Understanding the Performance Gap between Pull-based Mesh Streaming Protocols and Fundamental Limits

Chen Feng, Baochun Li
Dept. of Electrical and Computer Engineering
University of Toronto

Bo Li
Dept. of Computer Science
Hong Kong University of Science & Technology

***Abstract*—Pull-based mesh streaming protocols have recently received much research attention, with successful commercial systems showing their viability in the Internet. Despite the remarkable popularity in real-world systems, the fundamental properties and limitations of pull-based protocols are not yet well understood from a theoretical perspective, as there exists no prior work that studies the performance gap between the fundamental limits and the actual performance. In this paper, we develop a unified framework based on trellis graph techniques to mathematically analyze and understand the performance of pull-based mesh streaming protocols, with a particular focus on such a performance gap. We show that there exists a significant performance gap that separates the actual and optimal performance of pull-based mesh protocols. Moreover, periodic buffer map exchanges account for most of this performance gap. Our analytical characterization of the performance gap brings us not only a better understanding of several fundamental tradeoffs in pull-based mesh protocols, but also important insights on the design of practical streaming systems that can achieve high streaming rates and short initial buffering delays.**

I. Introduction

Live peer-to-peer (P2P) streaming has recently witnessed unprecedented growth on the Internet, delivering live streaming content to millions of users at any given time. The essential advantage of P2P streaming is to dramatically increase the number of peers a streaming channel may sustain with dedicated streaming servers. Intuitively, as participating peers contribute their upload bandwidth capacities to serve one another in the same channel, the load on dedicated streaming servers is significantly mitigated.

With a large number of P2P streaming protocols proposed, they generally fall into two strategic categories. *Push-based tree streaming* strategies (*e.g.*, [1]) organize participating peers into one or more multicast trees, and disseminate streaming content along these trees. In contrast, in *pull-based mesh streaming* strategies (*e.g.*, [2]), the streaming content is presented as a series of *segments*, each representing a short duration of playback. Every peer maintains a list of neighboring peers, and *periodically* exchanges segment availability information of streaming buffers (often referred to as *buffer maps*) with its neighboring peers. Based on such information, segments are *pulled* from appropriate neighbors, in order to meet their playback deadlines. Compared to push-based tree strategies, pull-based mesh strategies take advantage of the philosophy that *gossiping* segment availability is more resilient to peer dynamics and simpler to implement, which is commonly adopted in BitTorrent-like file swarming systems. However, such an advantage is achieved at the cost of increased delay of distributing streaming content to all participating peers, due to delays caused by periodic buffer map exchanges [3]. Nevertheless, most real-world systems (*e.g.,* PPLive) are implemented using pull-based mesh strategies, mainly due to their simplicity.

This work was supported in part by Bell Canada through its Bell University Laboratories R&D program. Bo Li's research was supported in part by grants from RGC under the contracts 615608, 616207 and 616406, by a grant from NSFC/RGC under the contract N_HKUST603/07, and by a grant from HKUST under the contract RPC06/07.EG27.

Despite the remarkable popularity in real-world systems, a number of fundamental questions on pull-based mesh protocols are not yet well understood from a theoretical perspective: What are the fundamental limits of the performance of pull-based mesh protocols? How large is the gap between the fundamental limits and the actual performance? What factors account for most of the gap separating the actual and optimal performance of pull-based mesh protocols? In this paper, we seek to mathematically analyze and understand the performance of pull-based mesh protocols, with a particular focus on these important questions. To achieve this objective, we have developed a unified theoretical framework based on the concept of *trellis graphs* [4] and provided a number of new analytical results along this direction.

With trellis graph techniques that have been traditionally used in the network coding literature [4], we have unified the treatment on the analysis of the fundamental limits and the actual performance of pull-based mesh protocols. This provides a solid theoretical foundation for the characterization of the performance gap between the fundamental limits and the actual performance. We perform an in-depth study of several important factors that account for the performance gap and quantify their impact on the performance of pull-based mesh protocols. Our analytical results show that there exists a significant performance gap between the fundamental limits and the actual performance of pull-based mesh protocols. Moreover, periodic buffer map exchanges account for most of the gap that separates the actual and optimal performance. To our knowledge, there has been no existing work in the literature that provides a thorough analytical understanding of pull-based mesh protocols, with a particular focus on both fundamental limits and the performance gap.

The remainder of the paper is structured as follows. In Sec. II, we highlight our original contributions in the context of related work. Sec. III presents our system model along with fundamental constraints in P2P streaming systems. In Sec. IV, we generalize several existing results on the fundamental limits of pull-based mesh protocols. Sec. V performs an in-depth study of important factors that account for the performance gap. Finally, Sec. VI concludes the paper.

II. Related Work

Coolstreaming [2] has pioneered a promising practice of pull-based mesh streaming protocols, which uses the basic concepts of *gossiping* and *swarming* in file sharing systems. Salient advantages of pull-based mesh streaming protocols include the simplicity and resilience to peer dynamics, which make them the choice of many real-world streaming systems. Due to the remarkable popularity, pull-based streaming protocols have received considerable research attention in recent years, with a large number of new protocols proposed (*e.g.,* [3], [5], [6], [7], [8], [9]).

Rather than designing a new streaming protocol and trying to evaluate the design with empirical studies (*e.g.,* Coolstreaming+ [5], [6] and GridMedia [3], [7]), this paper focuses on a thorough analytical understanding on the fundamental properties and limitations of pull-based mesh protocols. Rather than developing an elaborate algorithm to optimize the streaming rate (*e.g.,* [8], [9]), this paper focuses on exploring the performance gap that separates the actual and optimal performance of pull-based mesh protocols.

With respect to fundamental limits of pull-based mesh streaming protocols, the maximum sustainable streaming rate [10] and minimum delay bound [11] have been studied, with centralized scheduling algorithms proposed to approach the optimal values. Our work first generalizes the minimum delay bound in [11] by relaxing the assumption that the upload capacity of each peer should be divisible by the streaming rate. We then apply this result to derive tighter bounds on the maximum sustainable streaming rate.

Recently, Liu *et al.* [12] investigated the fundamental limits of push-based tree streaming protocols. In particular, they studied the effects of restrictions on the number of neighbors each peer can have. They derived several performance bounds on the maximum streaming rate, minimum server load, and minimum tree depth. They also proposed centralized tree-constructing algorithms to achieve these bounds. Our work is in parallel with theirs in that our focus is on pull-based mesh streaming protocols, which are widely adopted in real-world streaming systems.

