Computer Networks 91 (2015) 483–494

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Optimal bandwidth allocation for hybrid Video-on-Demand streaming with a distributed max flow algorithm

Chen Tian [a], Jingdong Sun [a], Weimin Wu [a,*], Yan Luo [b]

[a] School of Electronic Information and Communications, Huazhong University of Science and Technology, 1037 Luoyu Road, Wuhan, China
[b] Department of Electrical and Computer Engineering, University of Massachusetts Lowell, Lowell, MA, USA

ARTICLE INFO

Article history:
Received 29 October 2014
Revised 2 August 2015
Accepted 27 August 2015
Available online 11 September 2015

Keywords:
Video-on-Demand
Streaming
Bandwidth allocation
Max flow
Distributed algorithm

ABSTRACT

Video-on-Demand (VoD) streaming counts for the largest share of Internet traffic, and commonly relies on P2P–CDN hybrid systems. In such hybrid systems, a peer's upload bandwidth utilization is critical to P2P mode, thus the bandwidth allocation algorithm is important to the performance of VoD streaming. The current research either makes impractical assumptions, or is inefficient in real scenarios. Meanwhile some industrial practice such as additive-increase/multiplicative-decrease (AIMD) heuristic, although effective in practice, lacks theoretical foundation. This paper develops an optimal bandwidth allocation algorithm for hybrid VoD streaming. Specifically, we propose a novel *Demand Driven Max-Flow Formulation*, which treats each peer's bandwidth demand as the flow commodity. The proposed distributed *Free-for-All Push–Lift* algorithm can solve the formulation in each peer, and is free of any lock, shared memory and atomic operation. Following the theoretical analysis, we implement the algorithms in real-world VoD streaming systems. Through extensive evaluations we demonstrate that our approach can provide high-quality bandwidth allocation for hybrid VoD systems.

1. Introduction

Motivation Online streaming has been dominating the Internet traffic [1] and Video-on-Demand (VoD) is the primary method of content consumption [2,3]. Among VoD services, some services such as NetFlix and Hulu rely purely on Content Distribution Networks (CDNs) while others exploit Peer-to-Peer (P2P) transport to reduce service cost [4–6]. A growing trend is however leveraging P2P–CDN hybrid systems [5–10] for VoD services, where the contents requested by a user are delivered by either her peers or the CDN . Most modern P2P streaming networks are operated in such a hybrid mode. One representative example is the RTMFP-based P2P support of Adobe Flash platform [11]. In such hybrid platforms, it is beneficial to maximize the content distribution through peers rather than CDN due to CDN hosting costs.

The upload bandwidth utilization of peers is critical for VoD in P2P mode. Each peer has two roles: uploader and downloader. The more upload bandwidth a peer could exploit from its neighboring peers, the more data it could download from them. For hybrid systems, more downloaded data from peers translates to fewer video piece missing events in the P2P mode [12,13], hence reducing data serviced from CDN servers.

However, the existing approaches are ineffective or inadequate in optimizing the upload bandwidth utilization for peers. Many of the prior research either make impractical assumptions, or are inefficient in real-world scenarios. Some works such as [14,15] assume a complete connected graph, which is not realistic in real P2P systems. Zhao's work [16] showed that with a random peer selection and uniform bandwidth allocation, a system can asymptotically achieve a close to the optimal streaming rate if each peer

* Corresponding author. Tel.: +862787543236.
E-mail addresses: tianchen@hust.edu.cn (C. Tian), jdsun@hust.edu.cn (J. Sun), wuwm@hust.edu.cn, 565714801@qq.com (W. Wu), yan_luo@uml.edu (Y. Luo).

http://dx.doi.org/10.1016/j.comnet.2015.08.049
1389-1286/

maintains $O(\log N)$ downstream neighbors. For every practical system, there is an upper bound of the number of neighbors; there are also many scenarios where the uniform rate allocation certainly cannot work efficiently (see Section 3 for details). To the best of our knowledge, the common industry practice is based on the additive-increase/multiplicative-decrease (AIMD) heuristic. Motivated by TCP congestion control [17], AIMD is a simple allocation protocol in the P2P overlay layer [2,18]. Although proved effective in practice, AIMD lacks theoretical foundations thus is not extensible.

We are motivated to model the bandwidth allocation problem in hybrid VoD platforms and provide efficient solutions. Central to the upload bandwidth utilization is the bandwidth allocation algorithm, which decides in each peer how its bandwidth should be allocated to its neighboring peers. We aim to design such a bandwidth allocation algorithm that satisfies two important criteria. First of all, a good algorithm should not leave (much) unused bandwidth. Due to the pressure of market competition, the video streaming rates become higher and higher; usually in a video channel, peers' aggregated upload bandwidth is either almost equal to or less than the aggregated steaming bandwidth demand. It is beneficial to maximize the overall bandwidth utilization. Second, a good algorithm should allocate to each peer *just enough* bandwidth that can support the streaming rate. For a single peer, if the aggregated allocated bandwidth is less than the rate, it would download a significant portion of data from CDN, which needs to be prevented. The allocated bandwidth should not be (much) larger than the streaming rate either: each peer is consuming the video at the same rate; a peer exploiting unnecessary bandwidth is just accumulating its own buffered data. If the peer quits the channel early or skips the segment, which are very common [5], all downloaded data are wasted. In summary, unnecessary bandwidth in one peer would eventually reduce the download rates of others, which in turn deteriorates the performance of the whole system. In designing such an algorithm, we would like to ensure that the approach is guided by theoretical analysis, and implemented under practical constraints of real VoD systems.

Challenge In practice it is challenging to design an optimal bandwidth allocation algorithm as explained below.

The first challenge is *how to formulate the optimal objective.* To maximize peers' contribution, a max-flow formulation is the natural choice. The real question is: directly modeling allocated bandwidth between a pair of peers as flow commodity is intuitive, while not suited for the receiver-driven nature of streaming systems (see Section 4 for details).

The second challenge is *how to solve the optimal formulation.* Bandwidth allocation has to be solved in a distributed manner. Centralized scheduling is not applicable in P2P systems: the topology of peer connections is inherently dynamic, while a centralized control has to re-calculate the max-flow problem each time a peer joins or leaves the channel and then has to distribute the results back to every peer.

The third challenge is *how to convert the theoretical design to a practical implementation.* The real P2P scenarios have many requirements: *e.g.*, after agreement of bandwidth allocation between an uploader and a downloader, how to realize it in an uploader? What if the promised bandwidth is temporarily larger or lower than the actual bandwidth? How to tolerate topology dynamics?

