A Unified Framework for Sub-stream Scheduling in P2P Hybrid Streaming Systems and How to Do Better?*

Zhenjiang Li[1], Yao Yu[2], and Xiaojun Hei[3] and Danny H.K. Tsang[1]

[1] Department of Electronic and Computer Engineering
Hong Kong University of Science and Technology
[2] Department of Electronics Science and Engineering
Nanjing University
[3] Department of Electronics and Information Engineering
Huazhong University of Science and Technology
lzjiang@ust.hk,allanyu@ese.nju.edu.cn,heixj@hust.edu.cn,eetsang@ece.ust.hk

Abstract. The pull-push hybrid P2P streaming, as an emerging and promising approach, has achieved some success in delivering live video traffic. The sub-stream scheduling problem is a key design issue in a hybrid system. In this paper, we propose a max-flow model for unifying this sub-stream scheduling problem. We find that the sub-stream scheduling problem in GridMedia, CoolStreaming+ and LStreaming can be formulated into a special case of the proposed max-flow model. We further propose a min-cost flow model to combat peer heterogeneity in scheduling sub-streams. This min-cost flow model is implemented in a prototype system, LStreaming+. The accuracy of the max-flow model and the outstanding performance of LStreaming+ are demonstrated by extensive simulations. We also show that LStreaming+ achieves excellent performance in prototype experiments.

Keywords: P2P streaming, pull-push, hybrid, max-flow, min-cost flow

1 Introduction

P2P streaming architectures have advanced in two major approaches: pull-based (mesh-pull) approach versus push-based (tree-push) approach. The pull-based systems apply a simple design principle and combat peer churn and peer heterogeneity in dynamic P2P environment. However, these pull-based systems often suffer from high traffic overhead and long start-up delay [1]. In contrast, push-based systems may achieve high throughput, low overhead and small delay if the tree structure does not break down due to peer churn. Recently, researchers are exploring a new class of pull-push hybrid architectures. Some hybrid systems, including GridMedia [2], CoolStreaming+ [3] and LStreaming [4, 5], have already achieved performance improvement compared with pull-based systems in terms of throughput, signaling overhead and video viewing quality.

* This work is supported by RGC Earmarked Research Grant 620306.

The general idea of pull-push streaming works as follows. The original video is divided into data chunks and each chunk is assigned with a unique chunk number (e.g. 0, 1, 2, ...). Chunks are further organized into sub-streams logically. Suppose there are total S sub-streams, sub-stream 1 contains chunks 1, $1+S$, $1+2S$, ... Video chunks may be delivered using the chunk-pulling module and the sub-stream pushing module. Each peer starts up with pulling chunks. After the initial time, the sub-stream pushing module starts to work. Then peers need to decide which sub-stream should be obtained from which neighbors. This decision process is known as **sub-stream scheduling**. After the sub-stream pushing module is enabled, the chunk-pulling module is still working to serve as backup to download those missing chunks when their playback deadline is approaching. In a hybrid system, the sub-stream pushing module sets up a local tree structure between peers based on the topology formed by the chunk-pulling module. In sub-stream scheduling, peers actively select one neighbor among all candidate neighbors to obtain a sub-stream. However, there exists no explicit "scheduling" module in conventional tree-based systems. The neighbor-relationship is formed passively after peers successfully join single or multiple trees in conventional tree-based systems. The previously proposed sub-stream scheduling schemes [2–5] are mainly designed experimentally. We find that these heuristic designs have inherent limitations. Liu *et al.,* proposed to allocate sub-streams via bipartite graphs in [6]. The protocol in [6] is tailored against free-riders without an in-depth study on the sub-stream scheduling problem.

In this paper, we first highlight the importance of the sub-stream scheduling in hybrid systems on reducing overhead, improving throughput and maximizing video viewing quality. To the best of our knowledge, this is the first work thoroughly studying the sub-stream scheduling problem in P2P hybrid systems. Then we propose a *max-flow* model for unifying the sub-stream scheduling problem in the existing hybrid systems including GridMedia, CoolStreaming+ and LStreaming. We demonstrate that the sub-stream scheduling module in these three hybrid systems actually solves one special case of the proposed max-flow model, respectively. Our proposed max-flow model is solvable in polynomial time and provides useful insights for a better sub-stream scheduling design, in which we propose a *min-cost flow* model to better utilize heterogenous peers. We implement the proposed min-cost flow model in a prototype system, LStreaming+. We show that this min-cost flow scheduling scheme outperforms the existing sub-stream scheduling schemes. Compared with GridMedia and LStreaming, the total overhead reduction of LStreaming+ reaches as high as 43.9% and 20.0%, respectively. In addition, LStreaming+ achieves the highest throughput and the smoothest video playback.

The rest of this paper is organized as follows. In Section 2, we present the max-flow model and the corresponding special cases for three existing systems, GridMedia, CoolStreaming+ and LStreaming. Then we propose the min-cost flow model in Section 3 and evaluate the performance using simulations and prototype experiments in Section 4. Finally, conclusions are made in Section 5.