More importantly, our work not only provides new and tighter performance bounds for pull-based mesh streaming protocols, but also explores the performance gap by examining the effects of periodic buffer map exchanges (*i.e.,* gossiping) and lack of centralized scheduling, which are essential features of pull-based mesh protocols that should not be ignored.

There also exist a small number of analytical papers on the performance modeling and analysis of P2P streaming protocols. For instance, Zhou *et al.* [13] developed a simple stochastic model for streaming systems using pull-based mesh protocols. With this model, they evaluated and compared three different segment selection strategies. However, they did not take into account the feature of periodic buffer map exchanges. In contrast, we have considered this feature in our theoretical framework and demonstrated that it indeed plays a crucial role in the performance of pull-based mesh protocols.

Bonald *et al.* [14] studied several push-based protocols and proved that some of them can achieve near-optimal rate and delay in static streaming systems. Different from their work, we investigate the performance of pull-based protocols in dynamic streaming systems, with a particular focus on the performance gap between the fundamental limits and the actual performance.

III. System Model and Fundamental Constraints

In this section, we present our mathematical model for P2P streaming systems, including the underlying assumptions and the key notations summarized in Table I. Consider a streaming system with N participating peers. In accordance with measurement studies of existing P2P systems (*e.g.,* [15]), we assume peer upload capacities are the only *bottlenecks* in the streaming system. Let U_i denote the upload capacity of peer i ($i \in \{1, 2, \ldots, N\}$). For a given streaming rate R, we define the *relative capacity* u_i of peer i as the *ratio* of the upload capacity U_i to the streaming rate R. Let U_p be the average peer upload capacity and u_p be the relative average peer capacity, which is defined as the ratio of the average peer upload capacity U_p to the streaming rate R.

We assume there is only one streaming server in the system with upload capacity U_s. If multiple streaming servers exist in the system, they can be regarded as a *super streaming server* with upload capacity U_s equaling to the total upload capacities. The relative server capacity u_s is defined as the *ratio* of the server upload capacity U_s to the streaming rate R.

Without loss of generality, we assume time is *slotted* in the sense that it takes one time slot to playback a segment. We further assume peer i can send $\lfloor u_i \rfloor$ segments per time slot, plus one additional segment with probability $u_i - \lfloor u_i \rfloor$, corresponding to its relative capacity u_i. Similarly, the streaming server can send $\lfloor u_s \rfloor$ segments per time slot, plus one additional segment with probability $u_s - \lfloor u_s \rfloor$, corresponding to the *relative server capacity* u_s.

TABLE I
Key Notations in the System Model

N	Number of peers in the system.
R	Streaming rate.
U_i	Upload capacity of peer i.
u_i	Relative capacity of peer i ($= U_i/R$).
U_p	Average peer upload capacity.
u_p	Relative average peer capacity ($= U_p/R$).
U_s	Server upload capacity.
u_s	Relative server capacity ($= U_s/R$).
B	Number of segments in a buffer.

Now we discuss several fundamental constraints and their implications for P2P streaming systems, which are instrumental to establish an in-depth understanding of pull-based mesh protocols. First of all, we observe that *the total bandwidth consumption should not be greater than the total bandwidth supply*. This leads to the following lemma, which has been proved in [10].

Lemma 1: For a streaming system with given server upload capacity U_s and average peer upload capacity U_p, the maximum sustainable streaming rate $R_{\max}$ has the following upper bound:

$$R_{\max} \leq U_p + \frac{U_s}{N},$$

where N is the number of peers in the system.

In particular, if N tends to infinity, then the upper bound in Lemma 1 reduces to $R_{\max} \leq U_p$. In other words, the relative average peer capacity u_p satisfies $u_p \geq 1$ in large-scale streaming systems.

Another fundamental constraint for P2P streaming systems is as follows: *a peer cannot upload a segment until it completes the download of this segment*. As shown in [11], this constraint sets up a limit on how fast a segment can be disseminated to all peers in the system. The special case of homogeneous upload capacity with $u_p = 1$ has been discussed in both [11] and [14]. Here we consider the general homogeneous case where $u_i = u_p \geq 1$ for all $i \in \{1, 2, \ldots, N\}$. (We refer readers to our extended manuscript [16] for the discussions on several typical heterogeneous cases.)

Without loss of generality, we assume only one peer has this segment at the beginning of time slot 0. Afterwards, participating peers with this segment cooperatively disseminate it to other peers subject to their upload capacities. We are particularly interested in the following question: *What is the minimum number of time slots it takes for this segment to reach all N peers in the system*?

We observe that such a segment dissemination process can be well modeled by a *branching process*. More specifically, let Z_n denote the number of peers that have this segment at the beginning of time slot n. We know that $\{Z_n, n = 0, 1, \ldots\}$ is a branching process, with $Z_0 = 1$ and $E[Z_1] = 1 + u_p$. We define the delay function $D(k)$ as the minimum number of time slots it takes for at least k peers to receive this segment. More precisely, $D(k) = \min\{n : Z_n \geq k\}$, for $1 \leq k \leq N$. We are interested in the asymptotic behavior of $D(k)$, when k tends to infinity.

Lemma 2: Let $D(k) = \min\{n : Z_n \geq k\}$ be the minimum delay for at least k peers to receive a particular segment. Then it holds almost surely that

$$D(k) \approx \lceil \log_m k \rceil,$$

when k tends to infinity, where $m = E[Z_1] = 1 + u_p$.

Proof: Let $W_n = Z_n / m^n, n = \{0, 1, \ldots\}$. Kesten and Stigum [17] proved that if $m > 1$ and $E[Z_1 \log Z_1] < \infty$, then the random variables $\{W_n\}$ converge almost surely to a random variable W, for which

$$E[W] = 1,\ Var[W] = \frac{Var[Z_1]}{m^2 - m}.$$

It follows that

$$\frac{Z_{D(k)}}{m^{D(k)}} \rightarrow W,$$

almost surely as $k \rightarrow \infty$. Therefore,

$$\frac{W}{m} \leftarrow \frac{Z_{D(k)-1}}{m^{D(k)}} < \frac{k}{m^{D(k)}} \leq \frac{Z_{D(k)}}{m^{D(k)}} \rightarrow W.$$

In other words,

$$\log_m k - \log_m W \leq D(k) \leq \log_m k - \log_m W + 1.$$

Finally, note that $\log_m k \gg \log_m W$ with high probability when k is sufficiently large. We thus have

$$D(k) \approx \lceil \log_m k \rceil,$$

for sufficiently large k. ■

Lemma 2 suggests that it takes at least $\lceil \log_m N \rceil$ time slots for a particular segment to reach all N peers in the system, when the population N is very large. Intuitively, although $D(N)$ is a random variable in the branching process, $D(N)$ converges to $\lceil \log_m N \rceil$ as N increases, due to the effect of law of large numbers.