Contribution We make the following contributions in this paper to address the challenges:

- We propose a *Demand Driven Max-Flow Formulation*, which treats each peer's bandwidth demand as flow commodity. This formulation is friendly to the receiver-driven nature of streaming systems (Section 4).
- We propose a distributed *Free-for-All Push–Lift* algorithm to solve the optimization formulation and overcomes real distributed system constraints, such as the lack of global locks, shared memory and atomic operations (Section 5).
- We transform the theoretical design into a practical implementation by meeting real system requirements, such as piece scheduling and topology dynamics (Section 6).
- We conduct extensive evaluations and demonstrate that the proposed approach can provide high quality bandwidth allocation for hybrid VoD systems. In every scenario, its performance is better than AIMD (Section 7).

The rest of paper is organized as follows. Section 2 discusses the related work. Section 3 gives the background of hybrid VoD systems and the bandwidth allocation problem. Section 8 concludes the paper.

2. Related work

There are two previous studies closely related to our research. Although Yu and Chen model bandwidth allocation in P2P systems as a max-flow problem, they use a *Supply-Driven* formulation [19]. As mentioned in Section 4, *Supply-Driven* is not well suited for distributed receiver-driven nature of P2P systems. Also, their work solves the formulation using a centralized multi-stage max-flow algorithm. We have pointed out that centralized scheduling is not applicable in P2P systems, given that the topology is highly dynamic. Therefore, Yu's work of applying centralized computation on a static graph is impractical for hybrid streaming networks. He et al. present a systematic study on the throughput maximization problem in P2P–VoD applications [20]. A fully distributed algorithm is proposed for both buffer–forwarding and hybrid–forwarding architectures. The basis of their algorithm is linear programming with Lagrangian duality. However, any changes in peer's upload or download bandwidth requires a new programming execution on the whole graph. It is also thus impractical for highly dynamic P2P systems.

There are research studies on bandwidth allocation for multi-tree based overlay networks [21,22], which are related to P2P based VoD streaming. Our paper focuses on mesh-based hybrid VoD networks without content bottlenecks. To our knowledge, this is the first paper that presents a proved distributed algorithm, and implements a practical design for such systems. Bradai et al. focus on bandwidth allocation and the incentive mechanisms for layered video streaming [23]; our paper instead focuses on the widely supported non-layered VoD systems.

Some research papers also involves bandwidth allocation but in the scope of live streaming. Wu et al. propose bandwidth allocation algorithms to properly provision bandwidth among multiple live streaming channels [24]. ShadowStream project directly uses the industry AIMD approach [12,13]. One of our previous work also deals with topology dynamics such

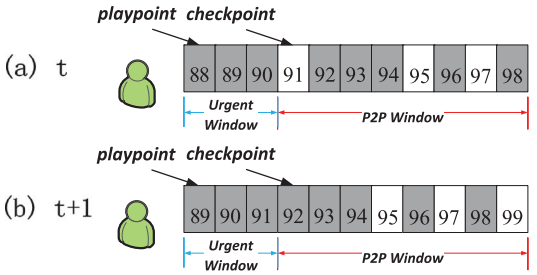

Fig. 1. Video buffer at time: (a) t; (b) t + 1.

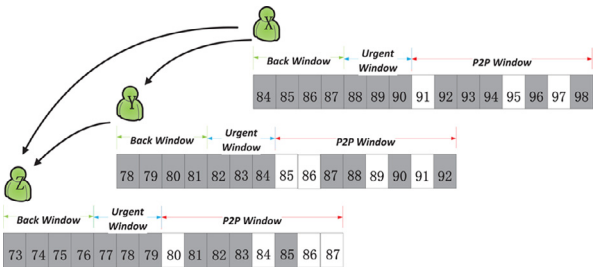

Fig. 2. Neighboring relationship.

as seek operation and peer nodes join/departure [3]. All these works are orthogonal to the research in this paper.

Most modern hybrid P2P streaming systems have their root in BitTorrent-like P2P systems [7,8]. Tit-for-Tat is used to guide bandwidth allocation in BitTorrent [25]. Qiu and Srikant develop a fluid model for BitTorrent-like content-distribution systems [26]. Cl³evenot-Perronninet al. further improve it to a multiclass fluid model, which models both heterogeneous peer accesses and multiple differentiated service classes [27]. Bandwidth allocation in these BitTorrent-like systems aims for the completion of download the whole content. Instead, bandwidth allocation in streaming focuses on maintain a downloading rate at least equal to the streaming rate.

3. Bandwidth allocation in hybrid VoD systems

3.1. Basics of hybrid VoD streaming

A hybrid streaming network is typically BitTorrent-like. Video data are encoded to small clips (*i.e.* , pieces), and for simplicity of explanation, we assume that each piece contains 1 s of video. Guided by a tracker, peers viewing the same video form an overlay network to relay video pieces through the overlay topology. A peer in the channel exchanges piece bitmaps with its neighbors so that the neighbors can request video pieces from it. A peer could download pieces from CDN if it cannot download them from any of its neighbors.

The key data structure maintained by a peer is its video buffer, which keeps track of pieces to feed a media player. Fig. 1(a) is an example illustrating the status of a video buffer at time t. A shaded piece is one that has been downloaded. We follow a convention that the index of a piece is the time of that piece in the video.

There are two windows: P2P window and urgent window. The left most piece of the P2P window is called *checkpoint*; the left most piece of urgent window, which is also the next piece to be delivered to the media player, is called *playpoint*. For the example in Fig. 1(a), the checkpoint is 91 and the playpoint is 88.

The windows, checkpoint and playpoint advance synchronously in time domain. Fig. 1(b) shows the video buffer at the next time t + 1. If the peer is not able to download piece 91 in P2P window before the checkpoint, we say that piece 91 is missing. Subsequently, piece 91 moves into the urgent window, and the peer would have to download it from the CDN.

3.2. Bottlenecks analysis

There are two types of bottlenecks that could affect the bandwidth allocation between a pair of P2P peers: link bottleneck and content bottleneck. Link bottleneck is the path capacity from an uploader to a downloader. Due to the asymmetric nature of current access networks [28], usually the upload bandwidth dominates the link bottleneck. Content bottleneck is critical in live streaming: an uploader cannot supply a bandwidth to a downloader that is larger than its received bandwidth [29], due to the synchronous content consumption nature.

However, content bottleneck can be removed in VoD by smart neighboring. The content consumption in VoD is asynchronous. Besides P2P window and urgent window, the video buffer also keeps some pieces older than the playpoint in a back buffer, so that the peer can serve other peers that lag behind in the viewing progress. As shown in Fig. 2, by neighboring peers that were close in playpoint positions, and assigning peers with faster playpoints as the uploaders to slower peers, the system can completely remove the content bottleneck.

3.3. Bandwidth allocation

Bandwidth allocation is important to peers's upload bandwidth utilization. Shown in Fig. 3(a) is an example topology. We assume that peer *A*, *B* and *C* each has an upload bandwidth of 20 while *B*, *C* and *D* each has a download demand of 20 (assuming *A* has finished viewing the video). The optimal bandwidth allocation is shown in Fig. 3(b), where no additional CDN capacity is needed. As mentioned above, the state-of-art research is inefficient in many scenarios. For example, uniform bandwidth allocation [16] would allocate only half of the required bandwidth to *D*, as shown in Fig. 3(c).