2 Problem Statement and Formulation

The general scheduling pattern of peers in a hybrid system is shown in Fig. 1. During the initial τ_0 time, only the chunk-pulling module is enabled so as to start up the streaming process as quickly as possible. After τ_0, the sub-stream pushing module starts to work concurrently with the chunk-pulling module. Peers will conduct sub-stream scheduling for the sub-stream pushing module after τ_t time. Whether the duration of each τ_t $(t = 1, 2, ...)$ is set to be constant depends on a particular system design. If sub-stream scheduling is triggered periodically (e.g. the time-driven type), each time interval is identical; otherwise, the time intervals can vary (e.g. the event-driven type).

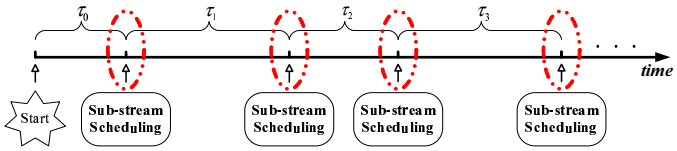

Fig. 1. The general scheduling pattern of peers in hybrid systems

2.1 Sub-stream scheduling Problem

In pull-push hybrid systems, the sub-stream scheduling decisions are generally controlled by the *peer qualification rule* and the *token number*. Based on the *peer qualification rule*, peers need to determine qualified neighbors in terms of the sub-stream availability to obtain sub-streams; then, each sub-stream is scheduled to be obtained from only one qualified neighbor. These selected qualified neighbors, called **push-neighbors**, push chunks of sub-streams to the requested peer. The *peer qualification rule* may be set differently in different systems. After selecting qualified neighbors, on each sub-stream scheduling event, every peer (e.g. peer i in Fig. 2) needs to calculate the maximum number of sub-streams that can be obtained from each neighbor. In existing systems, this number is usually computed based on the end-to-end bandwidth between two peers. In this paper, we call this number *token number*, denoted as T_{ih}. It represents neighbor h can be scheduled to upload at most T_{ih} sub-streams to peer i.

We now examine peer i in Fig. 2 for instance to explain the design issues in the *sub-stream scheduling*. Peer i has four neighbors and five sub-streams needed to be scheduled. For sub-stream 0, peer i schedules it to be obtained from neighbor 1 which is the only peer qualified for this sub-stream. On scheduling sub-streams 1 and 2, if random selection or sequential selection is used, both sub-streams 1 and 2 can be downloaded from neighbor 2. Although neighbor 3 has 2 tokens left, peer i cannot download any sub-streams from peer 3. Therefore, the chunks in sub-streams 3 and 4 can only be downloaded by the chunk-pulling module

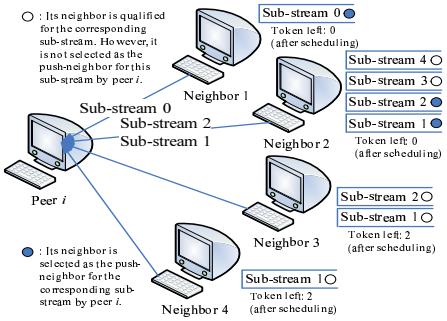

Fig. 2. Inefficient scheduling

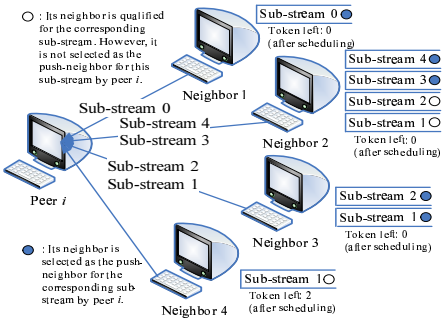

Fig. 3. Efficient scheduling

later. As a result, more signaling overhead will incur. Furthermore, note that the portion of the buffer, in which the missing chunks are requested by the chunk-pulling module, is usually close to the playback pointer. If too many sub-streams are not successfully scheduled (without finding a qualified push-neighbor), then a large amount of chunks can only be downloaded by the chunk-pulling module; therefore, the chance that chunks miss their playback deadlines increases and the viewing quality deteriorates accordingly. If we change to the strategy in Fig. 3, all sub-streams can be successfully scheduled. Most of the chunks will be delivered through multiple trees and only a few of them may be left to the chunk-pulling module. Hence, the viewing quality is improved with reduced signaling overhead. Although scheduling in Fig. 3 is more efficient than that in Fig. 2, a problem still remains. In Fig. 3, sub-stream 1 can be scheduled to either qualified neighbor 3 or 4, it is not clear which peer is more appropriate. This issue will be discussed in Section 3.