IV. A Study of Fundamental Limits

In this section, we proceed to the fundamental limits of pull-based mesh protocols with regard to several important performance metrics. *First,* an *initial buffering delay* must be experienced by a peer when it first joins or switches to a new channel. How do we improve user experience with the shortest initial buffering delay? *Second,* if media segments do not arrive in a timely fashion, they have to be skipped at playback. How do we consistently sustain a high streaming rate with as few playback skips as possible?

We believe these two performance metrics should be given priority when evaluating a pull-based mesh protocol, as they matter most to the *user satisfaction*. We derive several fundamental limits on the initial buffering delay and sustainable streaming rate. In particular, we focus on the *flash crowd* scenario where most of the peers join the system at approximately the same time, just after a new live event has been released. We note that typically, in steady state, it is quite possible to maintain a high streaming rate with short initial buffering delays [6]. Hence, we have mainly focused on the flash crowd scenario as it exercises the streaming systems the most. Our theoretical framework may also be extended to other dynamic scenarios of interest, such as peer churning.

The first question we are interested in is that: *What is the minimum initial buffering delay that should be experienced by a flash crowd of N peers*? Here we give a lower bound on the minimum initial buffering delay. For simplicity, we assume that the relative server capacity u_s is an integer number. This is a reasonable assumption, as the total upload capacities of commercial streaming servers are typically much larger than the streaming rate [18] so that the round error can be ignored.

Lemma 3: Let $D_{\min}$ denote the minimum initial buffering delay that should be experienced by a flash crowd of N peers, for the given relative server capacity u_s and relative peer capacity u_p. Then

$$D_{\min} \geq \lceil \log_m (N/u_s) \rceil + 1,$$

where $m = 1 + u_p$.

Proof: Lemma 2 suggests that it takes at least $\lceil \log_m N \rceil$ time slots for a particular segment to reach all N peers. It follows that it takes at least $\lceil \log_m (N/u_s) \rceil$ time slots for u_s copies of a particular segment to reach all N peers. Note that one additional time slot is required for the streaming server to upload these u_s copies. Therefore, a lower bound for the minimum initial buffering delay $D_{\min}$ is given by $D_{\min} \geq \lceil \log_m (N/u_s) \rceil + 1$. ■

It has been shown in [11] that this lower bound can be achieved by a centralized snow-ball algorithm in the special homogeneous case of $u_p = 1$. We will show later that this lower bound can also be achieved for the general homogeneous case of $u_p \geq 1$. To this end, we introduce the concept of *trellis graphs* and develop a centralized *graph labeling* algorithm that achieves this lower bound. In the extended manuscript [16], we extend this lower bound to several typical heterogeneous cases and discuss how to achieve it using trellis graph techniques.

Trellis Graphs and the Graph Labeling Algorithm

The concept of trellis graphs has been originally proposed in the network coding literature [4]. We now introduce it in the context of P2P streaming systems. Given a streaming system, the associated trellis graph is defined as follows. For each peer p in the system and $t \in \{0, 1, \ldots\}$, the trellis graph includes a node p_t, which corresponds to the associated peer p at the beginning of time slot t. Similarly, for the streaming server S and $t \in \{0, 1, \ldots\}$, the trellis graph includes a node S_t. Fig. 1 illustrates an example of the trellis graph for a streaming system with 6 peers. The trellis graph can be treated as a detailed description of the original streaming system that allows us to conveniently *design* and *analyze* certain streaming protocols for the system.

We are now ready to introduce the graph labeling algorithm that achieves the minimum delay bound. Consider a streaming system with N participating peers. There is a single streaming server which distributes media segments, in playback order, to the N peers subject to its upload capacity. Each segment has a unique sequence number, starting from 1. In other words, the streaming server sends segment i to a number of u_s peers during time slot $i - 1$.

The label on the node p_t in the trellis graph represents the newest segment (*i.e.,* the segment of the largest sequence number) in the playback buffer on peer p at the beginning of time slot t. When peer p has a chance to serve others during time slot t, it would send the segment corresponding to the label on the node p_t. In other words, a peer always selects the newest segment in the buffer to serve others in our graph labeling algorithm.

The basic idea behind our graph labeling algorithm is simple. The minimum delay bound can be achieved, if each node in the trellis graph is appropriately labeled according to certain patterns. We are particularly interested in such labeling patterns that achieve the minimum delay bound.

To this end, we introduce the following notations. Let $m_i(t)$ denote the number of label i on the nodes in the trellis graph at the beginning of time slot t. Denote by $\mathcal{S}_i(t)$ the set of peers that have label i at the beginning of time slot t. Clearly, we have $|\mathcal{S}_i(t)| = m_i(t)$. Algorithm 1 specifies the evolution patterns for $\{m_i(t)\}$ and $\{\mathcal{S}_i(t)\}$ that achieve the minimum delay bound.