As an industry practise, AIMD could utilize available bandwidth efficiently, but not optimally. The reason is that AIMD heuristically makes decisions locally; it is hard to converge due to the lack of cooperation among peers. More specifically, even if all peers are already in optimal allocation, the heuristic AIMD would unnecessarily probe again. For example, suppose all peers already converge to the optimal state of Fig. 3(a); in the next step, shown in Fig. 3(d), AIMD would unnecessarily increase the bandwidth by m/n in links $(A \to B)/(B \to C)$; as a consequence, the bandwidth in links $(A \to C)/(B \to D)$ are reduced too; the whole system goes back to a suboptimal state and the performance is degraded.

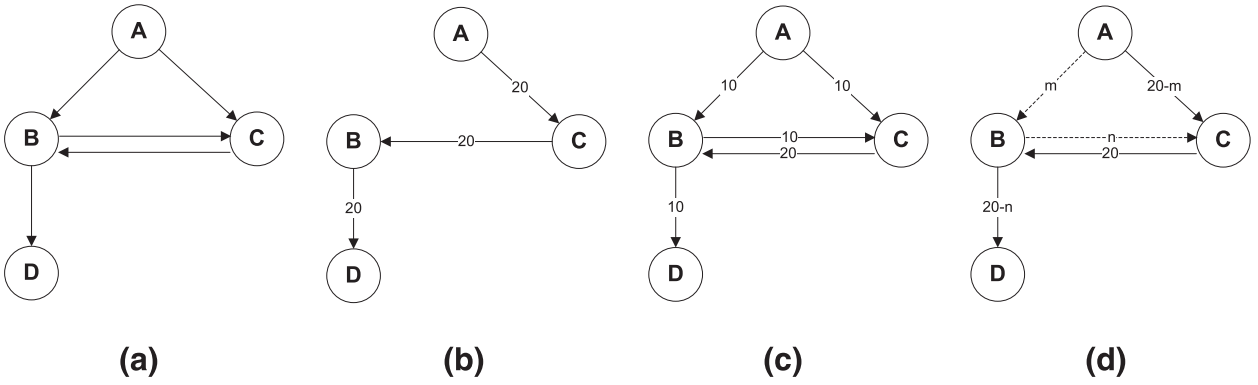

Fig. 3. Bandwidth allocation example (a) topology (b) optimal (c) uniform (d) AIMD.

Low Extra Delay Background Transport (LEDBAT) [30] is a delay-based congestion control algorithm that can exploit available bandwidth while limit the increase in delay. It is used by Apple for software updates and by BitTorrent for transfers. However, when compete with TCP, it backs off earlier since it is delay-sensitive while traditional TCP is loss-sensitive. Also, LEDBAT could invite an arm race among P2P software companies, by setting different target delay values [31]. To our best knowledge, many P2P streaming companies use AIMD for congestion control, such as PPLIVE [5].

3.4. Multi-rate support

Multi-rate support is necessary to deal with the inherent heterogeneity among streaming users. For example, a user inside an ADSL network usually has much lower uplink and downlink bandwidth compared with another user in an enterprise network.

There are two common multi-rate layered coding approaches: Scalable Video Coding (SVC) [32] and Multiple Description Coding (MDC) [33]. SVC generates from source one base video layer and several enhancement layers; the base layer is required for decoding in all receivers; the enhancement layers are optional and can be received to improve video quality. As a comparison, MDC generates multiple substreams, and each substream can be independently decoded; the more substreams received, the better the video quality. Multi-rate coding schemes all have coding overhead: compared with single-rate coding for the same quality, SVC has an overhead of around 10%, and MDC is around 30% [32,34–36]. As a result, they are not widely accepted by the industry yet due to increased consumption of bandwidth.

Another multi-rate approach is Partitioning, which is non-layered coding and widely used for dynamic streaming [37]. A single source is encoded to several versions with different qualities and rates; specific receivers are partitioned to different groups according to their received versions; only receivers in the same group can help each other, as content are different among groups. Apparently, Partitioning has no coding overhead. A receiver could switch to another version; the process is equivalent to leave one P2P swarm and join another. As mentioned above, in this paper we focus on the non-layered Partitioning scenario; in such a multi-rate context, our algorithm performs allocation in a single receiver group. We leave the research in the layered coding scenarios to future work.

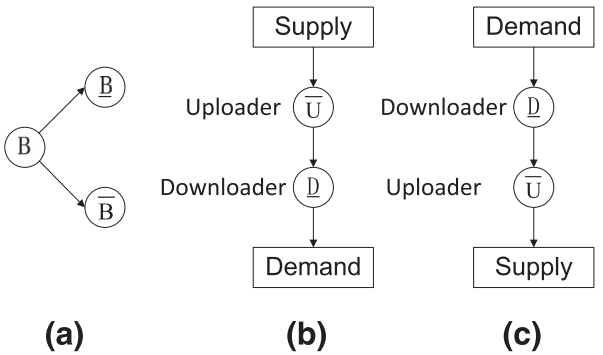

Fig. 4. Formulation (a) roles (b) Supply-Driven (c) Demand-Driven.

4. Demand-Driven formulation

We formulate the bandwidth allocation problem in this section. Shown in Fig. 4(a), two roles of a peer are modeled as separate nodes: the peer name with an underline indicates the downloader role and the name with an overline indicates the uploader role. Such abstraction allows us to precisely model the operations of a peer.

To maximize peers contribution, a max-flow formulation is the natural choice. A *flow network* is a directed graph $G = (V, E)$ with $|V| = n$ nodes and $|E| = m$ edges; s and t are the source and sink, respectively. For each edge (u, v), we define a non-negative capacity value $c(u, v)$. The notation f, called a *flow*, on G is a function on the edges that satisfies three constraints: (1) *capacity constraint*, the flow along each edge does not exceed the capacity of this edge; (2) *conservation constraint*, at each node u except the source and the sink, the *excess flow* $e(u)$, defined as the difference between the incoming and outgoing flows, is equal to zero; (3) *anti-symmetry constraint*, $f(u, v) = -f(v, u)$. We also define $c_f(u, v)$ as the residual capacity where $c(u, v) - f(u, v)$, and the residual graph $G_f(V, E_f)$, where $E_f = \{(u, v)|u \in V, v \in V, c_f(u, v) > 0\}$. The goal of max-flow is to maximize the value of the *flow*, which is the net flow out of the source $\sum_V f(s, v)$.

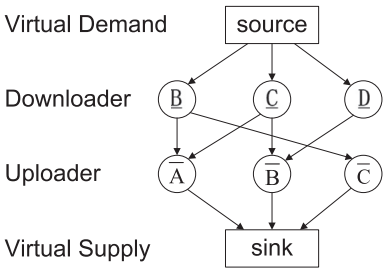

Fig. 5. *Demand-Driven* formulation of the example.

