAP). This model takes into account the impact of both outer-swarm and intra-swarm bandwidth provisions on a peer swarm and can well match the measurement data. We further prove that the bandwidth multiplier effect is closely related to the marginal utility of cloud bandwidth allocation. To solve OBAP, we develop a *fast-convergent* iterative algorithm by finding the optimal direction and adaptively setting the stepsize in each iteration step. Compared with the commonly used iterative algorithms, our proposed iterative algorithm has provable convergence, faster convergence speed and ease of use (as to a large-scale highly dynamic CloudP2P system). Our whole solution is named as "FIFA" which denotes the combination of the **fi**ne-grained performance model ("FI") and the **fa**st-convergent iterative algorithm ("FA").

We use both large-scale trace-driven simulations and small-scale prototype implementation to evaluate the bandwidth allocation performance of FIFA. Simulation results based on the log trace of around one million peers reveal that the overall bandwidth multiplier of FIFA is 20%, 17% and 8% larger than that of the existing bandwidth allocation algorithms: *free-competitive*, *proportional-allocate* and *Ration* [8], respectively. Meanwhile, the total control overhead bandwidth of FIFA stays below 15 KBps – even less than a common user's download bandwidth. Small-scale prototype implementation also confirms the efficacy of FIFA.

The remainder of this paper is organized as follows. Section II reviews the related work. Through real-world measurements, we construct a fine-grained performance model to address OBAP in Section III. Then we propose a fast-convergent iterative algorithm to solve OBAP in Section IV. After that, the performance of our solution (FIFA) is evaluated in Section V and VI. Finally, we conclude our paper in Section VII.

II. Related Work

The commonly used bandwidth allocation algorithms of CloudP2P generally adopt a coarse-grained performance model by making some ideal or simplified assumptions. For example, most commercial systems (e.g., [7]) use the *free-competitive* algorithm (or make some minor changes). Such an algorithm simply allocates a certain amount of cloud bandwidth for all peer swarms to compete freely. Obviously, *free-*

competitive benefits those aggressive or selfish peer swarms who might set up numerous TCP/UDP connections to grasp as much cloud bandwidth as possible. To alleviate the drawback of *free-competitive*, some systems (e.g., [14]) employ the *proportional-allocate* algorithm which proportionally allocates cloud bandwidth to peer swarms based only on their swarm scale. *Proportional-allocate* implies the assumption that the demand of cloud bandwidth is only dependent on the number of peers inside a swarm, which is deviant from reality.

A similar work to FIFA is AntFarm [15], which uses a centralized coordinator to dynamically split seed peers' bandwidth among peer swarms (AntFarm does not consider the outer-swarm cloud bandwidth provision). The practical application of AntFarm may be difficult for two reasons: 1) Accurately splitting a seed peer's bandwidth and allocating it into multiple swarms is quite difficult due to the dynamic nature of end host peers, so it is rarely supported by commercial P2P systems; 2) The coordinator employs a centralized token protocol which strictly controls the behavior of each peer (e.g., neighbor selection and data exchange), which is not quite compatible with the distributed working principle of P2P.

Another similar work is Ration [8], a cloud bandwidth allocation algorithm for P2P live TV streaming. Ration constructs its CloudP2P performance model by using the impact factors S_i and l_i, where S_i denotes the cloud bandwidth allocated to peer swarm i and l_i denotes the number of online leechers inside peer swarm i. Since Ration works in a live TV streaming environment where most viewers devote their bandwidth to downloading and would leave their peer swarm as soon as they finish viewing (that is to say, $s_i \ll l_i$, where s_i is the number of online seed peers inside peer swarm i.), Ration does not (need to) consider the impact of seed peers – the intra-swarm bandwidth provision. In our work, the impact of seed peers is also taken as a key factor and we find it is *nontrivial* for modeling a large-scale CloudP2P file-sharing system. Given that a P2P file-sharing system usually accommodates much more dynamics and heterogeneity than a P2P live streaming system, our proposed performance model should be more fine-grained and general.

To solve the optimal bandwidth allocation problem, AntFarm employs the "hill-climbing" iterative algorithm ("HC") while Ration employs the "water-filling" iterative algorithm ("WF") [16]. Both HC and WF are commonly used iterative algorithms to solve optimization problems; however, we find their convergence speeds might be quite slow on handling a large number of highly dynamic peer swarms, mainly because they *have to* use a very short stepsize to make their iteration progress converge. We note that Ration had realized the problem of WF and thus proposed an incremental version of WF to increase its convergence speed, but the incremental version only alleviates rather than resolves its problem. By finding the optimal direction and adaptively setting the stepsize in each iteration step, our proposed fast-convergent iterative algorithm (FA) has provable convergence, faster convergence speed and ease of use.

III. Fine-grained Performance Model

A. Key Impact Factors

A number of factors may affect the bandwidth multiplier of a peer swarm, such as the *allocated cloud bandwidth*, *number of leechers*, *number of seeders*[1], *available bandwidth of each peer*, *connection topology among peers* and *distribution of unique data blocks*. Obviously, it is impossible to take all these factors into account, and considering too many detailed/trivial factors will bring the bandwidth allocation algorithm unbearable communication/computation overhead. Instead, our methodology is to find out the key impact factors that influence the bandwidth multiplier of a peer swarm.

To find the key impact factors, we utilize the real-world measurements from QQXuanfeng [11], a large-scale CloudP2P file-sharing system [17]. We track 1457 peer swarms in one day (involving around one million peers), record their respective working parameters: D_i, S_i, s_i and l_i per five minutes, and then analyze the relationships between these parameters. The meanings of these parameters are listed as follows:

- D_i: the aggregate download bandwidth of the peers inside (peer) swarm i.
- S_i: the cloud bandwidth allocated to swarm i. S_i denotes the outer-swarm bandwidth provision to swarm i.
- s_i: the number of online seeders in swarm i. s_i denotes the intra-swarm bandwidth provision to swarm i.
- l_i: the number of online leechers in swarm i. l_i denotes how many peers are the bandwidth consumers that benefit from S_i and s_i. (Note that a leecher also uploads data to others.)