Algorithm 1 *Graph Labeling Segment Scheduling Algorithm*

```
1.  Compute the minimum delay bound $D_{\min}$.
    Here, $D_{\min} = \lceil \log_m (N/u_s) \rceil + 1$.
2.  Set current time slot $t = 1$.
3.  Set $p = u_p - \lfloor u_p \rfloor$.
4.  Initialize $m_i(1)$ and $\mathcal{S}_i(1)$. Set
        $m_i(1) = \begin{cases} u_s, & \text{if } i = 1, \\ 0, & \text{otherwise}, \end{cases}$
        $\mathcal{S}_i(1) = \begin{cases} \{1, 2, \ldots, u_s\}, & \text{if } i = 1, \\ \varnothing, & \text{otherwise}. \end{cases}$
5.  while current time slot $t <$ the maximum time slot do
6.      Set $s = t - D_{\min} + 2$.
7.      Compute $m_i(t+1)$ for each $i$ as follows:
8.      if $i = t + 1$ then
9.          $m_i(t+1) = u_s$.
10.     else if $\max\{s+1, 0\} < i < t + 1$ then
11.         $m_i(t+1) = (1 + \lfloor u_p \rfloor) m_i(t) + \text{binornd}(m_i(t), p)$,
            where binornd$(M, p)$ is a binomial random number
            with parameters $M$ and $p$.
12.     else if $i = s + 1 > 0$ then
13.         $m_i(t+1) = \min\{g(t+1),\ (1 + \lfloor u_p \rfloor) m_i(t) + \text{binornd}(m_i(t), p)\}$,
            where $g(t+1) = N - \sum_{j=s+2}^{t+1} m_j(t+1)$.
14.     else
15.         $m_i(t+1) = 0$.
16.     end if
17.     Label the nodes in the trellis graph at the beginning
        of time slot $t + 1$ according to $m_i(t+1)$ such that
        $\{\mathcal{S}_i(t+1)\}$ are pairwise disjoint and $\mathcal{S}_i(t) \subseteq \mathcal{S}_i(t+1)$
        if $m_i(t) \leq m_i(t+1)$.
18.     Schedule the segment transmissions during time slot $t$
        according to $\{\mathcal{S}_i(t)\}$ and $\{\mathcal{S}_i(t+1)\}$.
19.     Set $t = t + 1$.
20. end while
```

To illustrate how to achieve the minimum delay bound using our graph labeling algorithm, we provide the following example.

Example: We consider a streaming system with $N = 6$ peers. We illustrate the associated trellis graph in Fig. 1. We set $u_s = u_p = 1$ in this example. It is easily verified that the minimum delay bound is 4 time slots in this example.

Initially, we have $m_1(1) = 1$ and $\mathcal{S}_1(1) = \{1\}$. Hence, we

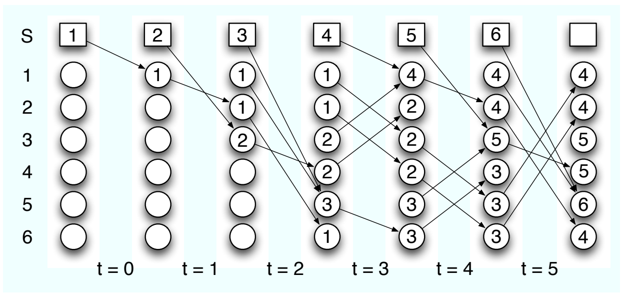


Fig. 1. An example to illustrate segment scheduling using the Graph Labeling algorithm. We set $N = 6$ and $u_s = u_p = 1$ in this example. That is, both the streaming server and the participating peer can upload only one segment during each time slot. The minimum delay bound in this example is 4 time slots. This example shows that each segment can be disseminated to all participating peers within the minimum delay bound (4 time slots) after it is injected by the streaming server.

give label 1 to peer 1 at the beginning of time slot 1. In the first iteration ($t = 1$), we have $m_1(2) = 2$ and $m_2(2) = 1$. We choose $\mathcal{S}_1(2) = \{1, 2\}$ and $\mathcal{S}_2(2) = \{3\}$. Clearly, $\mathcal{S}_1(2)$ and $\mathcal{S}_2(2)$ are disjoint with $\mathcal{S}_1(1) \subseteq \mathcal{S}_1(2)$, which satisfies the requirement of Line 17 in Algorithm 1. Thus, we give label 1 to peer 2 and label 2 to peer 3 at the beginning of time slot 2. Next we schedule the segment transmissions during time slot 1 according to the labels. As shown in Fig. 1, the streaming server uploads segment 2 to peer 3 and peer 1 uploads segment 1 to peer 2.

In the second iteration ($t = 2$), we have $m_1(3) = 3$, $m_2(3) = 2$, and $m_3(3) = 1$. We choose $\mathcal{S}_1(3) = \{1, 2, 6\}$, $\mathcal{S}_2(3) = \{3, 4\}$, and $\mathcal{S}_3(3) = \{5\}$, which satisfies the requirement of Line 17 in Algorithm 1. We label the nodes based on $\{\mathcal{S}_i(3)\}$ at the beginning of time slot 3 and schedule the segment transmissions during time slot 2 (see Fig. 1 for details). We repeat the iteration process in Algorithm 1, which results in a trellis graph with a segment scheduling scheme as shown in Fig. 1.

TABLE II
DOWNLOADING SEGMENTS ON EACH PARTICIPATING PEER

Time Slot	Participating Peers					
	1	2	3	4	5	6
0	1					
1		1	2			
2				2	1, 3	1
3	2, 4	2	1	1		3
4		4	3, 5	3	2	2
5	3	3		5	4, 6	4

Now we verify that the minimum delay bound has been achieved with no playback skips in this example. We focus on the downloading process on each participating peer. Table II shows the downloading segments on each participating peer during each time slot. It is easily verified that each segment can be disseminated to all participating peers within the minimum delay bound (4 time slots) after it is injected by the streaming server.

The correctness of our graph labeling segment scheduling algorithm is shown by the following theorem.

Theorem 1: If the relative peer capacity u_p is an integer number, the minimum delay bound can be achieved with no playback skips. Otherwise, the minimum delay bound can be achieved with some playback skips.

Due to space constraints, we omit the proof here. Interested readers are referred to our extended manuscript [16]. The key idea of the proof is to show that the minimum delay bound can be achieved if Line 17 and 18 in Algorithm 1 are feasible and then prove the feasibility of Line 17 and 18. In [16], we also quantify the playback skips when the relative peer capacity u_p is a fraction. Theorem 1 generalizes the minimum delay bound in [11] by allowing u_p to be a fraction. This relaxation enables us to derive tighter bounds on the maximum sustainable streaming rate, as shown by the following theorem.

Theorem 2: Let $R_{\max}$ denote the maximum sustainable streaming rate for a streaming system with given server capacity U_s and peer capacity U_p, then

$$R_{\max} = \begin{cases} U_p & \text{if } N \leq 2^{B-1}\frac{U_s}{U_p}, \\ R^* & \text{otherwise,} \end{cases} \tag{1}$$

where B is the number of segments in the playback buffer and R^* is the maximum R such that

$$(1 + \frac{U_p}{R})^{B-1}\frac{U_s}{R} \geq N.$$

.