We could model flow commodity as either the allocated bandwidth from an uploader points to a downloader (*Supply-Driven* in Fig. 4(b)), or the requested bandwidth from a downloader points to an uploader (*Demand-Driven* in Fig. 4(c)). Although mathematically they are symmetric formulations, the choice of the flow commodity model has to take into consideration the realistic application behavior.

A key observation is that in a real-world streaming system, a peer needs to dynamically control its requested bandwidth. The requested bandwidth usually is closely related to the buffer vacancy ratio: it should be *just enough* to support the streaming rate when the buffer is almost full, and it should be larger when the buffer is close to empty. That is, streaming systems have a receiver-driven nature.

A *Supply-Driven* model is not well suited for distributed receiver-driven application scenarios like in VoD systems. When the demand increases in a downloader, it has to poll every uploader to check whether there is excess flow to push, and wait for each uploader's reply. As a comparison, the downloader could directly start the next round operation in a *Demand-Driven* formulation. Therefore, we choose to use *Demand-Driven* model in our analysis and optimization.

Shown in Fig. 5 is the *Demand-Driven* formulation of the Fig. 3(a) topology. As mentioned above, we could omit all other bottlenecks along the path except the peer uplink. There are constraints on the vertices and edges in the downloader–uploader topology as listed below:

- A virtual source node has outbound edges to every downloader; the capacity of each edge is equal to the bandwidth demand of each peer.
- Each downloader vertex has outbound edges to uploader vertices according to the neighboring relationship. The capacity of each edge is equal to the downloader demand.
- Every uploader vertex has an outbound edge to the virtual sink node, each of which has a capacity limit equal to the uploader bandwidth.

5. Distributed algorithms for the max-flow problem

5.1. Choices

There are two main types of solutions to the max-flow problem: augmenting path [38] and push–relabel [39]. The augmenting path method is more suitable for sequential implementation and proved to be inefficient by industry practice. Therefore, we develop our solution to be of push–relabel type.

A push–relabel algorithm relaxes the capacity constraint in the execution procedure. It maintains a *pre-flow* and a *height* label in each vertex, and uses *push* and *relabel* operations repeatedly to update the *pre-flow* and *height* until a maximum flow is found. For any vertex u who has excess flow, either *push* or *relabel* operation needs to be applied. The *push* operation sends flow from higher vertices to the lower ones. Suppose $h(u)$ is the current height of u, then *push(u,v)* is applicable when $h(u) = h(v) + 1$ and $(u, v) \in E_f$. When u has excess flow but $h(u) \leq h(v)$ where $(u, v) \in E_f$, *relabel* operation is applied to set $h(u) = min\{h(v) + 1 | (u, v) \in E_f\}$.

Most of the existing parallel push–relabel implementations rely on locks to protect every push and relabel operation [40,41]. For example, Anderson and Setubal's *global relabeling* periodically updates the distance labels to be the exact distance to the sink [41]. Bader's *gap relabeling* uses heuristics with considerations of the cache performance [40]. Although locks can protect every push and relabel operation, they would degrade the performance in large scale parallel scenarios such as our hybrid streaming networks.

Hong's *lock-free multi-threaded* variant serves the basis of our design [42]. The major modifications to the original *push–relabel* are: (1) flow is *pushed* to the lowest neighbor, instead of to a neighbor whose height is lower by just 1, and (2) *relabel* operation increases the node's height by 1 based on its lowest neighbor's height. Figuratively speaking, the *relabel* operation can be renamed as *lift*.

Both *push* and *lift* operations can be divided into two stages: *preparation* (check whether an operation is applicable) and *fulfill* (execute the operation). For vertex u, the implementation defines $h(u)$, $e(u)$, $c_f(u, v)$ as the global information stored in shared memory. The atomic operation and shared memory guarantee that global information would be loaded into per-thread private variables before any stage is executed, so that the local information in each thread can be up to date.

Since there are no locks, the two stages could be interleaved, thus two operations can form either a *stage-clean trace*, which means two operations' stages do not have any overlapping, or a *stage-stepping trace*, which means two operations' *prepare stages* are all executed before their *fulfill stages*. Through the discussion on all the interleaved situations, Hong proved that multiple *push* or *lift* operations can be equivalent to a sequence of non-overlapping traces, each of which can be defined as either a *stage-clean trace* or a *stage-stepping trace*. He further proved the correctness of his method by discussing with the two traces.

To distinguish it from other *push–relabel* methods, we call Hong's approach as *push–lift* method. This method has been deployed on a CUDA-based CPU–GPU hybrid platform [43].

The following theorem and lemmas have been proved by previous work [38].

Theorem 1. A *flow f* is maximum *if* and *only if* there is no *augmenting path*; that is, t is not reachable from s in G_f.

Lemma 1. *For any $u \in V$, $h(u)$ never decreases during the execution of the algorithm.*

Lemma 2. *During the execution of the algorithm, for any vertex that has excess flow, either a relabel or a push operation can be applied.*

Table 1
Messages and events: U⇒D is from *uploader* to *downloader*; D⇒U vice versa.

Message/Event	Category	Direction	Description
MSG_PUSH	NRC	U⇒D/D⇒U	One node pushes flow to its lowest neighbor
MSG_RELABEL	NRC	U⇒D/D⇒U	One node relabels its height and notify to its neighbors
MSG_PIECE_REQ	DDC	D⇒U	Downloader requests pieces from uploader, waiting for response
MSG_PIECE_REPLY	DDC	U⇒D	Uploader replies piece request and pushes pieces to downloader
MSG_PIECE_REJ	DDC	U⇒D	Uploader rejects piece request from downloader, no piece delivered
MSG_PEER_CONN	PCC	D⇒U	Downloader connects to a new neighbor retrieved from the tracker
MSG_PEER_DISCONN	PCC	U⇒D/D⇒U	One node sends disconnection signal to its neighbors
EVT_TRACKER_COMM	PCC	U/D periodically	Each node communicates with tracker and exchanges information
EVT_REFRESH_NEIGHBOR	PCC	U/D periodically	Node check its neighbors and exchange information periodically
EVT_PIECE_SCHEDULE	DDC	D periodically	Downloader schedules target pieces into piece request queue
EVT_BANDWIDTH_UPDATE	NRC	U/D periodically	Updates the supply or demand of peer

However, *push-lift* approach still requires shared memory and atomic operations supported from the system. In distributed P2P-VoD systems, there is neither atomic operation nor shared-memory. Information about bandwidth allocation is recorded separately in each peer. Partial information stored in trackers is outdated soon due to the communication delays and dynamics on content consumption. The challenge is to develop a practical design, which overcomes the constraints of real-world distributed system by exploring the distributed characteristics of peers.