As depicted in Fig. 3(a), we discover *approximate* exponential relationship between $\frac{D_i}{l_i}$ and $\frac{S_i}{l_i}$, which is further confirmed by the corresponding *log-log* curve in Fig. 4(a). Thus, as to a typical peer swarm i:

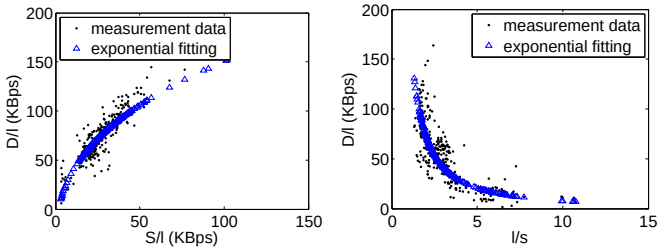
$$\frac{D_i}{l_i} \propto (\frac{S_i}{l_i})^{\alpha_i}, \quad 0 < \alpha_i < 1;$$

on the other hand, from Fig. 3(b) and Fig. 4(b) we find that the exponential relationship also *approximately* holds between $\frac{D_i}{l_i}$ and $\frac{l_i}{s_i}$, except that the exponent is negative:

$$\frac{D_i}{l_i} \propto (\frac{l_i}{s_i})^{-\beta_i}, \quad \beta_i > 0.$$

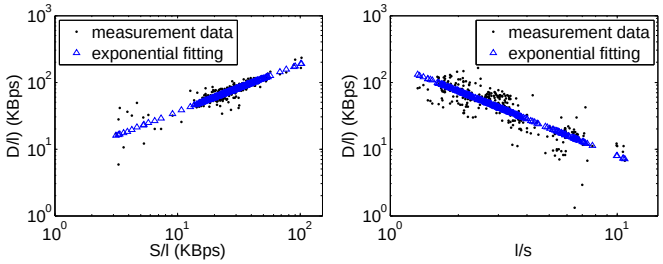
The abovementioned exponential relationships basically comply with our intuitive experiences obtained from Fig. 2 – when a peer swarm is allocated with too much cloud bandwidth or contains too many seeders, the marginal utility for increasing its aggregate download bandwidth will become trivial. Now that the above exponential relationships are merely *approximate*, we still have a problem whether we should use $\frac{S_i}{l_i}$, $\frac{l_i}{s_i}$, both or even more parameters to well model $\frac{D_i}{l_i}$. To this end, we use $\frac{S_i}{l_i}$, $\frac{l_i}{s_i}$ and both, respectively, to model $\frac{D_i}{l_i}$, where the corresponding constant parameters (α_i, f_i), (β_i, f_i) and (α_i, β_i, f_i) are computed based on the one-day measurements of peer swarm i. From Fig. 5 and Table I we confirm that the

[1] "seeder" has the same meaning with "seed peer".



(a) $\frac{S_i}{l_i}$ vs $\frac{D_i}{l_i}$. (b) $\frac{l_i}{s_i}$ vs $\frac{D_i}{l_i}$.

Fig. 3. Relationships between (a) $\frac{D_i}{l_i}$ and $\frac{S_i}{l_i}$, (b) $\frac{D_i}{l_i}$ and $\frac{l_i}{s_i}$, as to a typical peer swarm i.



(a) $log(\frac{S_i}{l_i})$ vs $log(\frac{D_i}{l_i})$. (b) $log(\frac{l_i}{s_i})$ vs $log(\frac{D_i}{l_i})$.

Fig. 4. Relationships between (a) $log(\frac{D_i}{l_i})$ and $log(\frac{S_i}{l_i})$, (b) $log(\frac{D_i}{l_i})$ and $log(\frac{l_i}{s_i})$, as to a typical peer swarm i.

key impact factors should include both $\frac{S_i}{l_i}$ and $\frac{l_i}{s_i}$. Therefore, we get the following equation:

$$\frac{D_i}{l_i} = (\frac{S_i}{l_i})^{\alpha_i} \cdot (\frac{l_i}{s_i})^{-\beta_i} \cdot f_i,^2 \quad (1)$$

where $0 < \alpha_i < 1$, $\beta_i > 0$ and $f_i > 0$. Then the aggregate download bandwidth of peer swarm i is:

$$D_i = S_i^{\alpha_i} \cdot l_i^{1-\alpha_i-\beta_i} \cdot {s_i}^{\beta_i} \cdot f_i. \quad (2)$$

Since S_i is the only decision variable that we can schedule, we also write D_i as $D_i(S_i)$. Finally, the bandwidth multiplier of peer swarm i is:

$$\frac{D_i}{S_i} = S_i^{\alpha_i-1} \cdot l_i^{1-\alpha_i-\beta_i} \cdot {s_i}^{\beta_i} \cdot f_i. \quad (3)$$

To compute the constant parameters α_i, β_i and f_i, we first transform Equation (1) into its "log" form:

$$\log \frac{D_i}{l_i} = \log \frac{S_i}{l_i} \cdot \alpha_i - \log \frac{l_i}{s_i} \cdot \beta_i + \log f_i, \quad (4)$$

so that α_i, β_i and f_i can be computed by using the measurements of D_i, l_i, S_i and s_i, via the classical linear regression method. One thing to note is that the abovementioned constant parameters (α_i, β_i and f_i) can only be taken as "constant" during a certain period (typically one day or several hours), so they need to be periodically updated using the latest measurements.

[2]If $s_i = 0$, we just let $\frac{l_i}{s_i} = 1$ so that $(\frac{l_i}{s_i})^{-\beta_i}$ is ignored.

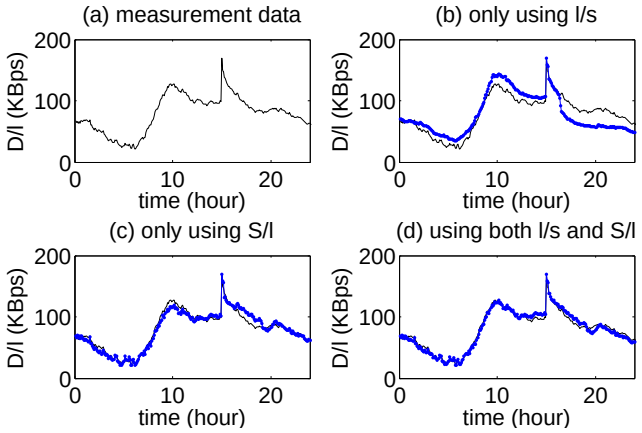


Fig. 5. Modeling $\frac{D_i}{l_i}$ using (b) $\frac{l_i}{s_i}$ (i.e., $\frac{D_i}{l_i} = (\frac{l_i}{s_i})^{-\beta_i} \cdot f_i$), (c) $\frac{S_i}{l_i}$ (i.e., $\frac{D_i}{l_i} = (\frac{S_i}{l_i})^{\alpha_i} \cdot f_i$) and (d) both (i.e., $\frac{D_i}{l_i} = (\frac{S_i}{l_i})^{\alpha_i} \cdot (\frac{l_i}{s_i})^{-\beta_i} \cdot f_i$), as to a typical peer swarm. Clearly, the key impact factors should include both $\frac{S_i}{l_i}$ and $\frac{l_i}{s_i}$ so that the model can match the (a) measurement data well.