We have presented the importance of sub-stream scheduling on reducing signaling overhead and improving viewing quality. Since the total traffic overhead is due to signaling messages and redundant video chunks, we now briefly discuss the impact of sub-stream scheduling on video redundancy overhead. Note that a push-neighbor continues to push chunks if it does not receive any canceling messages. If the sub-stream scheduling is not conducted appropriately, peers in the system will suffer from frequent switching of push-neighbors (we call this phenomena **push-neighbor switching**). Redundant video download occurs as follows. Chunks have been pushed out from previous push-neighbors before push-neighbor switching. Due to the propagation delay and the network buffering effect, these chunks may be still in transmit. The same chunks may be disseminated via new push-neighbors later, then redundant download occurs. Frequent push-neighbor switching can incur significant video redundancy overhead, which will be further examined in Section 4. In summary, the sub-stream scheduling highly impacts viewing quality, signaling and video redundancy overhead. In the next subsection, we propose a unified model on sub-stream scheduling based on our study of existing hybrid systems.

2.2 Modeling

To facilitate the discussions, we tabulate the notations in Table 1. The default values are used in our simulation and prototype unless specified explicitly.

Table 1. Notations

e^i_{hj}	Decision variable to indicate whether peer i obtains sub-stream j from neighbor h
I^i_{hj}	Decision variable to indicate whether neighbor h is qualified for sub-stream j according to the *peer qualification rule* of peer i
Γ_i	Set of all sub-streams for peer i to schedule
Ω_i	Set of neighbor peers of peer i
R	Playback rate of the streaming (e.g. 300kbps)
τ	Period of sub-stream scheduling for LStreaming+ (e.g. 10s)
χ	A large integer (e.g. 10^4)
L_{hi}	End-to-end bandwidth from neighbor h of peer i to peer i (kbps)
P^i_{hj}	Largest chunk number already available for sub-stream j at neighbor h of peer i
Q_{ij}	Starting chunk number needed by peer i for sub-stream j
T_{ih}	Token number for neighbor h of peer i in sub-stream scheduling
B	Number of chunks in 1 second video (e.g. 30)
S	Total number of sub-streams in the system (e.g. 15)

For each sub-stream scheduling, peer i needs to schedule $|\Gamma_i|$ sub-streams in total, where Γ_i represents the set of all sub-streams for peer i to schedule. For any sub-stream j in Γ_i, the set $\{h|I^i_{hj} = 1, \forall h \in \Omega_i\}$ includes all the qualified neighbors, from which peer i can obtain sub-stream j. Ω_i is the set of all neighbors of peer i. If multiple neighbors are qualified for one sub-stream, peers in existing systems will randomly select one neighbor to obtain this sub-stream. Thus, we assume there is no difference among all these qualified neighbors in scheduling. When peer i conducts *sub-stream scheduling*, it basically aims to maximize the number of sub-streams to be scheduled successfully; hence, we propose the following max-flow model for peer i to unify the sub-stream scheduling operations in existing hybrid systems:

$$\text{Max} \sum\nolimits_{h\in\Omega_i} \sum\nolimits_{j\in\Gamma_i} I^i_{hj} e^i_{hj}, \tag{1}$$

$$\text{s.t.} \sum\nolimits_{h\in\Omega_i} I^i_{hj} e^i_{hj} \leq 1, j \in \Gamma_i, \tag{2}$$

$$\sum\nolimits_{j\in\Gamma_i} I^i_{hj} e^i_{hj} \leq T_{ih}, h \in \Omega_i, \tag{3}$$

$$e^i_{hj} \in \{0, 1\}, h \in \Omega_i, j \in \Gamma_i. \tag{4}$$

Constraint (2) ensures that each sub-stream can be scheduled to one neighbor at most. In constraint (3), the token number for neighbor h of peer i, T_{ih}, is calculated by $T_{ih} = \left\lceil \frac{L_{hi}}{R/|I_i|} \right\rceil$ (we choose rounding up instead of rounding down so that peers' upload capacities can be fully utilized). The maximum number of sub-streams can be scheduled to neighbor h is limited by their end-to-end bandwidth. Constraint (4) ensures that this optimization problem is a binary integer programming problem. We will show that this model can be transformed into an equivalent *max-flow* problem, which is solvable in polynomial time [7].

Proposition 1. *Implementations of sub-stream scheduling in GridMedia, CoolStreaming+ and LStreaming are special cases of the proposed max-flow model.*

Proof. Due to the page limitation, we only demonstrate the proof for GridMedia. Other two systems can be similarly proved. Note that both constraints (2) and (4) are satisfied in all these systems obviously; therefore, the proof only focuses on constraint (3) and the objective function of the max-flow model. Without loss of generality, we focus on peer i in the proof. GridMedia does not consider token limit on sub-stream scheduling (i.e. $\forall h \in \Omega_i$, $T_{ih} = \chi$). This indicates that constraint (3) can be satisfied all the time. Peers in GridMedia group a fixed number of consecutive packets as a *packet group*. Moreover, peers cluster every g consecutive packet groups into a *packet party* with group number from 0 to $g - 1$. In GridMedia, whenever missing chunks in *packet group* 0 are requested by the chunk-pulling module, sub-stream scheduling is actually conducted at this moment. There can be multiple missing chunks in packet group 0 and each missing chunk belongs to one particular sub-stream. Suppose a packet-group-0 chunk in sub-stream j is missing, if neighbor h has this particular chunk (known from the buffer map), then $I^i_{hj} = 1$. This indicates that peer h is qualified for sub-stream j. Since there is no bandwidth limitation, peer i can always schedule sub-streams successfully. In other words, the number of sub-streams successfully scheduled is always maximized. Thus the objective function of the max-flow model holds for GridMedia. Therefore, the sub-stream scheduling scheme in GridMeida is a special case of the max-flow model.