5.2. Achieving atomic-operation-free and shared-memory-free

We propose a *Free-for-All Push–Lift* algorithm which eliminates any centralized mechanisms, such as shared memory and atomic operations. In the following we use node and vertex interchangeably to describe the vertices in a graph G.

We have defined $h(a)$ as the height of node a. Due to information dissemination delays, the height of the nodes may be inconsistent, thus $h_a(b)$ is defined as the height of b is recorded in node a. Note that $h_a(b)$ may be outdated information. We apply *push* and *lift* operations as follows: *push(a,b)* applies when $h(a) > h_a(b)$; *lift(a)* applies and sets $h(a) = min\{h_a(neighbors)\} + 1$. The problem arises to how to apply *push* or *lift* operations because of the uncertain and dynamic relationship between $h(a)$ and $h(b)$.

A new operation *error-push* is added to specifically handle inconsistencies due to communication delays. Assume *push(a, b)* is applied by a, but when the message arrives, b finds that $h(a) \leq h(b)$ (which is $h_a(b) < h(a) \leq h(b)$). Then b will return all the erroneously pushed flow back to a immediately with an *error-push* operation. Note that *error-push(b, a)* acts like *push(b, a)* but without *preparation* stage, where the flow value is the same to the previous *push(a, b)* operation, and carries the current height $h(b)$.

The *uploader* nodes initialize the height of virtual supply node $h(t) = |V| = n$. The *downloader* nodes initialize the height of the virtual demand node $h(s) = 0$, and execute *preflow* during initialization. After the initialization, each node will enter a main loop to periodically check the excess flow and receive messages from other nodes. For any node who has excess flow, either a *push* or *lift* operation is applied. We set that every operation carries the sender's current height, and height update $h_a(b) \leftarrow h(b)$ will be applied *if* and *only if* node a receives a message that carries a higher height $h(b)$.

The proposed *Free-for-All Push–Lift* algorithm is a distributed algorithm that is atomic-operation-free and shared-memory-free. For succinct presentation, we provide the proof in Appendix A.

6. Theory to implementation

We derive a practical implementation that is built on our theoretical model and proposed new algorithmic operations in Section 5. In this section, we first describe the implementation framework, then present the detailed designs.

6.1. Framework

The proposed framework includes properties of a peer and communication patterns among peers. Peers data structures in the framework record peer states while the communications among peers are described in messages and events that may alter the states of peers. We start by introducing the three major components of a peer:

- *Control block*: It maintains an ID to uniquely identify the peer in the network, a state flag, a message queue for receiving and processing messages, and an event queue for events listening and responding.
- *Uploader*: A uploader component has two elements. One is the node information, including node's height, excess flow and end-node flow (connection from virtual *supply node*). The other is the information of its neighbors, including neighbor's height and pushed flow.
- *Downloader*: The data structure is similar to that of *uploader*, except that the end-node flow indicates the connection to virtual *demand node*.

Peer communication in the framework relies on messages and events. We introduce three sets of information exchanging functions that triggered by messages/events: (1) NRC (network resource control), bandwidth scheduling among streaming nodes; (2) DDC (data distribution control), sending and receiving video pieces; (3) PCC (peer connection control), connecting or disconnecting neighbor peers. The callback function corresponding to each message or event belongs to either one of the three key sets. Table 1 lists the detailed function descriptions and classification of messages and events.

6.2. Pieces scheduling

Before a *downloader* could request pieces from an *uploader*, the *bandwidth allocation* stage ascertains the bandwidth that a node would share to each of its neighbors. Functions of *bandwidth allocation* stage belong to the NRC set. Functions of *pieces scheduling* stage belong to the DDC set.

An *uploader* maintains a piece queue for each *downloader*; a weighted round robin (WRR) scheduling approach is adopted based on the promised bandwidth. Suppose u is an *uploader* for streaming resources with supply capacity $supply(u)$, and node $v_1, \ldots, v_n$ are u's *downloader* neighbors, $flow(v_i)$ where $i = 1, \ldots, n$ is the flow information recorded in the neighbors' list, $e(u)$ is the excess flow of node u. The scheduling works as follows: If $e(u)$ is zero, all demand from *downloader* neighbors can be satisfied, and u will allocate its bandwidth based on the flow information $flow(v_i)$. Otherwise (which implies $e(u) > 0$), the total demand from *downloaders* exceeds u's supply capacity, thus u will allocate its bandwidth based on the flow percentage that each *downloader* neighbor occupies, which is $supply(u) * flow(v_i) / \sum_1^n flow(v_j)$.

6.3. Handling peer dynamics

Peer dynamics comes from both topology and peer states: peers could come and leave, a single peer could adjust its upload bandwidth or download bandwidth. Functions of peer connectivity belong to the PCC set. Connecting execution includes connection signal, setup of peer connection, initial information exchange of height and initialization of flow. Disconnecting execution includes disconnection signaling, clean-up of peer connection and retrieval of flow.

Streaming peer may use EVT_BANDWIDTH_UPDATE to update supplies and demands, and this call-back function belongs to the NRC set. During the execution of *bandwidth allocation* stage, we can include the changing demand and supply bandwidth through simple flow pushing or modifying the end-node flow. For instance, the increasing of supply bandwidth can be achieved by increasing the *uploader* node's end-node flow.

6.4. Global relabeling heuristic

The global relabeling heuristic could accelerate the convergence, by periodically relabeling the accurate height of a node to the sink node in the residual graph [39]. Our approach could also benefit from such a strategy. When more than one neighbor has the same lowest height, the active node could push flow to the neighbor that has the lowest *global distance*. However, it requires a central controller to run a breadth first search, and this is impractical for distributed systems such as the hybrid VoD streaming.

We introduce a completely distributed global relabeling scheme based on the observation that our formulated graph for any topology all have just 4 levels, which means the distance between the source and sink is always 3. During the execution, the flows will be pushed back and forth between the *uploader* level and *downloader* level. This simple and natural graph structure allows peers to easily record its global distance to the sink node.

The distributed global relabeling scheme is as follows: each node has a list that records the information of its neighbors. We add one *global distance* variable G_i for each neighbor, which is expected to record the neighbor's actual distance to the sink node in the residual graph. Since the graph direction is always from the *downloader* to the *uploader*, the exchange logic of *global distance* is different. Suppose n is the number of neighbors, the *global distance* of *downloader* node d is

$$G_d = min\{G_1, G_2, \ldots, G_n\} + 1$$

and the *global distance* of *uploader* node u is

$$G_u = \begin{cases} 1, & c_f(u,t) > 0 \\ min\{G_1, G_2, \ldots, G_n\} + 1, & c_f(u,t) = 0 \end{cases}$$

6.5. Complexity analysis

The time complexity of our algorithm is $O(V^2E)$. In the worst case, the total number of lift operations is $(2 \mid V \mid -1)(\mid V \mid -2)$, which is bounded by $O(V^2)$, because the height of each node can never decreases. The number of saturating push operations is bounded by $O(VE)$. The number of non-saturating push operations is bounded by $O(V^2E)$ to make sure there is no excessive flow at termination. So the overall time complexity will be bounded by $O(V^2E)$. The global relabeling heuristic won't change the time complexity, but will accelerate the convergence speed by providing accurate height information of neighbor nodes using our distributed algorithm, thus the push operations can be reduced in a practice way.