TABLE I
RELATIVE ERRORS OF THE THREE MODELS APPLIED TO ALL THE 1457 PEER SWARMS, COMPARED WITH THEIR MEASUREMENTS DATA.

Model	**Avg** (relative error)	**Min**	**Max**
(b) only using $\frac{l_i}{s_i}$	0.1391	0.006	0.9036
(c) only using $\frac{S_i}{l_i}$	0.0738	0	0.2366
(d) using both $\frac{S_i}{l_i}$ and $\frac{l_i}{s_i}$	0.0308	0	0.0972

B. OBAP and Its Optimal Solution

Till now, the optimal bandwidth allocation problem (OBAP) of CloudP2P can be formalized as follows:

OBAP

Maximize the overall bandwidth multiplier $(\frac{D}{S})$

subject to the following conditions:

$$\begin{cases} D = \sum_{i=1}^{m} D_i, \text{ where } m \text{ is the number of swarms;} \\ S = \sum_{i=1}^{m} S_i, \text{ where } S \text{ is taken as a constant during an allocation period;} \\ S_i \geq 0, \quad \forall i \in \{1, 2, \cdots, m\}; \\ D_i = S_i^{\alpha_i} \cdot l_i^{1-\alpha_i-\beta_i} \cdot {s_i}^{\beta_i} \cdot f_i, \quad \forall i \in \{1, 2, \cdots, m\}; \end{cases}$$

with decision variables $S_1, S_2, \cdots, S_m$.

We can see that OBAP is a constrained nonlinear optimization problem [18]. Given that S is taken as a constant during an allocation period, maximizing $\frac{D}{S}$ is equal to maximizing D. When the optimal solution of OBAP exists, suppose the optimal solution is $\mathbf{S}^* = (S_1^*, S_2^*, \cdots, S_m^*)$ [3] and the corresponding aggregate download bandwidth of each swarm is $(D_1^*, D_2^*, \cdots, D_m^*)$. Thus, according to the optimality

[3]The **bold** font is used to represent a vector. It is possible that OBAP has no optimal solution within its constrained set.

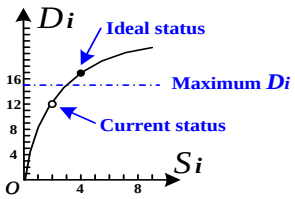


Fig. 6. An exceptional case in which the peer swarm cannot be adjusted to its ideal status.

condition of constrained nonlinear optimization [18], we have:

$$\sum_{i=1}^{m} \frac{\partial D_i(S_i^*)}{\partial S_i}(S_i - S_i^*) \leq 0, \quad \forall S_i \geq 0 \text{ with} \sum_{i=1}^{m} S_i = S. \quad (5)$$

Then fixing an arbitrary i and letting j be any other index, we construct a feasible solution $\mathbf{S}'$ to the constraints as:

$$S_i' = 0, S_j' = S_i^* + S_j^*, S_k' = S_k^*, \quad \forall k \neq i, j.$$

Applying $\mathbf{S}'$ to Equation (5), we get:

$$(\frac{\partial D_j(S_j^*)}{\partial S_j} - \frac{\partial D_i(S_i^*)}{\partial S_i}) \cdot S_i^* \leq 0, \quad \forall i, j \quad (i \neq j).$$

If $S_i^* = 0$, peer swarm i gets no cloud bandwidth and thus we do not need to consider such a swarm for cloud bandwidth allocation. Consequently, we have $\forall i \in \{1, 2, \cdots, m\}$, $S_i^* > 0$, and then

$$\frac{\partial D_j(S_j^*)}{\partial S_j} \leq \frac{\partial D_i(S_i^*)}{\partial S_i}, \quad \forall i, j \quad (i \neq j). \quad (6)$$

Therefore, the optimal solution $\mathbf{S}^*$ has the following form:

$$\frac{\partial D_j(S_1^*)}{\partial S_1} = \frac{\partial D_2(S_2^*)}{\partial S_2} = \cdots = \frac{\partial D_m(S_m^*)}{\partial S_m}, \quad (7)$$

which means the *marginal utility* of the cloud bandwidth allocated to each peer swarm should be equal in the optimal solution (if it exists). In practice, there is an exceptional case in which a peer swarm i cannot be adjusted to its "ideal" status (i.e., the marginal utility of the cloud bandwidth allocated to peer swarm i is equal to that of the other swarms), and this exceptional case will cause OBAP to have no optimal solution in the form of Equation (7). As illustrated in Fig. 6, for some reasons peer swarm i has an upper bound of its aggregate download bandwidth ("Maximum D_i"), which prevents the bandwidth allocation algorithm from adjusting peer swarm i to its ideal status. In this situation, we just allocate the least cloud bandwidth to improve its aggregate download bandwidth to "Maximum D_i" so that the relative deviation of marginal utility among all the peer swarms can be as little as possible. In conclusion, we have the following theorem:

Theorem 1. *For CloudP2P content distribution, the maximum bandwidth multiplier implies that the marginal utility of the cloud bandwidth allocated to each peer swarm should be equal. In practice, we want the relative deviation of marginal utility among all the peer swarms to be as little as possible, i.e., larger bandwidth multiplier implies more balanced marginal utilities among peer swarms.*

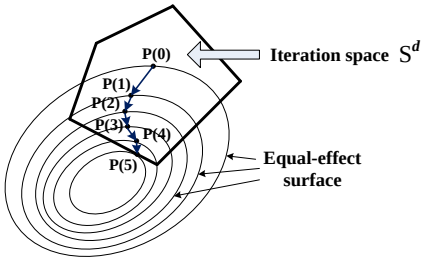


Fig. 7. A demo iteration process. The *equal-effect surface* is the set of all the points $\mathbf{P}$ that have the same performance value $f(\mathbf{P})$.