Proof: Theorem 1 states that it takes $D_{\min}$ time slots for a particular segment to arrive at almost all the peers in the system. Therefore, the buffer size B should be no less than the initial buffering delay $D_{\min}$. In other words, we need to enforce the following condition:

$$\log_m(N/u_s) + 1 \leq B. \tag{2}$$

After simple algebraic manipulations, we obtain an equivalent condition as follows:

$$(1 + \frac{U_p}{R})^{B-1}\frac{U_s}{R} \geq N. \tag{3}$$

If we set the streaming rate R to its maximum value U_p, then condition (3) reduces to

$$2^{B-1}\frac{U_s}{U_p} \geq N,$$

completing the proof. ■

The term $2^{B-1}\frac{U_s}{U_p}$ reflects the *scalability* of the streaming system during a flash crowd. It suggests that a streaming system can accommodate a flash crowd of scale less than $2^{B-1}\frac{U_s}{U_p}$ with maximum streaming rate. Hence, the most effective approach to improve the scalability of a streaming system is to increase the buffer size on participating peers.

Previous performance bounds on the sustainable streaming rate (*e.g.*, [12]) have not taken into account the impact of buffer size. In contrast, Theorem 2 characterizes how a limited buffer size affects the sustainable streaming rate, as well as the scalability of the streaming system during a flash crowd. We believe this would shed new insight on the design of playback buffers for practical streaming systems.

V. Understanding the Performance Gap

In this section, we identify several important factors that separate the actual and optimal performance of pull-based mesh protocols, and mathematically quantify their effects on the initial buffering delay and sustainable streaming rate. We mainly focus on the general homogenous case in this section and refer the readers to [16] for the discussions on several typical heterogeneous cases.

A. Effect of Periodic Buffer Map Exchanges

In practical streaming systems, the buffer maps are periodically exchanged so as to maintain an acceptable level of overhead. We are interested in how periodic buffer map exchanges affect the system performance in terms of the initial buffering delay and sustainable streaming rate. Without loss of generality, we assume buffer maps are exchanged every T time slots.

Theorem 3: Assume that buffer maps are exchanged every T time slots. Let $D_{\min}(T)$ denote the corresponding minimum initial buffering delay that should be experienced by a flash crowd of N peers, for the given relative server capacity u_s and relative peer capacity u_p. Then

$$D_{\min}(T) = T\lceil \log_{(1+u_pT)}(N/u_s)\rceil + T.$$

Due to space constraints, we omit the proof and interested readers are referred to our extended manuscript [16]. The main idea of the proof is to show that the minimum initial buffering delay $D_{\min}(T)$ can be achieved by our *generalized graph labeling* algorithm.

Note that the only constraint imposed by periodic buffer map exchanges is that: *a new segment on a peer cannot be uploaded until the peer has a chance to exchange buffer maps with its neighbors*. Therefore, we need to schedule the segment transmissions every T time slots, corresponding to the period of buffer map exchanges. We hence group every continuous T segments into a *segment group*. More precisely, the segment group of sequence number i is defined as the segments $\{(i-1)T+1, (i-1)T+2, \ldots, iT\}$. The *newest segment group* on peer p is defined as the segment group of the largest sequence number in the playback buffer on peer p.

In the first stage of our generalized graph labeling algorithm, we schedule the transmissions of segment groups rather than individual segments. The output of the first stage of the algorithm is a trellis graph with labels representing the newest segment group on each node. In the second stage of our generalized graph labeling algorithm, we further schedule the transmissions of individual segments within each segment group based on the trellis graph obtained in the first stage of the algorithm. The final output of our algorithm is a trellis graph with labels representing individual segments being uploaded. Note that a node in the final trellis graph may have multiple labels, as the associated peer may upload several different segments in T time slots.

In the extended manuscript [16], we formally describe our generalized graph labeling algorithm and prove its correctness. To illustrate how to schedule segments using our generalized graph labeling algorithm, we provide the following example.

Example: We consider a streaming system with $N = 6$ peers. We set $T = 2$ and $u_s = u_p = 1$ in this example. That is, buffer maps are exchanged every 2 time slots and both the streaming server and the participating peer can upload only one segment during each time slot. It is easily verified that the minimum initial buffering delay $D_{\min}(2)$ equals to 6 time slots in this example.

We schedule the segment transmissions every 2 time slots in this example, corresponding to the period of buffer map exchanges. Hence, we change the time unit from 1 time slot to 2 time slots. The associated trellis graph is defined based on the new time unit, as shown in Fig. 2.

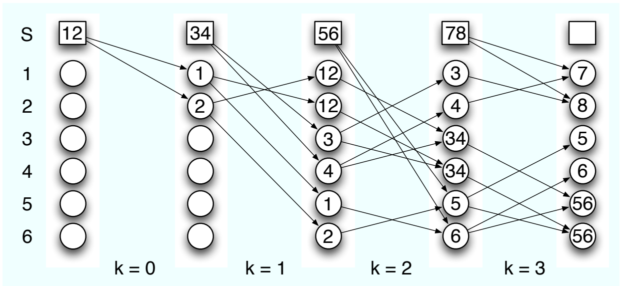


Fig. 2. An example to illustrate segment scheduling using the generalized Graph Labeling algorithm. We set the number of peers N to 6, the period of buffer map exchanges T to 2. Both the relative server capacity u_s and the relative peer capacity u_p are set to 1. In this example, the minimum initial buffering delay is 6 time slots, and the time unit is 2 time slots. This example shows that each segment can be disseminated to all participating peers within the minimum initial buffering delay (6 time slots or 3 time units) after it is injected by the streaming server.

TABLE III
Downloading Segments on Each Participating Peer

Time Unit	Participating Peers					
	1	2	3	4	5	6
0	1	2				
1	2	1	3	4	1	2
2	3	4	1, 2, 4	1, 2, 3	2, 5	1, 6
3	4, 7	3, 8	5	6	3, 4, 6	3, 4, 5

Fig. 2 shows the final output of our generalized graph labeling algorithm for this example: a trellis graph with labels on each node representing the segments being uploaded. Table III shows the downloading segments on each participating peer during each time unit. It is easily verified that each segment can be disseminated to all participating peers within the minimum initial buffering delay (6 time slots or 3 time units) after it is injected by the streaming server.