AIMD does not have a fixed value of time complexity because it has no stable termination condition. The efficiency of AIMD can be affected by total bandwidth W, maximum segment size MSS, network topology and initial bandwidth allocation of each node. To the best of our knowledge, Lahanas and Tsaoussidis perform analysis for a two-flow AIMD case. Let β be the backoff parameter, the number of steps converge to fairness is $O(Wlog_\beta W)$ [44].

7. Evaluation

7.1. Methodology

In this section, we evaluate the performance of our proposed approach in a hybrid VoD system. A dedicated event-driven emulator is developed based on our previous work [12]. We focus on one key performance metric: *piece missing ratio*, which is the fraction of pieces that not received before the P2P mode deadlines. A major factor affecting the *piece missing ratio* is the *supply ratio*, which is the ratio between the total uploader bandwidth capacity and total streaming bandwidth demand of all participating peers.

To accurately emulate a real-world VoD system, we use the following default settings: (1) peers are randomly assigned different upload bandwidth; (2) peers can connect to new neighbors when their buffers are insufficient for video playback; (3) a tracker is responsible for assigning new

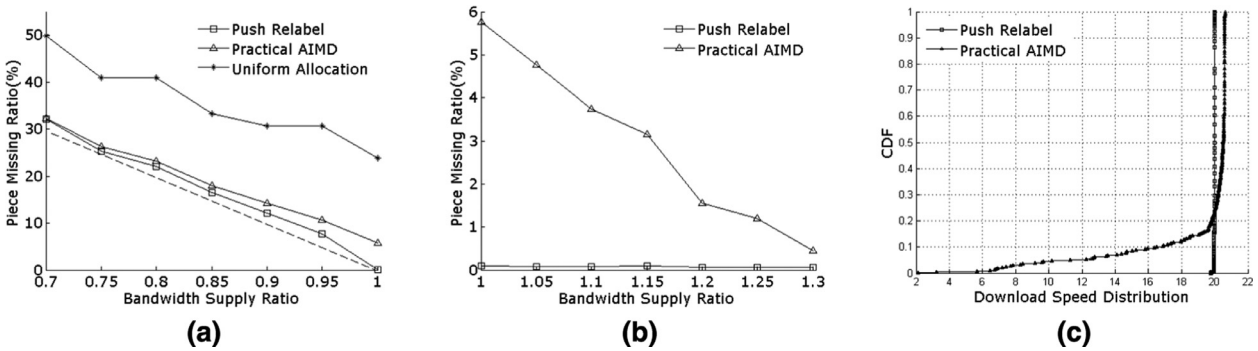

Fig. 6. Performance comparison with different supply ratios.

neighbors with surplus upload bandwidth to the demanding peer; (4) the peer buffer is set to 30 s; and (5) a peer's bandwidth demand is dynamically adjusted based on the vacancy ratio of its buffer.

Three algorithms are evaluated: the industry practice (*AIMD*), the uniform rate allocation algorithm (*Uniform Allocation*), and our distributed push–relabel approach (*Push–Relabel*). Each scenario runs 10 simulations repeatedly with different random seeds. Each simulation emulates 1000 s of a channel. In all scenarios, peers join into swarm pool randomly within 1 s.

7.2. Bandwidth supplies

In this part, we evaluate three approaches under different *supply ratio*. There are 400 peers that gradually join the channel. Each peer may have 6–10 download neighbors and the same number of upload neighbors. We evaluate *supply ratio* from 0.7 to 1.3 in a 0.05 step.

Insufficient Bandwidth Supply: The result is shown in Fig. 6(a). The performance of *Uniform Allocation* is worse than the other two approaches, consistent with our expectation. The performance of *AIMD* and *Push–Relabel* are close to each other when *supply ratio* is very small (*e.g.*, 0.7) while the performance of *Push–Relabel* achieves slightly better performance. We conjecture that *Push–Relabel* cannot exercise its full strength under such constrained scenario. However, with the increase of supply, the performance gap between these two approaches also increases: when supply ratio is 1, *Push–Relabel* has a *piece missing ratio* close to 0 (0.097%) while AIMD is 5.767%. The experiment log data confirm our expectation: as a heuristic, it is hard for AIMD to judge when to stop probe. As a result, the allocation by AIMD would never be stabilized hence the performance is affected.

Sufficient Bandwidth Supply: The results are shown in Fig. 6(b). With the further increase of bandwidth supply, the performance gap between *AIMD* and *Push–Relabel* shrinks again. When supply ratio increases to 1.3, the AIMD could also achieve a *piece missing ratio* less than 1%. The evaluation results show clearly that *Push–Relabel* performs always better than *AIMD*.

We also present the cumulative distribution function (CDF) of download bandwidth of peers in Fig. 6(c). The setting of *supply ratio* is 1.05. It is clear that the distribution of AIMD is uneven. As a comparison, *Push–Relabel* allocates just enough bandwidth to almost every peer. This is the main reason that AIMD has higher *piece missing ratio* than our Push–Relabel approach.

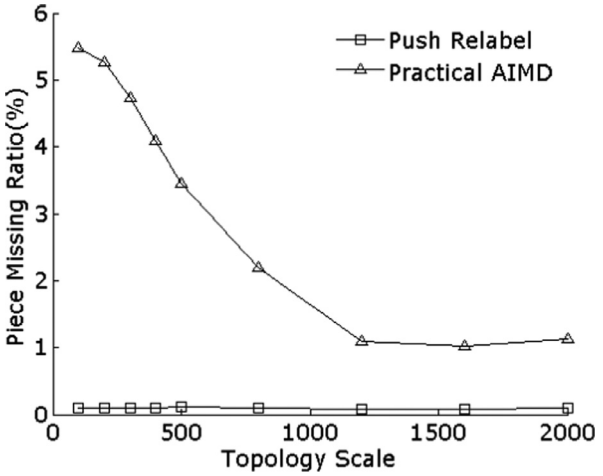

Fig. 7. Performance comparison with different scales.

7.3. Topology scaling

Next, we evaluate the performance of *AIMD* and *Push–Relabel* under different scales of network size. The scales under study include [100–500, 800, 1200, 1600, 2000] peers. We fixed *bandwidth supply ratio* to 1.05. The evaluation results are shown in Fig. 7. An interesting observation is that with the increase of network scale, the performance of *AIMD* also improves. This is consistent with industry practice we learned: the more peers, the more alternative choices for new neighbors when AIMD failed to meet the current bandwidth demand. While AIMD improves its allocation, the performance of *Push–Relabel* is always better than AIMD.