IV. Fast-convergent Iterative Algorithm

In last section we have formulated the optimal bandwidth allocation problem (OBAP) into a constrained nonlinear optimization problem. The optimal solution of such a problem is typically obtained via iterative operations in multiple steps until the algorithm converges [18]. Therefore, the convergence property of the iterative algorithm is critical in solving OBAP.

The convergence property of an iterative algorithm mainly depends on two aspects: *iteration direction* and *iteration stepsize*. For a d-dimension[4] constrained nonlinear optimization problem, all its feasible solutions compose a d-dimension *iteration space* $\mathbb{S}^d$. Suppose the iterative algorithm starts at an arbitrary point $\mathbf{P^{(0)}} = (P_1^{(0)}, P_2^{(0)}, \cdots, P_d^{(0)}) \in \mathbb{S}^d$. Then in each subsequent iteration step, the algorithm must determine an *iteration direction* and an *iteration stepsize* to go further to a new point $\mathbf{P^{(k)}} = (P_1^{(k)}, P_2^{(k)}, \cdots, P_d^{(k)}) \in \mathbb{S}^d$ so that $\mathbf{P^{(k)}}$ is closer to the optimal point $\mathbf{P}^*$ than $\mathbf{P^{(k-1)}}$, as shown in Fig. 7. Specifically, the iteration process can be formalized as:

$$\begin{aligned} &\mathbf{P}^{(k+1)} = \mathbf{P}^{(k)} + t^{(k)}(\overline{\mathbf{P}}^{(k)} - \mathbf{P}^{(k)}), \\ &\text{until} \quad |f(\mathbf{P}^{(k+1)}) - f(\mathbf{P}^{(k)})| < \epsilon. \end{aligned} \quad (8)$$

where $f(.)$ is the performance function, ϵ is a very small constant, $(\overline{\mathbf{P}}^{(k)} - \mathbf{P}^{(k)})$ is the iteration direction, and $t^{(k)}$ is the iteration stepsize in the k-th step. The task of our fast-convergent iterative algorithm (FA) is to determine appropriate $\overline{\mathbf{P}}^{(k)}$ and $t^{(k)}$ in the k-th step so that the iteration process can be as fast as possible.

Iteration direction. For a nonlinear optimization problem, usually it is impossible to directly find the ultimate direction $\mathbf{P}^* - \mathbf{P}^{(0)}$ (or $\mathbf{P}^* - \mathbf{P}^{(k)}$ for a certain k) because this is basically as difficult as to directly find $\mathbf{P}^*$. Instead, FA utilizes the *conditional gradient method* [18] to determine the iteration direction in each step. For a function $f(\mathbf{P})$, it is well known that $f(\mathbf{P}^{(k+1)})$ can be approximated via the Taylor expansion:

$$\begin{aligned} &f(\mathbf{P}^{(k+1)}) = f(\mathbf{P}^{(k)}) + \nabla f(\mathbf{P}^{(k)})(\mathbf{P}^{(k+1)} - \mathbf{P}^{(k)})^T + \\ &\frac{1}{2}(\mathbf{P}^{(k+1)} - \mathbf{P}^{(k)})\nabla^2 f(\mathbf{P}^{(k)})(\mathbf{P}^{(k+1)} - \mathbf{P}^{(k)})^T + \cdots. \end{aligned} \quad (9)$$

[4] d-dimension means the optimization problem deals with d decision variables in total. As to OBAP, d is the total number of peer swarms.

where $\nabla f(\mathbf{X}) = (\frac{\partial f(\mathbf{X})}{\partial X_1}, \frac{\partial f(\mathbf{X})}{\partial X_2}, \cdots, \frac{\partial f(\mathbf{X})}{\partial X_d})$. The conditional gradient method uses the first-order Taylor expansion to approximate $f(\mathbf{P}^{(k+1)})$:

$$f(\mathbf{P}^{(k+1)}) \approx f(\mathbf{P}^{(k)}) + \nabla f(\mathbf{P}^{(k)})(\mathbf{P}^{(k+1)} - \mathbf{P}^{(k)})^T. \quad (10)$$

As to the OBAP problem, the dimension (d) is just the number of peer swarms (m), so that $\mathbf{P}^{(k)} = \mathbf{S}^{(k)}$, $f(\mathbf{P}^{(k)}) = f(\mathbf{S}^{(k)}) = \frac{D^{(k)}}{S} = \frac{\sum_{i=1}^{m} D_i(S_i)}{S}$ and $D_i(S_i) = S_i^{\alpha_i} \cdot l_i^{1-\alpha_i-\beta_i} \cdot {s_i}^{\beta_i} \cdot f_i$. Then we have:

$$f(\mathbf{S}^{(k+1)}) \approx f(\mathbf{S}^{(k)}) + \nabla f(\mathbf{S}^{(k)})(\mathbf{S}^{(k+1)} - \mathbf{S}^{(k)})^T. \quad (11)$$

Since our goal is to maximize $f(\mathbf{S})$ on condition that $\sum_{i=1}^{m} S_i = S$ and $S_i \geq 0, \forall i \in \{1, 2, \cdots, m\}$, we need to (greedily) maximize $f(\mathbf{S}^{(k+1)})$ in Equation (11) in the k-th iteration step. Thus, we must find the specific $\mathbf{S}$ that satisfies the following problem:

Maximize $\nabla f(\mathbf{S}^{(k)})(\mathbf{S} - \mathbf{S}^{(k)})^T$
subject to $\sum_{i=1}^{m} S_i = S$ and $S_i \geq 0, \forall i \in \{1, 2, \cdots, m\}$.

By expanding $\mathbf{S}$, $\mathbf{S}^{(k)}$ and $\nabla f(\mathbf{S}^{(k)})$, we transform the above problem into

Maximize $\sum_{i=1}^{m} \frac{\partial D_i(S_i^{(k)})}{\partial S_i}(S_i - S_i^{(k)})$
subject to $\sum_{i=1}^{m} S_i = S$ and $S_i \geq 0, \forall i \in \{1, 2, \cdots, m\}$.