2.3 Problem Transformation

In this subsection, we show that the proposed *sub-stream scheduling* model can be transformed into an equivalent classical *max-flow* problem, which can be solved in polynomial time. We briefly outline the max-flow problem here. Let a network $G = (V, E)$ be a directed graph in which each arc $(m, n) \in E$ has a nonnegative capacity $u(m, n) \geq 0$. V contains two special elements s and t, which represent *source* and *sink*, respectively. The decision variable, $f(m, n)$, denotes the amount of flow along (m, n). If $(m, n) \notin E$, both $u(m, n) = 0$ and

$f(m,n) = 0$. The max-flow problem can be written as follows:

$$\text{Max} \sum\nolimits_{n \in V} f(s,n), \tag{5}$$

$$\text{s.t.} \sum\nolimits_{n:(m,n) \in E} f(m,n) - \sum\nolimits_{n:(n,m) \in E} f(n,m) = 0, \forall m \in V - \{s,t\}, \tag{6}$$

$$0 \leq f(m,n) \leq u(m,n), \forall (m,n) \in E. \tag{7}$$

The time complexity of solving the max-flow problem is bounded by $O(VE^2)$ by adopting the *Edmonds-Karp algorithm* [7]. Due to the page limitation, the detail of the transformation algorithm is omitted here; nevertheless, we provide an illustration of the transformation shown in Fig. 4 for the example in Fig. 3. In Fig. 4, $\forall h$, N_{ih} indicates that peer h is a neighbor of peer i and $\forall j$, S_{ij} denotes that sub-stream j needs to be scheduled by peer i. The arc between each pair of N_{ih} and S_{ij} ensures that neighbor h is qualified for sub-stream j based on the *peer qualification rule*. If $I^i_{hj} = 1$, an arc exists from N_{ih} to S_{ij}. Nodes s and t in Fig. 4 are artificially introduced in order to set up the max-flow graph. The arc from s to each N_{ih} with capacity T_{ih} satisfies the token number limitation in our model (constrain (3)), and the arc from each S_{ij} to t with capacity 1 guarantees that each sub-stream can be scheduled to one neighbor at most.

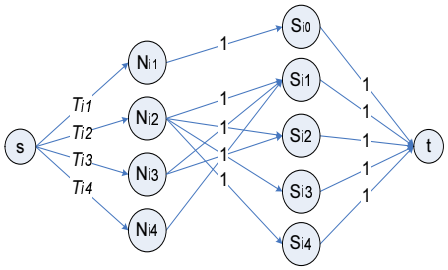

Fig. 4. Transformation to the max-flow problem at peer i

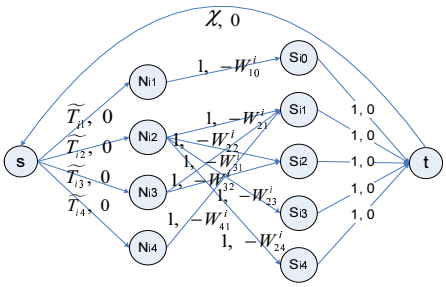

Fig. 5. Transformation to the min-cost flow problem at peer i

Theorem 1. *The unified sub-stream scheduling model at peer i is equivalent to a max-flow problem.*

Proof. Applying the *flow balance equations* to all internal nodes N_{ih}, $\forall h \in \Omega_i$, and S_{ij}, $\forall j \in \Gamma_i$, in Fig. 4, we have

$$f(s, N_{ih}) = \sum\nolimits_{j \in \Gamma_i} f(N_{ih}, S_{ij}), \forall h \in \Omega_i, \tag{8}$$

$$f(S_{ij}, t) = \sum\nolimits_{h \in \Omega_i} f(N_{ih}, S_{ij}), \forall j \in \Gamma_i. \tag{9}$$

Because each sub-stream can be scheduled to at most one neighbor, we have

$$f(N_{ih}, S_{ij}) = I^i_{hj} e^i_{hj}, \forall h \in \Omega_i, \forall j \in \Gamma_i. \tag{10}$$

Plugging Eq. (10) into Eq. (8) and Eq. (9) and applying capacity constraint on each arc, we have

$$f(s, N_{ih}) = \sum\nolimits_{j \in \Gamma_i} f(N_{ih}, S_{ij}) = \sum\nolimits_{j \in \Gamma_i} I^i_{hj} e^i_{hj} \leq T_{ih}, \forall h \in \Omega_i, \tag{11}$$

$$f(S_{ij}, t) = \sum\nolimits_{h \in \Omega_i} f(N_{ih}, S_{ij}) = \sum\nolimits_{h \in \Omega_i} I^i_{hj} e^i_{hj} \leq 1, \forall j \in \Gamma_i. \tag{12}$$

Eqs. (11) and (12) are the first two constraints of our *sub-stream scheduling* model. Since the capacities in Fig. 4 are all binary integers, the optimal flow on each arc is either 0 or 1 based on the *Integer Theorem* [7]. Therefore $f(N_{ih}, S_{ij}) = \{0 \textit{ or } 1\}$, which implies the constraint (4) of our original model is always satisfied.