Theorem 3 characterizes the performance gap in terms of the initial buffering delay due to periodic buffer map exchanges. This is closely related to the fundamental overhead-delay tradeoff ([3], [7]). To minimize the initial buffering delay, each peer has to exchange buffer maps in a timely fashion, resulting in an excessive overhead. On the other hand, to reduce the overhead, peers need to exchange buffer maps periodically, leading to a considerable delay.

A by-product of Theorem 3 is the first *exact* characterization of the overhead-delay tradeoff during flash crowds. As the overhead is inversely proportional to the period of buffer map exchanges, the overhead-delay tradeoff can be quantified by investigating how the initial buffering delay increases with the period of buffer map exchanges, which has been answered completely in Theorem 3. For large-scale streaming systems, Theorem 3 can be restated as follows.

Corollary 1: Assume that buffer maps are exchanged every T time slots. The corresponding minimum initial buffering delay $D_{\min}(T)$ that should be experienced by a *large* flash crowd can be approximated as follows:

$$\frac{D_{\min}(T)}{D_{\min}} \approx \frac{T}{\log_m(1+u_pT)},$$

where $m = 1 + u_p$ and $D_{\min}$ is the fundamental limit on the initial buffering delay.

Proof: From Theorem 3, we have

$$D_{\min}(T) = T\lceil \log_{(1+u_pT)}(\frac{N}{u_s})\rceil + T.$$

The fundamental limit $D_{\min}$ of the initial buffering delay is given by

$$D_{\min} = \lceil \log_{(1+u_p)}(\frac{N}{u_s})\rceil + 1.$$

Therefore,

$$\begin{aligned}\frac{D_{\min}(T)}{D_{\min}} &= \frac{T\lceil \log_{(1+u_pT)}(N/u_s)\rceil + T}{\lceil \log_{(1+u_p)}(N/u_s)\rceil + 1} \\ &\approx \frac{T\log_{(1+u_pT)}(N/u_s)}{\log_{(1+u_p)}(N/u_s)} \text{ (when } N \text{ is large)} \\ &= \frac{T}{\log_m(1+u_pT)}.\end{aligned}$$

■

Corollary 1 suggests that the performance gap in terms of the initial buffering delay increases significantly with the period of buffer map exchanges. However, it does not depend on the scale of the system. The periodic buffer map exchanges also lead to a performance gap in terms of the sustainable streaming rate, as shown by the following theorem.

Theorem 4: Assume that buffer maps are exchanged every T time slots. Let $R_{\max}(T)$ denote the corresponding maximum sustainable streaming rate for a streaming system with given server capacity U_s and peer capacity U_p. Then

$$R_{\max}(T) = \begin{cases} U_p & \text{if } N \le (1+T)^{\frac{B}{T}-1}\frac{U_s}{U_p}, \\ R^*(T) & \text{otherwise,} \end{cases} \quad (4)$$

where B is the number of segments in the buffer and $R^*(T)$ is the maximum R such that

$$(1 + T\frac{U_p}{R})^{\frac{B}{T}-1}\frac{U_s}{R} \ge N.$$

.

The proof of Theorem 4 is a straightforward extension of that of Theorem 2. We omit the details here due to space constraints. Theorem 4 suggests that under the constraint of periodic buffer map exchanges, a streaming system can accommodate a flash crowd of scale less than $(1+T)^{\frac{B}{T}-1}\frac{U_s}{U_p}$ with the maximum streaming rate. It unveils the overhead-scalability tradeoff in pull-based mesh protocols during flash crowds. This tradeoff has received little attention in the literature, as it is not as intuitive as the overhead-delay tradeoff.

B. Effect of Lack of Centralized Scheduling

In practical streaming systems, the participating peers employ simple decentralized schemes in order to maintain the simplicity. Intuitively, a simple decentralized scheme would lead to a certain degree of performance loss, compared to a sophisticated centralized streaming scheme that approaches the optimal performance (*e.g.,* our generalized graph labeling algorithm in Sec. V-A). Such performance loss is referred to as the performance gap due to lack of centralized scheduling. We are interested in characterizing this performance gap and comparing it with the performance gap caused by periodic buffer map exchanges.

The first question is that: What pull-based streaming scheme should be analyzed in this section? We naturally prefer a simple streaming scheme with minimum performance gap to the optimal centralized scheme and minimum disruption to traditional pull-based protocols that real-world streaming systems use. To achieve this objective, we choose to make minimum modifications to traditional pull-based protocols, based on the insights from our generalized graph labeling algorithm. Specifically, we slightly modify the segment selection component of the pull-based protocol implemented in [3] and keep other components of that protocol (*e.g.,* overlay construction and peer selection) unchanged.

The pull-based protocol in [3] employs a random segment selection strategy: when a downstream peer has a chance to request a segment from an upstream peer, it *randomly* selects a missing segment in its own playback buffer. In our simple pull-based streaming scheme, we adopt a *newest group first* segment selection strategy: when a downstream peer has a chance to request a segment from an upstream peer, it randomly selects a segment in the *newest segment group* (defined in Sec. V-A) on that upstream peer, which is inspired by our generalized graph labeling algorithm.

We again use trellis graph techniques to study the performance of our simple streaming scheme. For ease of presentation, we introduce the following notations. We say a peer has segment group i if it has at least one segment in that group. The label on each node in the trellis graph represents the newest segment group on the associated peer. Let $m_i(k)$ denote the average number of label i on the nodes in the trellis graph at the beginning of time unit k. Let $q_i(k)$ denote the average number of peers that have segment group i at the beginning of time unit k. We are interested in the evolution patterns for $\{m_i(k)\}$ and $\{q_i(k)\}$ in the trellis graph.

Proposition 1: The evolution patterns for $\{m_i(k)\}$ and $\{q_i(k)\}$ in the trellis graph using our simple streaming scheme can be approximated by the following difference equations.

$$\begin{aligned}
q_{k+1}(k+1) &= u_s T \\
q_k(k+1) &= q_k(k) + u_p T m_k(k)\big(1 - \frac{q_k(k)}{N}\big) \\
q_{k-i}(k+1) &= q_{k-i}(k) + u_p T m_{k-i}(k)\big(1 - \frac{q_{k-i}(k)}{N}\big) \\
m_k(k) &= q_k(k) \\
m_{k-i}(k) &= q_{k-i}(k) \prod_{j=0}^{i-1} \big(1 - \frac{q_{k-j}(k)}{N}\big)
\end{aligned}$$

Proof: Consider the system performance during time unit k. Note that segment group $k+1$ is the newest group on the streaming server during this time unit. Thus, we have $q_{k+1}(k+1) = u_s T$. Let us turn to segment group k. Note that there are a total of $m_k(k)$ peers with label k that would upload segment group k during this time unit. If a peer without any segment in group k receives such a segment during time unit k, then the number of peers that have segment group i at the beginning of time unit $k+1$ would be increased by 1. Therefore, we have $q_k(k+1) = q_k(k) + u_p T m_k(k)(1 - q_k(k)/N)$. This argument also applies to other segment groups.