It is not difficult to find that the above problem is a linear optimization problem and the optimal solution $\overline{\mathbf{S}}^{(k)}$ is:

$$\overline{S}_j^{(k)} = S, \text{ for the } j = \arg\max_{i \in \{1,2,\cdots,m\}} \frac{\partial D_i(S_i^{(k)})}{\partial S_i}; \quad \text{and} \quad \overline{S}_i^{(k)} = 0, \forall i \in \{1, 2, \cdots, j-1, j+1, \cdots, m\}. \quad (12)$$

So we get the optimal iteration direction in the k-th step:

$$\mathbf{d}^{(k)} = \overline{\mathbf{S}}^{(k)} - \mathbf{S}^{(k)}. \quad (13)$$

Iteration stepsize. Till now we have got that the k-th step of our FA iterative algorithm proceeds as:

$$\mathbf{S}^{(k+1)} = \mathbf{S}^{(k)} + t^{(k)}\mathbf{d}^{(k)}$$

where $\mathbf{d}^{(k)}$ is determined in Equation (12) and (13). Ideally, the stepsize $t^{(k)}$ should satisfy the following conditions:

Maximize $f(\mathbf{S}^{(k)} + t^{(k)}\mathbf{d}^{(k)})$
subject to $\mathbf{S}^{(k)} + t^{(k)}\mathbf{d}^{(k)}$ is a feasible solution.

Unfortunately, the above problem is still a nonlinear optimization problem and thus it is impossible to directly obtain its optimal solution. Instead, we utilize the *Armijo rule* [19] to adaptively set the iteration stepsize $t^{(k)}$, in order to guarantee that $f(\mathbf{S}^{(k+1)})$ is at least larger than $f(\mathbf{S}^{(k)})$ by a bound:

$$f(\mathbf{S}^{(k)} + \tau^j \mathbf{d}^{(k)}) - f(\mathbf{S}^{(k)}) \geq |\sigma\tau^j \nabla f(\mathbf{S}^{(k)}) {\mathbf{d}^{(k)}}^T| \quad (14)$$

where the two constant parameters $\tau, \sigma \in (0, 1)$, and j is tried successively as 0, 1, 2, ..., until the above inequality is satisfied for a certain j (which is $j^{(k)}$). As a result, we get the adaptive iteration stepsize in the k-th step for FIFA:

$$t^{(k)} = \tau^{j^{(k)}} \quad (15)$$

Summary of FA. The fast-convergent iterative algorithm (FA) efficiently solves OBAP by finding the optimal direction and adaptively setting the stepsize in each iteration step. First, the convergence of FA is provable due to its combinatory use of the *conditional gradient method* and the *Armijo rule* (refer to Proposition 2.2.1 in [18]). Second, FA is easy to use because all the related parameters, τ and σ, can be easily configured. For example, we simply configure $\tau = 0.5$ and $\sigma = 0.01$ for FA, and then it is well applicable to all the simulation/implementation scenarios in Section V and VI. Finally, although the accurate convergence speed of FA cannot be theoretically proved, FA exhibits nearly-linear convergence speed in our performance evaluation (refer to Section V-C). That is to say, for a CloudP2P system consisting of m peer swarms, FA converges in nearly $\Theta(m)$ steps.

Comparisons of WF, HC and FA. The "water-filling" algorithm (WF) is a classical iterative algorithm in solving constrained nonlinear optimization problems (e.g., [8]) for its simplicity and intuitive explanation. In each iterative step, WF only finds *two* components of $\mathbf{S}^{(\mathbf{k})}$, i.e., $S_h^{(k)}$ and $S_l^{(k)}$ satisfying the following conditions: $h = \arg\max_{i \in \{1,2,\cdots,m\}} \frac{\partial D_i(S_i^{(k)})}{\partial S_i}$ and $l = \arg\min_{i \in \{1,2,\cdots,m\}} \frac{\partial D_i(S_i^{(k)})}{\partial S_i}$. Then WF moves a constant portion δ from $S_l^{(k)}$ to $S_h^{(k)}$: $S_h^{(k)} \leftarrow S_h^{(k)} + \delta$ and $S_l^{(k)} \leftarrow S_l^{(k)} - \delta$. This movement looks like "filling some water from one cup to the other". In other words, the iteration direction and iteration stepsize of WF are set as follows: $\mathbf{d}^{(k)} = (d_1^{(k)}, d_2^{(k)}, \cdots, d_m^{(k)})$, where $d_h = 1, d_l = -1, d_i = 0, \forall i \neq h, l$; and $t^{(k)} = \delta$.

Obviously, WF uses a restricted iteration direction (only in two dimensions among the total m dimensions) and a fixed iteration stepsize (δ). The fundamental problem of WF lies in the setting of δ. If δ is set too big, WF will not converge; if δ is set too small, WF will converge slowly. Still worse, on handling a large number of highly dynamic peer swarms, setting an *appropriate* (i.e., neither too big nor too small) iteration stepsize (δ) for WF becomes extremely difficult. Consequently, the only practical choice is to set an extremely small δ resulting in a huge number of iteration steps and slow convergence speed. On the contrary, the iteration stepsize of FA is adaptively set so the number of iteration steps depends on the number of peer swarms. Fig. 8 is a comparison of the iterative operations of WF and FA when there are only two peer swarms: $S_1 = 0.15$ and $S_2 = 0.85$ (the total cloud bandwidth is normalized as $S = 1$). Additionally, the restricted iteration direction further slows down the iteration process of WF because WF always "walks" only in two dimensions among the total m dimensions. On the contrary, the iteration direction of FA can be in all dimensions. Fig. 9 illustrates a demo comparison when there are three peer swarms.

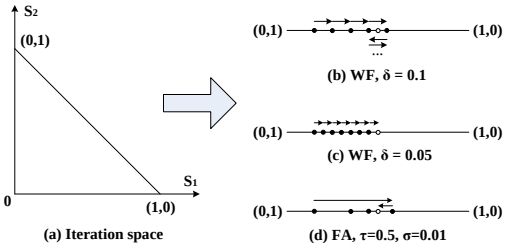


Fig. 8. A comparison of the iterative operations of WF and FA when there are only two peer swarms. (a) The iteration space is a line. (b) WF does not converge for the stepsize is too big. (c) WF converges in 7 steps for the stepsize is small enough. (d) FA converges in 2 steps.