The equivalence of the two objective functions is shown as follows:

$$\text{Max} \sum\nolimits_{h \in \Omega_i} f(s, N_{ih}) = \text{Max} \sum\nolimits_{h \in \Omega_i} \sum\nolimits_{j \in \Gamma_i} f(N_{ih}, S_{ij}) \tag{13}$$

$$= \text{Max} \sum\nolimits_{h \in \Omega_i} \sum\nolimits_{j \in \Gamma_i} I^i_{hj} e^i_{hj}. \tag{14}$$

The left-hand side of Eq. (13) is the objective function of the max-flow problem. Eq. (13) holds due to Eq. (8). If we apply Eq. (10) into the right-hand side of Eq. (13), we obtain Eq. (14), which is the objective function of our model. Thus, these two objective functions are equivalent. **Based on proposition 1 and theorem 1, the sub-stream scheduling in existing systems basically solves one particular max-flow problem.**

3 Min-cost flow scheduling

As shown in the proof of Proposition 1, there is no token limit imposed on GridMedia and CoolStreaming+. Without any consideration of load balancing, push-neighbor switching may incur as a direct consequence, which leads to increased redundant video download. The corresponding max-flow problems of all three existing systems usually have multiple optimal solutions. Since qualified neighbors are not differentiated in all three systems, they randomly select any optimal solution to schedule sub-streams. In practice, these qualified neighbors do not have the same network condition. In this section, we propose the *min-cost flow* scheduling scheme with the consideration of peer heterogeneity.

The tree-based streaming can achieve high throughput and low overhead when the system is stable. However, it is not able to cope well with a dynamic network. Since the throughput and overhead of the hybrid system highly depends on the sub-streaming scheduling component, a good hybrid system should adapt to peer churn quickly. LStreaming+ achieves this adaptability via the *min-cost flow* scheduling module. This module works periodically and has the ability to respond to network dynamics and schedule sub-streams to combat the network

fluctuation. Since sub-stream scheduling is triggered periodically in LStreaming+, each τ_i, $i = 1, 2, \ldots$, in Fig. 1 is set to be a constant τ. We assign a *weight* for each neighbor on sub-stream scheduling in LStreaming+. The weight is determined as follows:

- Each LStreaming+ peer examines the download performance from push-neighbors every τ time and adjusts the previous scheduling based on neighbors' download performance in the previous τ time. For example, let $S = 15$, $B = 30$ and $\tau = 10$. Ideally, $\tau \times B/S = 10 \times 30/15 = 20$ chunks should be downloaded for one sub-stream over τ time. We define a threshold to be half of this ideal value. If the number of the downloaded chunks in sub-stream j from neighbor h in the previous τ time is larger than this threshold, peer i sets the weight for this neighbor to be $W^i_{hj} = \chi$. This implies that peer i allows neighbor h continuing to push chunks in sub-stream j during the next τ time.
- For some sub-streams, if the chunk download threshold is not exceeded, they need to be scheduled to another neighbor in the next τ time. In the max-flow model, if one sub-stream of peer i can be downloaded from multiple neighbors, we do not differentiate among the qualified neighbors. However, in LStreaming+, if neighbor h has more fresh chunks in sub-stream j compared with another qualified neighbor, peer i assigns a larger weight for neighbor h. We denote the largest chunk number already available for sub-stream j at peer h (known from the exchanged information) as P^i_{hj} and the starting chunk number needed by peer i for sub-stream j as Q_{ij}, respectively. Since P^i_{hj} represents the chunk availability of sub-stream j from neighbor h, we set the weight as $W^i_{hj} = P^i_{hj} - Q_{ij}$ to differentiate the qualified neighbors. A larger $P^i_{hj} - Q_{ij}$ value shows that more available chunks for sub-stream j can be obtained from neighbor h.
- In summary, the weight (W^i_{hj}, $\forall h \in \Omega_i$ and $\forall j \in \Gamma_i$) for each neighbor of peer i is determined as follows:

$$W^i_{hj} = \begin{cases} \chi, & \text{if the chunk download threshold is exceeded,} \\ P^i_{hj} - Q_{ij}, & \text{otherwise.} \end{cases} \tag{15}$$

  We aim to maximize the total weights on sub-stream scheduling with the objective function $\text{Max} \sum_{h \in \Omega_i} \sum_{j \in \Gamma_i} W^i_{hj} I^i_{hj} e^i_{hj}$.