Now let us turn to the number of labels at the beginning of time unit k. Note that a peer has label $k-i$ at the beginning of time unit k if and only if segment group $k-i$ is the newest group in its buffer. Assume that for any given peer, the event that it has segment group $k-i$ is independent of the event that it has any other segment group. Thus, the probability that segment group $k-i$ is the newest group for a given peer can be approximated by $\prod_{j=0}^{i-1}(1 - q_{k-j}(k)/N)$. We therefore have $m_{k-i}(k) = q_{k-i}(k) \prod_{j=0}^{i-1}(1 - q_{k-j}(k)/N)$. ■

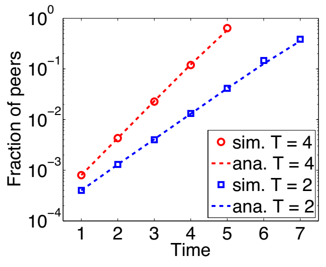
The following corollary characterizes the evolution of peers that have a given segment group.

Corollary 2: For a given segment group, let $f(k)$ denote the fraction of peers that have this segment group k time units later after it is injected by the streaming server. Then the evolution of $f(k)$ can be approximated as follows.

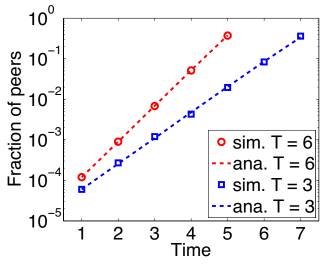
$$\begin{aligned}
f(1) &= u_s T/N \\
f(i+1) &= f(i) + u_p T f(i)\big(1 - f(i)\big) \prod_{j=1}^{i-1} \big(1 - f(j)\big)
\end{aligned}$$

Fig. 3 compares the fraction of peers that have a given segment group obtained by Corollary 2 and by running large-scale simulations, in a number of different scenarios. We observe that our analytical approximations correctly predict the evolution behavior of the system, both qualitatively and quantitatively. As seen in Fig. 3, the fraction of peers that have a given segment group increases almost exponentially under our newest group first strategy. This implies that our simple streaming scheme is able to approach the performance of our generalized graph labeling algorithm, in which the fraction of peers that have a given segment group increases exponentially.

Now we are ready to study the performance gap due to lack of centralized scheduling and compare it with the performance gap caused by periodic buffer map exchanges. To achieve this objective, we compare the fundamental limits (referred to as *limit*) obtained in Sec. IV with the performances of our generalized graph labeling algorithm (referred to as *period*) and our simple streaming scheme (referred to as *simple*). The performance difference between *limit* and *period* reflects the performance gap due to periodic buffer map exchanges, as buffer maps are periodically exchanged in our generalized graph labeling algorithm. The performance difference between *period* and *simple* reflects the performance gap due to lack of centralized scheduling, as explained earlier.

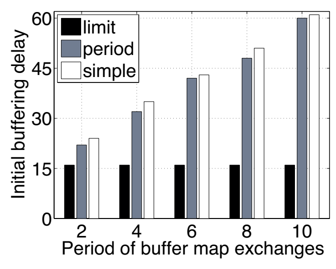


(a)

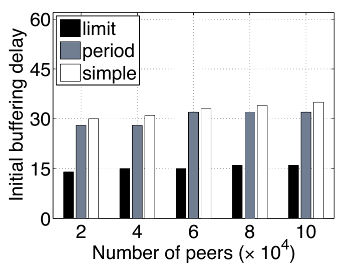


(b)

Fig. 3. Validation of the difference equation approximations for our simple pull-based streaming scheme. In (a), we set the relative server capacity u_s to 2 and the relative peer capacity u_p to 1.1. The number of peers in the system is set to 10000. In (b), we set the relative server capacity u_s to 2 and the relative peer capacity u_p to 1.1. The number of peers in the system is set to 100000. We observe that our analytical approximations match the simulation results quite well.



(a)



(b)

Fig. 4. Comparisons of the performance gap in terms of the initial buffering delay. In (a), we set the relative server capacity u_s to 2 and the relative peer capacity u_p to 1.1. The number of peers in the system is set to 100000. We vary the period of buffer map exchanges from 2 time slots to 10 time slots. In (b), we set the relative server capacity u_s to 2 and the relative peer capacity u_p to 1.1. The period of buffer map exchanges is set to 4 time slots. We vary the number of peers in the system from 20000 to 100000. We observe that periodic buffer map exchanges account for most of the gap that separates the actual and optimal initial buffering delay in pull-based mesh protocols. In contrast, the lack of centralized scheduling only results in a small degree of performance loss. Moreover, the performance gap increases significantly as the period of buffer map exchanges increases, but is insensitive to the number of peers in the system.

Fig. 4 illustrates the comparisons of the performance gap between the fundamental limits and the actual performance of pull-based mesh protocols in terms of the initial buffering delay. From Fig. 4, we observe that periodic buffer map exchanges account for most of the gap that separates the actual and optimal initial buffering delay in pull-based mesh protocols. In contrast, the lack of centralized scheduling only results in a small degree of performance loss. Moreover, we observe that the performance gap increases significantly as the period of buffer map exchanges increases, but is insensitive to the number of peers in the system. This observation agrees with Corollary 1.

We now turn to the performance gap in terms of the sustainable streaming rate. As shown in Fig. 5, we observe again that periodic buffer map exchanges account for most of the performance gap, while the lack of centralized scheduling only leads to a small degree of performance loss. Furthermore, the sustainable streaming rate deteriorates significantly after the system scale exceeds a threshold and this threshold depends critically on the period of buffer map exchanges. This confirms the overhead-scalability tradeoff in Sec. V-A. Both Fig. 4 and Fig. 5 suggest that periodic buffer map exchanges play a critical role in the actual performance of pull-based mesh protocols and should deserve special treatment in the system design. Moreover, simple pull-based streaming schemes with fine tuned system parameters are good enough to achieve high streaming rates and short initial buffering delays, as the lack of centralized scheduling only leads to a small degree of performance loss.