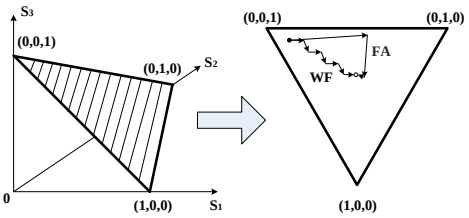


Fig. 9. A comparison of the iterative operations of WF and FA when there are three peer swarms. WF always "walks" only in two dimensions while FA can walk in all the m dimensions

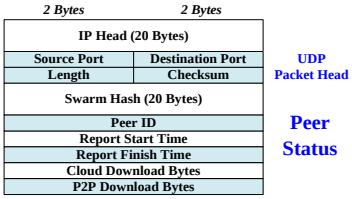


Fig. 10. Structure of the peer status report. A seeder's status report does not have the fields of "Cloud Download Bytes" and "P2P Download Bytes".

The "hill-climbing" algorithm (HC) always sets all the components of $\mathbf{S}^{(0)}$ to zero and stores the total cloud bandwidth S in a "repository" (R) at the starting point. Then in each iteration step, HC just finds *one* component of $\mathbf{S}^{(k)}$, i.e., $S_h^{(k)}$ which satisfies the following condition: $h = \arg\max_{i\in\{1,2,\cdots,m\}} \frac{\partial D_i(S_i^{(k)})}{\partial S_i}$. Then HC moves a constant portion δ from the repository to $S_h^{(k)}$: $S_h^{(k)} \leftarrow S_h^{(k)} + \delta$ and $R \leftarrow R - \delta$. This movement looks like "climbing the hill with each step in the steepest dimension". It is easy to see that HC can be taken as a special case of WF which only "walks" in one dimension among the total m dimensions. Consequently, the number of iteration steps of HC is usually as about several (4 – 5) times as that of WF when the same stepsize (δ) is used.

V. Trace-Driven Simulations

A. Trace Dataset

Our trace dataset is got from QQXuanfeng [11], a large-scale CloudP2P file-sharing system. Every online peer reports its *peer status* in a UDP packet to the Bandwidth Scheduler (a centralized server) per 5 minutes, so the cloud bandwidth allocation period is also set as 5 minutes. The *peer status* report is structured as in Fig. 10. During each allocation period, the Bandwidth Scheduler aggregates peer status reports into the corresponding *swarm status* which indicates the status information (including S_i, s_i, l_i and D_i) of a peer swarm i in the allocation period. Because the peer status reports are carried in UDP packets, a small portion (less than 1%) of reports might be lost in transmission, which would influence the performance of the bandwidth allocation algorithm. Therefore, if a peer status report is found to be lost, we simply take its previous peer status report as its substitution.

The simulations are performed on a one-day trace (August 17, 2011) of 1457 peer swarms involving around one million peers. As depicted in Fig. 11, the number of simultaneously online leechers (l) varies between 4K and 50K and the total cloud bandwidth (S) varies between 0.2 and 2.25 GBps. As to a peer swarm i, the required constant parameters (α_i, β_i, f_i) for modeling its performance ($D_i = S_i^{\alpha_i} \cdot l_i^{1-\alpha_i-\beta_i} \cdot {s_i}^{\beta_i} \cdot f_i$) are computed based on its one-day swarm statuses (including 288 swarm statuses in total, where $288 = \frac{24 \text{ hours}}{5 \text{ minutes}}$). After obtaining the performance model for each peer swarm, we simulate the *free-competitive*, *proportional-allocate*, *Ration*, and *FIFA* allocation algorithms to reallocate the cloud bandwidth into each peer swarm during each allocation period, and meanwhile observe their *bandwidth multiplier*, *marginal utility*, and so on.

B. Metrics

- *Bandwidth multiplier* is defined as $\frac{D}{S}$, where D denotes the end hosts' aggregate download bandwidth and S denotes the invested cloud bandwidth. Large bandwidth multiplier means the cloud bandwidth is efficiently used to accelerate the P2P data transfer among end hosts.
- *Marginal utility* is defined as $\mu_i = \frac{\partial D_i}{\partial S_i}$ for a peer swarm i. In **Theorem 1** we have proved that for CloudP2P content distribution, (ideally) the maximum overall bandwidth multiplier implies that the marginal utility of the cloud bandwidth allocated to each peer swarm should be equal. In practice, we want the *relative deviation of marginal utility* ($dev_\mu = \frac{\sum_{i=1}^{m} |\mu_i - \bar{\mu}|}{m \cdot \bar{\mu}}$) among all the peer swarms to be as little as possible.
- *Convergence speed* is denoted by the number of iteration steps for an iterative algorithm (FA, WF or HC) to solve OBAP. Besides, we also care about the ease of use of an iterative algorithm.
- *Control overhead* is denoted by the extra communication cost brought by a bandwidth allocation algorithm, because the allocation algorithm usually needs to collect extra status information from end host peers.

C. Simulation Results

Bandwidth multiplier. Fig. 12 depicts the evolution of the bandwidth multiplier for each allocation algorithm in one day, starting from 0:00 am (GTM+8). We can see that the bandwidth multiplier of *proportional-allocate* is generally close to that of *free-competitive*. On the other hand, *FIFA* and *Ration* obviously outperform free-competitive, with considerable improvements in average (*FIFA*: 2.20 $\rightarrow$ 2.64 = 20% increment, and *Ration*: 2.20 $\rightarrow$ 2.45 = 11% increment). That is to say, the bandwidth multiplier of *FIFA* is 20%, 17% and 8% larger than that of *free-competitive*, *proportional-allocate* and *Ration*, respectively.

Marginal utility. Marginal utility provides the microscopic explanation of the bandwidth multiplier – larger bandwidth

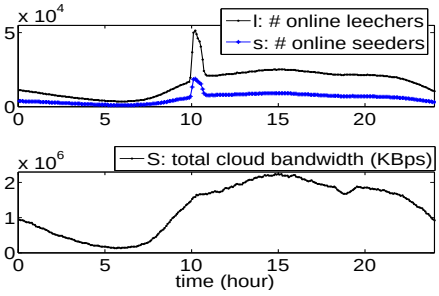


Fig. 11. Evolution of l, s and S over time.