7.4. Neighbor connectivity

We then study how the connectivity of neighbors affects the allocation algorithms. As we vary the average degree of peers, shown in Fig. 8, the performance gap between *AIMD* and *Push–Relabel* shrinks with the increase of the average

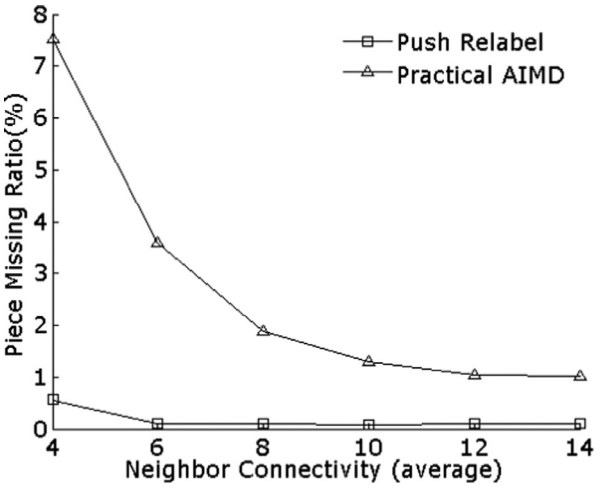

Fig. 8. Performance comparison with different numbers of neighbors.

degree. With more neighbors, AIMD is more robust to peer dynamics, hence its performance improves. However, in reality many systems have degree constraints: *e.g.*, our practice suggests that RTMFP-based Adobe Flash P2P platform degrades when degree per peer exceeds 6. Therefore, the proposed *Push–Relabel* has significant advantages over *AIMD* as *Push–Relabel* performs well independent on peer connectivity.

8. Conclusion

Video-on-Demand streaming has been dominating the Internet traffic for a while. P2P–CDN hybrid approaches are popular for modern VoD systems. Peers' upload bandwidth utilization is critical: more downloaded data from peers translates to fewer video piece missing, hence reducing data serviced from CDN servers. However, the existing academic approaches are ineffective or inadequate in optimizing the upload bandwidth utilization for peers. The industry practice is a heuristic based on the additive-increase/multiplicativedecrease (AIMD) from TCP; it lacks theoretical foundations thus is not extensible. Guided by theoretical foundation, this paper develops an optimal bandwidth allocation algorithm for hybrid VoD Streaming. We also convert the theoretical design to a practical implementation by meeting real system requirements, such as piece scheduling and topology dynamics. Confirmed by extensive evaluations, the performance of *Free-for-All Push–Lift* is better than the industry AIMD heuristic.

Acknowledgment

The authors would like to thank anonymous reviewers for their valuable comments. This work is partially supported by National Natural Science Foundation of China (no. 61202107, 61371141, 61202303), by National High Technology Research and Development Program of China (863 program no. 2014AA01A702), by Natural Science Foundation of Hubei Province (no. 2014CFB1007), by National Key Technology Research and Development Program of China (no. 2012BAH46F03), by National Science Foundation of USA (nos. 1440737, 1541434, 1450996 and 1530989), and by Fundamental Research Funds for the Central Universities.

Appendix A. Proof

Lemma 3. *During the execution of the algorithm, for any* $a, b \in V$ *and* $(a, b) \in E$, $h_a(b)$ *never decreases. And the equation* $h_a(b) \leq h(b)$ *always holds.*

$h_a(b)$ indicates the height of b recorded in a, it will never be updated until a receives a higher height from b, because of Lemma 1, $h(b)$ never decreases, it is straightforward to see that $h_a(b)$ never decreases and $h_a(b) \leq h(b)$.

Lemma 4. *If the algorithm terminates, then* $h(u) \leq h(v) + 1$ *for any edge* $(u, v) \in E_f$.

The proof enumerates all interleaved execution scenarios of *push* and *lift* operations. The initialization state satisfies the requirement after a *pre-flow* operation, now we discuss the following scenarios possibly happen afterwards.

- ***lift(a)*operation.** This operation is executed in its entirety without interleaving with any other operations. For the residual edge (a, b) that leaves a, before the *lift* implied, we get $h_a(b) > h(a)$, since b is one of the neighbors of a, the new height of a after *lift* operation will never exceed $h_a(b)$. So we can get $h(a) \leq h_a(b) + 1$ after *lift(a)* operation, the equation $h(a) = h_a(b) + 1$ is achieved when node b is the lowest neighbor of a. For the residual edge (c, a) that enters a, we have $h_c(a) > h(c)$ before the *lift* operation is applied, and the height of a never decreases since from Lemma 1, so we obtain $h(c) \leq h_c(a) + 1$ afterwards.

  Lemma 3 leads to $h_a(b) \leq h(b)$. So, for the first case that leaves a, $h(a) \leq h_a(b) + 1 \leq h(b) + 1$, for the second case that enters a, $h(c) \leq h_c(a) + 1 \leq h(a) + 1$.
- ***push(a,b)* operation.** This *push* operation is also executed in its entirety without interleaved with any other operations. Note that we must have $(a, b) \in E_f$ and $h(a) > h_a(b)$ for *push(a,b)* to be applicable. Under this scenario, there are two cases that may occur.
  1. The *push(a,b)* operation is successful. This case implies that $h(a) > h(b)$ and $h_b(a)$ is updated to be equal to the current $h(a)$. Thus we have $h(b) \leq h(a) + 1$. If this operation removes (a, b) from E_f, the removal of residual edge also removes the requirement that $h(a) \leq h(b) + 1$. If *push(a,b)* does not remove (a, b) from E_f, which means $(a, b) \in E_f$ and $(b, a) \in E_f$. We can obtain $h(b) \leq h(a) + 1$ from above discussion. For the result of $h(a) \leq h(b) + 1$, we can start iterating from the initialization state. After *pre-flow* operation, the state satisfies Lemma 4, and thus $h(a) \leq h(b) + 1$. Also, we got $h(a) > h(b)$ from the successful *push*, so we can get $h(a) = h(b) + 1$, and the in-equation $h(a) \leq h(b) + 1$ still holds after the operation *push(a,b)*. This iteration will continue with the same result under this case. This will lead to $h(a) \leq h(b) + 1$ at last.

2. The *push(a,b)* operation is an *error push*. This case implies that $h(a) \le h(b)$, so b will push the flow back to a, $h_b(a)$ is also updated to be equal to current $h(a)$. If the process of *error push* removes the (b, a) in E_f and inform a of the latest height of b, the requirement that $h(b) \le h(a) + 1$ will also be canceled. We got $h(a) \le h(b)$, and obviously $h(a) \le h(b) + 1$ is satisfied. If $(b, a) \in E_f$ exists after the *back push* operation, $h(a) \le h(b) + 1$ can be obtained from above discussion. Also, due to the existence of (b, a) from E_f, we are sure that a at least pushes to b once successfully. Right after that successful *push(a,b)*, there is $h_b(a) > h(b)$, so even after b lifts its height, from the discussion of scenario 1, we can get $h(b) \le h_b(a) + 1$, and because of $h_b(a) \le h(a)$, we get $h(b) \le h(a) + 1$.