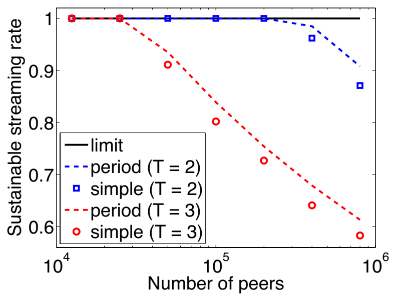


Fig. 5. Comparisons of the performance gap in terms of the sustainable streaming rate. We set the server upload capacity U_s to be 2 times of the average peer upload capacity U_p. We vary the number of peers in the system from 12500 to 800000. We observe again that periodic buffer map exchanges account for most of the performance gap, while the lack of centralized scheduling only leads to a small degree of performance loss. Furthermore, the sustainable streaming rate deteriorates significantly after the system scale exceeds a threshold and this threshold depends critically on the period of buffer map exchanges.

VI. Conclusion

The unique strength of pull-based mesh streaming protocols is the simplicity, which makes them the choice of many real-world streaming systems. The essential features of pull-based mesh protocols include periodic buffer map exchanges and lack of centralized scheduling. These features contribute most to the simplicity of pull-based mesh protocols, but at the same time, lead to a performance gap between the fundamental limits and the actual performance.

In this paper, we have developed a unified framework based on trellis graph techniques to mathematically analyze and understand the performance of pull-based mesh protocols, with a particular focus on such a performance gap. Our analytical results show that there exists a significant performance gap between the fundamental limits and the actual performance of pull-based mesh protocols. Moreover, periodic buffer map exchanges account for most of the gap that separates the actual and optimal performance of pull-based mesh protocols. In contrast, the lack of centralized scheduling only results in a small degree of performance loss.

Our analytical characterization of the performance gap brings us not only a better understanding of several fundamental tradeoffs in pull-based mesh protocols, but also important insights on the design of practical streaming systems that can achieve high streaming rates and short initial buffering delays. For example, we give the first exact characterization of the overhead-delay tradeoff in pull-based mesh protocols during flash crowds. We further unveil the overhead-scalability tradeoff that receives little attention in the literature. More importantly, our analytical results suggest that simple pull-based streaming protocols are good enough to achieve high streaming rates and short initial buffering delays, with fine tuned system parameters, such as the buffer size and the period of buffer map exchanges.

References

[1] V. Venkataraman, K. Yoshida, and P. Francis, "Chunkyspread: Heterogeneous Unstructured Tree-based Peer-to-Peer Multicast," in *Proc. of IEEE International Conference on Network Protocols (ICNP)*, 2006.

[2] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum, "CoolStreaming/DONet: A Data-Driven Overlay Network for Efficient Live Media Streaming," in *Proc. of IEEE INFOCOM*, 2005.

[3] M. Zhang, Q. Zhang, L. Sun, and S. Yang, "Understanding the Power of Pull-based Streaming Protocol: Can We Do Better?" *IEEE J. on Sel. Areas in Communications*, December 2007.

[4] R. Ahlswede, N. Cai, S. R. Li, and R. W. Yeung, "Network Information Flow," *IEEE Transactions on Information Theory*, July 2000.

[5] B. Li, S. Xie, G. Y. Keung, J. Liu, I. Stoica, H. Zhang, and X. Zhang, "An Empirical Study of the Coolstreaming+ System," *IEEE J. on Sel. Areas in Communications*, December 2007.

[6] B. Li, S. Xie, Y. Qu, G. Y. Keung, C. Lin, J. Liu, and X. Zhang, "Inside the New Coolstreaming: Principles, Measurements and Performance Implications," in *Proc. of IEEE INFOCOM*, 2008.

[7] M. Zhang, Y. Xiong, Q. Zhang, and S. Yang, "A Peer-to-Peer Network for Live Media Streaming: Using a Push-Pull Approach," in *Proc. of ACM Multimedia 2005*, November 2005.

[8] M. Zhang, C. Chen, Y. Xiong, Q. Zhang, and S. Yang, "Optimizing the Throughput of Data-Driven based Streaming in Heterogeneous Overlay Network," in *Proc. of ACM Multimedia Modeling 2007*, January 2007.

[9] L. Massoulie, A. Twigg, C. Gkantsidis, and P. Rodriguez, "Randomized Decentralized Broadcasting Algorithms," in *Proc. of IEEE INFOCOM*, 2007.

[10] R. Kumar, Y. Liu, and K. W. Ross, "Stochastic Fluid Theory for P2P Streaming Systems," in *Proc. of IEEE INFOCOM*, 2007.

[11] Y. Liu, "On the Minimum Delay Peer-to-Peer Video Streaming: How Realtime Can It Be?" in *Proc. of ACM Multimedia*, 2007.

[12] S. Liu, R. Z. Shen, W. Jiang, J. Rexford, and M. Chiang, "Performance Bounds for Peer-Assisted Live Streaming," in *Proc. of ACM SIGMETRICS*, 2008.

[13] Y. Zhou, D. M. Chiu, and J. C. Lui, "A Simple Model for Analyzing P2P Streaming Protocols," in *Proc. of IEEE International Conference on Network Protocols (ICNP)*, 2007.

[14] T. Bonald, L. Massoulie, F. Mathieu, D. Perino, and A. Twigg, "Epidemic Live Streaming: Optimal Performance Trade-Offs," in *Proc. of ACM SIGMETRICS*, 2008.

[15] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang, "Measurements, Analysis, and Modeling of BitTorrent-like Systems," in *Proc. of Internet Measurement Conference*, 2005.

[16] C. Feng and B. Li, "Understanding the Performance Gap between Pull-based Mesh Streaming Protocols and Fundamental Limits," http://www.eecg.toronto.edu/∼bli/techreports/gap.pdf, ECE, University of Toronto, Tech. Rep., 2008.

[17] H. Kesten and B. Stigum, "A Limit Theorem for Multidimensional Galton-Watson Processes," *Ann. Math. Statist.*, vol. 37, 1966.

[18] C. Wu, B. Li, and S. Zhao, "Multi-channel Live P2P Streaming: Refocusing on Serves," in *Proc. of IEEE INFOCOM*, 2008.