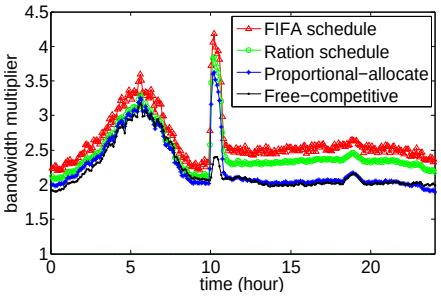


Fig. 12. Evolution of the bandwidth multiplier.

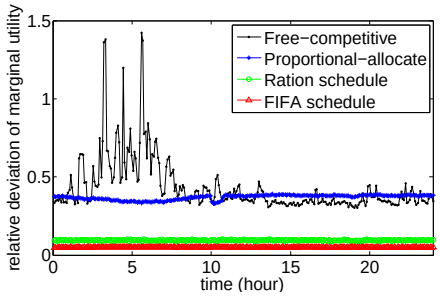


Fig. 13. Evolution of the relative deviation of marginal utility.

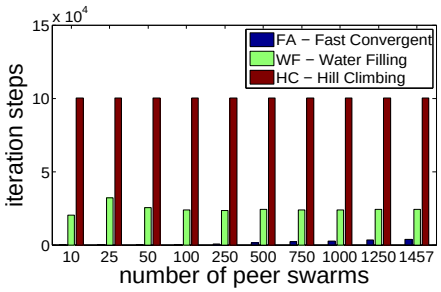


Fig. 14. Convergence speed of FA (τ=0.5, σ=0.01), WF and HC (δ=0.00001).

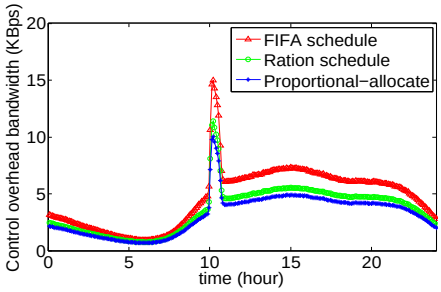


Fig. 15. Evolution of the control overhead bandwidth.

multiplier implies smaller relative deviation of marginal utility (dev_u). Thus, in Fig. 13 we plot the evolution of the relative deviation of marginal utility of all the peer swarms. Clearly, the relative deviation of marginal utility has tight negative correlation with the bandwidth multiplier – *FIFA* has the smallest dev_u and thus its bandwidth multiplier is the biggest.

Convergence speed. We theoretically analyzed the convergence property of FA, WF and HC in Section IV. In this part, to examine their practical convergence speeds, we utilize the *swarm status* data of different number of peer swarms: 10, 25, ..., 1250, 1457. For FA, we simply use the parameters $\tau = 0.5$ and $\sigma = 0.01$ which are well applicable to all the experimented swarm scales. However, for WF and HC, we made a number of attempts to find an appropriate parameter (stepsize) δ which could make WF and HC converge for all the experimented swarm scales. Finally, we found $\delta = 0.00001$ to be an appropriate parameter and plot the corresponding convergence speeds in Fig. 14. We mainly get two findings: (1) FA exhibits nearly-linear convergence speed ($\Theta(m)$) as the swarm scale increases, and FA converges faster than WF and HC as to each swarm scale. (2) WF and HC exhibit nearly-constant ($\Theta(\frac{1}{\delta})$) convergence speed as the swarm scale increases. The bigger δ is, the faster WF and HC converge, but a bigger δ increases the risk that WF and HC may not converge. If the swarm scale further increases to more than 1457, (very possibly) we need to find a smaller δ to satisfy all the swarm scales. On the contrary, the convergence of FA is not sensitive to its parameters and thus it is easier to use.

Control overhead. The control overhead of *free-competitive* is zero since it does not collect peers' status information. In Fig. 11 we recorded the number of online leechers and online seeders per five minutes in one day, so it is easy to compute the control overhead of *proportional-allocate*, *Ration* and *FIFA*. As to *proportional-allocate*, its peer status report does not have the fields of "Cloud Download Bytes" and "P2P Download Bytes" (see Fig. 10) and it only collects leechers' status reports. Different from *proportional-allocate*, *FIFA* collects both leechers' and seeders' status reports. Because *Ration* does not consider the impact of seeders, it only collects leechers' status reports. From Fig. 11 we figure out that *proportional-allocate* collects 4.52M leecher status reports without the fields of "Cloud Download Bytes" and "P2P Download Bytes" (accounting to $4.52M \times 60\text{B} = 271$ MB in total), *Ration* collects 4.52M leecher status reports (accounting to $4.52M \times 68\text{B} = 307$ MB in total), and *FIFA* collects 6.05M peer status reports (accounting to $4.52M \times 68$ Bytes$+1.53M \times 60\text{B} = 399$ MB in total). Averaging the total control overhead into each second, we plot the *control overhead bandwidth* of *FIFA*, *Ration* and *proportional-allocate* in Fig. 15. Obviously, the total control overhead bandwidth of *FIFA* always stays below 15 KBps – even less than a common user's download bandwidth.

VI. Prototype Implementation

Besides the trace-driven simulations, we have also implemented the FIFA algorithm on top of a small-scale prototype system named "CoolFish" [20]. CoolFish is a CloudP2P VoD (video-on-demand) streaming system mainly deployed in the CSTNet [21]. With its "micro cloud" composed of four streaming servers and its P2P organization of end users, CoolFish is able to support an average video bit rate over 700 Kbps (about 50% higher than that of popular commercial P2P-VoD systems).

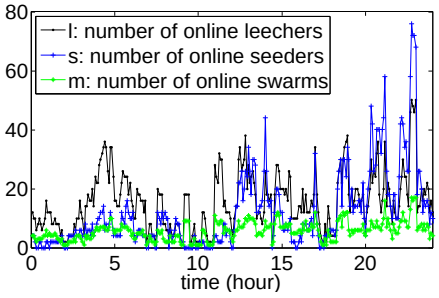


Fig. 16. Evolution of l, s and m in CoolFish.