In practice, it is difficult to obtain the exact value of L_{hi} between two peers h and i, where L_{hi} is the end-to-end bandwidth from peer h to peer i. In LStreaming+, each peer estimates the token number for each neighbor and then conducts the sub-stream scheduling. Suppose peer i has $|\Omega_i|$ neighbors. Their achievable uploading bandwidth in the previous τ time (by counting the number of received chunks) to peer i are denoted as $v_1, \ldots, v_{|\Omega_i|}$. Since the total number of sub-streams to be scheduled is $|\Gamma_i|$, in LStreaming+ the token number for each neighbor is proportionally assigned based on its individual performance in the previous τ time. Let $\widetilde{T}_{ih}$ denote the estimated token number for neighbor h by

peer i, and it is calculated as follows:

$$\widetilde{T}_{ih} = \left\lceil \frac{v_h}{\sum_{k=1}^{|\Omega_i|} v_k} \times |\Gamma_i| \right\rceil, \forall h \in \Omega_i. \tag{16}$$

Then the sub-stream scheduling problem in LStreaming+ is formulated as follows:

$$\text{Max} \sum\nolimits_{h\in\Omega_i} \sum\nolimits_{j\in\Gamma_i} W^i_{hj} I^i_{hj} e^i_{hj}, \tag{17}$$

$$\text{s.t.} \sum\nolimits_{h\in\Omega_i} I^i_{hj} e^i_{hj} \leq 1, j \in \Gamma_i, \tag{18}$$

$$\sum\nolimits_{j\in\Gamma_i} I^i_{hj} e^i_{hj} \leq \widetilde{T}_{ih}, h \in \Omega_i, \tag{19}$$

$$e^i_{hj} \in \{0,1\}, h \in \Omega_i, j \in \Gamma_i. \tag{20}$$

This model has similar constraints as the max-flow model in Section 2. Nevertheless, the sub-stream scheduling in LStreaming+ needs to be transformed into an equivalent *min-cost flow* problem, which can be solved in polynomial time. Let a flow network $G = (V, E)$ be a directed graph in which each arc $(m, n) \in E$ is associated with a cost $c(m, n)$. The lower bound $l(m, n)$ and the upper bound $u(m, n)$ of each arc capacity denote the minimum and maximum amount of flows that can pass through the arc. The min-cost flow problem can be written as follows:

$$\text{Min} \sum\nolimits_{\forall(m,n)\in V} c(m,n) f(m,n), \tag{21}$$

$$\text{s.t.} \sum\nolimits_{n:(m,n)\in E} f(m,n) - \sum\nolimits_{n:(n,m)\in E} f(n,m) = b(m), \forall m \in V, \tag{22}$$

$$\sum\nolimits_{k=1}^{|V|} b(k) = 0, \tag{23}$$

$$l(m,n) \leq f(m,n) \leq u(m,n), \forall (m,n) \in E. \tag{24}$$

In our problem, we set $b(m) = 0$, $\forall m \in V$, and $l(m, n) = 0$, $\forall (m, n) \in E$. The time complexity for solving this classic min-cost flow problem is bounded by $O(|V||E|(\log(\log U))\log(|V|C))$ [8], where U and C are the largest values of the arc capacity and the arc cost, respectively.

Theorem 2. *The scheduling in LStreaming+ is equivalent to a min-cost flow problem.*

Due to page limitation, the proof and the transformation algorithm are omitted here. We use Fig. 5 to illustrate this transformation for the example in Fig. 3 intuitively. Each arc in Fig. 5 is characterized by two parameters: arc capacity and cost. We generate internal nodes N_{ih} and S_{ij}, arcs and arc capacities similar to the max-flow model. The major difference between this min-cost flow problem and the max-flow problem is on the arc cost along each arc. In the problem transformation, only the arc between N_{ih} and S_{ij} may have nonzero cost, because only these arcs are related to each decision variable e^i_{hj} in the objective function Eq. (17). Minimizing the total negative costs is equivalent to maximizing the total positive weights.

4 Performance Evaluation

In this section, we evaluate the accuracy of the proposed max-flow model and the streaming performance of LStreaming+ in comparison to GridMedia (GM), LStreaming (LS) and the generic random mesh-pull scheme (MP) using simulations. We developed a discrete-event simulator coded in C++ to simulate the system behavior at the chunk level. The simulator implements the *max-flow scheduling* module and LStreaming+. In LStreaming+, we utilize the "CS2" library [9] to solve min-cost flow problems. To evaluate the accuracy of our max-flow model for characterizing the sub-stream scheduling of existing systems, we replace the original sub-stream scheduling modules of GridMedia and LStreaming by using the max-flow scheduling module instead, and compare the performance of each modified system with its original system, respectively.