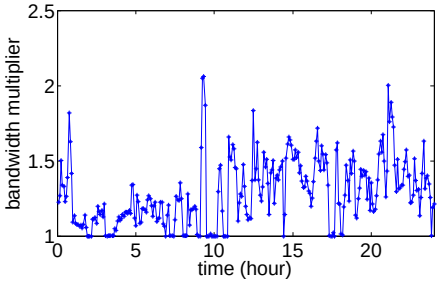


Fig. 17. Evolution of the bandwidth multiplier in CoolFish.

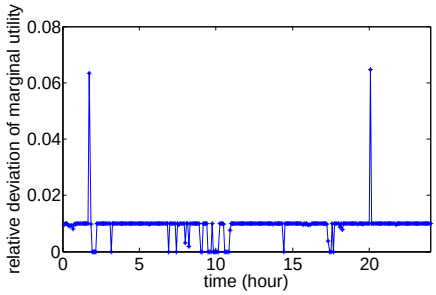


Fig. 18. Evolution of the relative deviation of marginal utility in CoolFish.

Fig. 16 plots the number of online leechers (l), online seeders (s) and online swarms (m) of CoolFish in one day. Obviously, the user scale of CoolFish is much smaller than that of the QQXuanfeng trace, in particular the average number of peers ($\overline{p}$) in one swarm: for the QQXuanfeng trace, $\overline{p} \approx 1M/1457 = 686$, while for CoolFish, $\overline{p} = 1327/46 \approx 29$. Since there are much fewer peer swarms working in CoolFish and a swarm usually possesses much fewer peers, the bandwidth multiplier of CoolFish is remarkably lower and more unstable than that of QQXuanfeng, as illustrated in Fig. 17. When FIFA is applied, the bandwidth multiplier of QQXuanfeng lies between 2.25 and 4.2 while that of CoolFish lies between 1.0 and 2.1. Although the bandwidth multiplier of CoolFish (using FIFA) seems not high, the efficacy of FIFA can still be confirmed from the relative deviation of marginal utility (dev_u stays around 1%, as shown in Fig. 18) since we have proved that very low relative deviation of marginal utility is equal to nearly maximum bandwidth multiplier.

VII. Conclusion and Future Work

As a hybrid approach, CloudP2P inherits the advantages of both cloud and P2P and thus offers a promising alternative in future large-scale content distribution over the Internet. This paper investigates the optimal bandwidth allocation problem (OBAP) of CloudP2P content distribution so as to maximize its bandwidth multiplier effect. Based on real-world measurements, we build a fine-grained performance model for addressing OBAP. And we prove that the bandwidth multiplier is closely related to the marginal utility of cloud bandwidth allocation. Then we propose a fast-convergent iterative algorithm to solve OBAP. Both trace-driven simulations and prototype implementation confirm the efficacy of our solution.

Still some future work remains. For CloudP2P content distribution, this paper focuses on the bandwidth multiplier effect and the corresponding microscopic aspect, i.e., marginal utility. In fact, for some special (but important) CloudP2P content distribution scenarios, we should also take *user satisfaction* or *swarm priority* into account. A download rate up to 30 KBps can be satisfactory for a file-sharing user, while a download rate up to 300 KBps may be still unsatisfactory for an HDTV viewer. Therefore, although FIFA has achieved maximum bandwidth multiplier for the whole CloudP2P system, we cannot say FIFA has brought the maximum user satisfaction. The difficulty lies in that we may need to simultaneously consider several metrics: bandwidth multiplier, user satisfaction and so forth, among which there are conflicts in essence. Consequently, special attention must be paid to a proper tradeoff then.

VIII. Acknowledgements

This work is supported in part by the China 973 Grant. 2011CB302305, China 863 Grant. 2010AA012500, China NSF Grants. 61073015 and 60933005, US NSF Grants CNS-0905037, CNS-1017092 and CNS-1017647.

References

[1] V. Adhikari, S. Jain, Y. Chen, and Z.L. Zhang. "Vivisecting YouTube: An Active Measurement Study," In IEEE INFOCOM, 2012.
[2] A. Cockroft, C. Hicks, and G. Orzell. "Lessons Netflix Learned from the AWS Outage," Netflix Techblog, 2011.
[3] BitTorrent web site. http://www.bittorrent.com.
[4] eMule web site. http://www.emule-project.net.
[5] iKu P2P accelerator. http://c.youku.com/ikuacc.
[6] iTudou P2P accelerator. http://www.tudou.com/my/soft/speedup.php.
[7] Y. Huang, T. Fu, D. Chiu, J. Lui, and C. Huang. "Challenges, design and analysis of a large-scale p2p-vod system," In ACM SIGCOMM, 2008.
[8] C. Wu, B. Li, and S. Zhao. "Multi-channel Live P2P Streaming: Refocusing on Servers," In IEEE INFOCOM, 2008.
[9] F. Liu, S. Shen, B. Li, B. Li, H. Yin, and S. Li. "Novasky: Cinematic-Quality VoD in a P2P Storage Cloud," In IEEE INFOCOM, 2011.
[10] Xunlei web site. http://www.xunlei.com.
[11] QQXuanfeng web site. http://xf.qq.com.
[12] H. Yin, X. Liu, T. Zhan, V. Sekar, F. Qiu, C. Lin, H. Zhang, and B. Li. "Design and deployment of a hybrid CDN-P2P system for live video streaming: experiences with LiveSky," In ACM Multimedia, 2009.
[13] Youku web site. http://www.youku.com.
[14] UUSee web site. http://www.uusee.com.
[15] R. Peterson and E. Sirer. "Antfarm: Efficient Content Distribution with Managed Swarms," In USENIX NSDI, 2009.
[16] S. Boyd. "Convex Optimization," Cambridge University Press, 2004.
[17] Z. Li, Y. Huang, G. Liu, and Y. Dai. "CloudTracker: Accelerating Internet Content Distribution by Bridging Cloud Servers and Peer Swarms," In ACM Multimedia (Doctoral Symposium), 2011.
[18] D.P. Bertsekas. "Nonlinear Programming," Athena Scientific Belmont Press, MA, 1999.
[19] L. Armijo. "Minimization of functions having Lipschitz continuous first partial derivatives," Pacific J. Math. 16 (1), 1966, pages 1 - 3.
[20] CoolFish web site. http://www.cool-fish.org.
[21] CSTNet web site. http://www.cstnet.net.cn.