To achieve a realistic latency setup, the end-to-end latency between peers is randomly selected from the real-world node-to-node latency matrix (2500×2500) [10]. The playback rate of the stream is 300kbps and the default neighbor number is 15. All peers are DSL users with three types of upload capacities of 1Mbps, 512kbps and 128kbps, and with the corresponding download capacities of 3Mbps, 1.5Mbps and 768kbps. The fractions of these three types of peers are 10%, 50% and 40%, respectively. The upload capacity of the server is 900kbps. We simulate a 15-minute streaming session. There are 10000 peers joining the channel in total. During 5 time intervals of $[0, 50]$sec, $[200, 250]$sec, $[400, 450]$sec, $[600, 650]$sec and $[800, 850]$sec, 20% of total 10000 peers join the channel, respectively. The joining time of peers is uniformly distributed within each time interval. Among all peers, 5% of them do not leave after joining. The viewing durations for the remaining 95% peers are uniformly distributed from 190 seconds to 210 seconds. With the setting above, we want to simulate a dynamic network in which peer churn occurs during the entire simulation time.

4.1 Simulation results

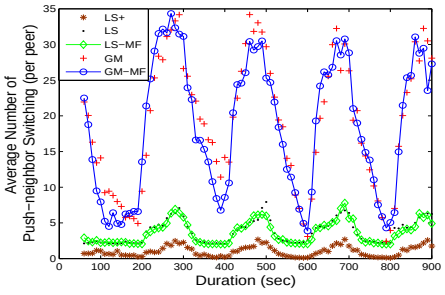

Fig. 6. Push-neighbor switching

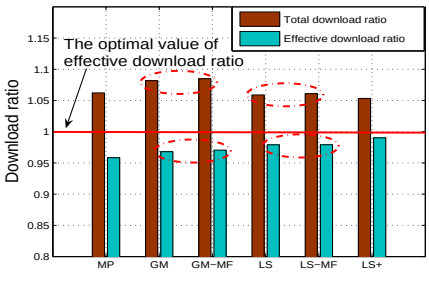

Fig. 7. Download performance

Push-neighbor switching We characterize push-neighbor switching using the average number of switching push-neighbors when peers conduct sub-stream scheduling. In our previous analysis, duplicate chunks are downloaded mainly due to push-neighbor switching. With the same simulation settings including networking topology, link latency and peer churn, we observe that the modified system with the max-flow scheduling module (GM-MF or LS-MF) performs almost the same as its original system (GM or LS). This shows that the *max-flow* model captures the behavior of sub-stream scheduling schemes in existing systems very well. In addition, Fig. 6 shows that LStreaming+ has the least push-neighbor switching, which implies that LStreaming+ incurs the smallest amount of video redundancy overhead among all systems.

Download performance We define two traffic ratios to characterize the download performance. The *total download ratio* is the ratio between the total download rate and the video playback rate. The *effective download ratio* is the ratio between non-duplicate video download rate and the video playback rate. The *effective download ratio* characterizes the useful video download of the system. In Fig. 7, we observe that the modified streaming systems and the original systems experience almost the same total download ratio and effective download ratio. In Fig. 7, the effective download ratios of all hybrid systems are higher than that of a generic random pull-based system (denoted as **m**esh-**p**ull (MP) in Fig. 7). In addition, the effective download ratio of LStreaming+ is closer to the optimal value 1 compared with GridMedia and LStreaming. LStreaming+ outperforms all other schemes in terms of the quality ratio.

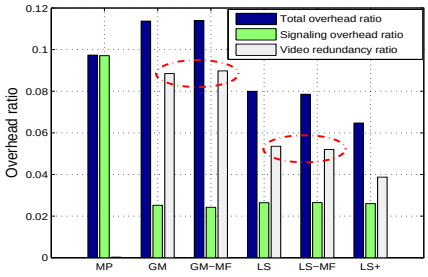

Fig. 8. Overhead breakdown

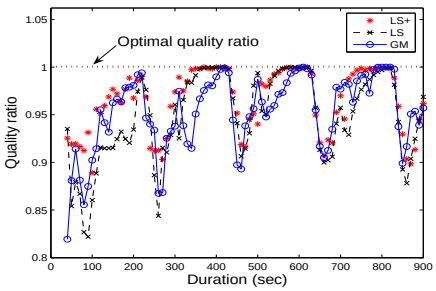

Fig. 9. Streaming quality

Overhead breakdown The total overhead traffic consists of signaling messages and redundant video chunks. In pull-push systems, the redundant video download is a potential problem. Video chunks usually have much larger size than signaling packets. If the number of duplicate video chunks is large, redundant video traffic becomes the major contributor in the total overhead traffic. In

Fig. 8, each pair of the modified system and its original system suffers from almost the same amount of video redundancy ratio, which provides another good indication that the max-flow model for the existing systems is very accurate. From Fig. 8, we observe that the total overhead ratios of mesh-pull, GridMedia and LStreaming reach 9.7%, 11.4% and 8.0%, respectively. Nevertheless, the total overhead of LStreaming+ is only 6.4%. Compared with mesh-pull, GridMedia and LStreaming, the overhead reduction of LStreaming+ is 34.0%, 43.9% and 20.0%, respectively. Through Fig. 8, it is clear that the video redundancy is the major contributor to the total traffic overhead in hybrid systems. GridMedia and LStreaming suffer from video redundancy as high as 8.9% out of the 11.4% total overhead and 5.4% out of the 8.0% total overhead, respectively. In contrast, LStreaming+ achieves a much smaller video redundancy ratio, only 3.8% out of the 6.4% total overhead.

Quality ratio We define the *quality ratio* as the ratio between the number of chunks, which has been played back, and the number of chunks which should be played back up to the current time. Fig. 9 shows the quality ratio of LStreaming+ outperforms the other two hybrid systems on average during the whole watching session. In particular, when peer churn occurs, LStreaming+ suffers the least and recovers the most quickly, compared with the other two systems.

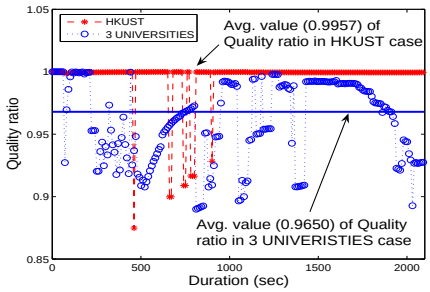

Fig. 10. Quality of the prototype

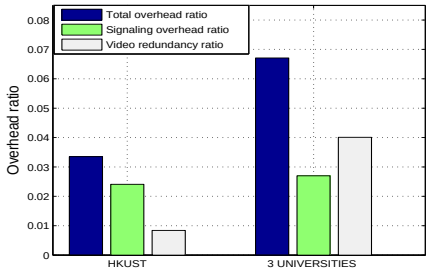

Fig. 11. Overhead of the prototype

4.2 Prototype experiments

Our experiments are first conducted on the HKUST campus network. Then, the experiments are conducted among three universities in Hong Kong, including HKUST, HKU and CUHK. In each experiment, 100 peers join a video channel with a 300kbps playback rate. At the application level, we manually limit the upload capacity of each peer with the same upload capacity distribution used in the simulation. As shown in Fig. 10, the *quality ratio* almost maintains at the optimal value 1 during the whole watching session on the campus network.

The *quality ratio* fluctuates in the inter-university experiments especially at the initial time. However, throughout the period of the experiments, there is only slight visual impact for a small number of peers. We also classify the traffic overhead in the prototype experiments in Fig. 11. Note that the total overhead of the campus experiments is as small as 3.35%. Even if in large-scale networks, the overhead ratio is merely 6.71%. Fig. 11 shows LStreaming+ effectively reduces the overhead and at the same time achieves a very good streaming performance.

5 Conclusion

In this paper, we propose a max-flow model for unifying the sub-stream scheduling problem in pull-push hybrid P2P streaming systems. We show that the sub-stream scheduling scheme in existing hybrid streaming systems including GridMedia, CoolStreaming+ and LStreaming, can be formulated into a special case of our proposed max-flow model. This max-flow model leads to useful insights for developing a min-cost flow model for scheduling sub-streams to better utilize heterogenous peers. We implement this min-cost flow model in a prototype system, LStreaming+. The accuracy of the max-flow model and the system performance of LStreaming+ are evaluated using extensive simulations. The prototype experiments also demonstrate that LStreaming+ achieves excellent streaming performance with significantly reduced traffic overhead.

References

1. Hei, X., Liang, C., Liang, J., Liu, Y., Ross, K.W.: A measurement study of a large-scale P2P IPTV system. IEEE Trans. on Multimedia **9**(8) (Dec. 2007) 1672–1687
2. Zhang, M., Zhang, Q., Sun, L., Yang, S.: Understanding the power of pull-based streaming protocol: Can we do better? IEEE JSAC **25**(10) (Dec. 2007) 1640–1654
3. Li, B., Xie, S., Keung, G., Liu, J., Stoica, I., Zhang, H., Zhang, X.: An empirical study of the Coolstreaming+ system. IEEE JSAC **25**(10) (Dec. 2007) 1627–1639
4. Li, Z., Yu, Y., Hei, X., Tsang, D.H.K.: Towards low-redundancy push-pull P2P live streaming. In: Proc. ICST QShine, Hong Kong (July 2008)
5. Li, Z., Yu, Y., Hei, X., Tsang, D.H.K.: Towards low-redundancy push-pull P2P live streaming. In: Proc. ACM SIGCOMM Demo, Seattle, USA (Aug. 2008)
6. Liu, Z., Shen, Y., Ross, K.W., Panwar, S.S., Wang, Y.: Substream trading: Towards an open P2P live streaming system. In: Proc. IEEE ICNP, Orlando, USA (Oct. 2008)
7. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithm. 2nd edn. The MIT Press (2001)
8. Hillier, F.S., Lieberman, G.J.: Introduction to Operations Research. McGraw-Hill (1995)
9. Goldberg, A.: Network optimization library `http://www.avglab.com/andrew/soft.html`.
10. Wong, B., Slivkins, A., Sirer, E.G.: Meridian: a lightweight network location service without virtual coordinates. In: Proc. ACM SIGCOMM. (Aug. 2005) 85–96