- ***lift(a)* and *lift(b)* interleaved.** For this scenario, we may have four cases to discuss.
  1. $(a, b) \in E_f$ and $(b, a) \in E_f$. In this case, we must have $h(a) = h_a(b)$ and $h_b(a) = h(b)$, or *push* operation will be applied. Because of $h_b(a) \le h(a)$ and $h_a(b) \le h(b)$, the four formulas give us $h(b) \le h(a)$ and $h(a) \le h(b)$, which is $h(a) = h(b) = h_a(b) = h_b(a)$. If *lift(a)* is applied, we can get $h(a) \le h_a(u)$, where $(u, a) \in E_f$. So u is exactly b and the min height will be just $h_a(b)$, and *lift(a)* sets $h(a) = h_a(b) + 1$. Similarly, *lift(b)* sets $h(b) = h_b(a) + 1$. So after two *lift* operations from a and b, we can still get $h(a) = h(b)$. Thus $h(a) \le h(b) + 1$ is maintained for the residual edge (a, b) and $h(b) \le h(a) + 1$ is maintained for the residual edge (b, a).
  2. $(a, b) \in E_f$ and $(b, a) \notin E_f$. In this case, we can get $h(a) < h_a(b)$ from the single direction residual edge, also we know $h_a(b) \le h(b)$. *lift(a)* sets $h(a) = min\{h_a(v)\} + 1$ where $(a, v) \in E_f$. This ensures $h(a) \le h_a(b) + 1$. $h(b)$ even further increases after *lift* operation. So $h(a) \le h(b) + 1$.
  3. $(a, b) \notin E_f$ and $(b, a) \in E_f$. This case is symmetric to case (b).
  4. $(a, b) \notin E_f$ and $(b, a) \notin E_f$. This case implies that there is no residual edge between a and b, which means the *lift* operations of a and b are not constrained by each other.
- ***push(a,b)* and *push(b,c)* interleaved.** *push(a,b)* can be executed either before or after *push(b,c)* in its entirety. So this scenario can be reduced to scenario 2 with the same analysis.
- ***push(a,b)* and *lift(b)* interleaved.** We have two cases to discuss based on the result of *push* operation.
  1. The *push(a,b)* operation is successful. In this case, if *lift(b)* applies first, then *lift* has nothing to do with the *push(a,b)* operation afterwards, and this case can be reduced to scenario 1 and 2, respectively. If the *push(a.b)* operates first, then $h(a) > h(b)$ and $h_b(a)$ updates to $h(a)$. From scenario 1, we can get $h(b) \le h_b(a) + 1$, thus $h(b) \le h(a) + 1$.
  2. The *push(a,b)* operation is an *error push*. Due to the process of *error push*, b will send the same flow back to a. Consequently, *push(a,b)* actually does not apply, and this case can be reduced to scenario 1, only *lift(b)* applies.
- ***push(a,b)* and *lift(a)* interleaved.** This scenario is impossible. From Lemma 2, either a *push* or a *lift* operation can be applied. These two operations can never apply at the same time within one node.
- ***push(a,b)* and *push(b,a)* interleaved.** Since $h(a) > h(b)$ and $h(a) < h(b)$ cannot be satisfied at the same time, it is easy to see that at least one *push* will fail. So we discuss the possible two cases based on the number of *error push*.
  1. One *error push*. This case implies that $h(a) \ne h(b)$. Suppose $h(a) > h(b)$, then *push(b,a)* is an *error push*, the result is equivalent to single *push(a,b)*, so this case can be reduced to scenario 2. For $h(a) < h(b)$ vice versa.
  2. Two *error pushes*. This case will only happen when $h(a) = h(b)$. And the result of this case is equivalent to nothing happens, all the equations before two *error pushes* will remain the same afterwards.
- **More than two operations interleaved.** The discussion is similar to the above and conclusion is the same, details are omitted.

Theorem 2. *Given graph G, if the algorithm terminates, the calculated flow f is the maximum flow for graph G.*

Assume there is a path from s to t, which is marked as $s \to u_1 \to \ldots \to u_k \to t$ where $k \le n - 2$. According to Lemma 4, since each edge of this path is in E_f, we have

$$h(s) \le h(u_1) + 1, h(u_1) \le h(k) + 1, \ldots, h(u_{n-2}) \le h(t) + 1$$

The number of nodes in the path is $n - 2$ apart from s and t, so $h(s) \le h(t) + n - 1$. However, when initializing the source and sink nodes, we set $h(s) = |V| = n$ and $h(t) = 0$, and the height of s or t never changes during the execution. The existence of the path contradicts the fact that $h(s) = h(t) + n$, so there is no path from s to t when the algorithm terminates. Based on Theorem 1, no path can reach t from s in G_f, so the flow f is maximum flow for graph G.

References

[1] Cisco Visual Networking Index: Forecast and Methodology, 2011-2016.
[2] H.H. Liu, Y. Wang, Y.R. Yang, H. Wang, C. Tian, Optimizing cost and performance for content multihoming, ACM SIGCOMM Comput. Commun. Rev. 42 (4) (2012) 371–382.
[3] H. Mao, C. Tian, J. Sun, J. Yan, W. Wu, B. Huang, Shadow VoD: performance evaluation as a capability in production P2P-CDN hybrid VoD networks, in: Proceedings of the USDE, IEEE, 2014, pp. 1–6.
[4] X. Zhang, J. Liu, B. Li, Y.-S. Yum, Coolstreaming/donet: a data-driven overlay network for peer-to-peer live media streaming, in: Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies INFOCOM 2005, 3, IEEE, 2005, pp. 2102–2111.
[5] Y. Huang, T.Z. Fu, D.-M. Chiu, J. Lui, C. Huang, Challenges, design and analysis of a large-scale P2P-VoD system, in: Proceedings of the ACM SIGCOMM Computer Communication Review, 38, ACM, 2008, pp. 375–388.
[6] H. Yin, X. Liu, T. Zhan, V. Sekar, F. Qiu, C. Lin, H. Zhang, B. Li, Design and deployment of a hybrid CDN-P2P system for live video streaming: experiences with LiveSky, in: Proceedings of the 17th ACM International Conference on Multimedia, ACM, 2009, pp. 25–34.
[7] C. Tian, H. Jiang, X. Liu, W. Liu, Revisiting dynamic query protocols in unstructured peer-to-peer networks, IEEE Trans. Parallel Distrib. Syst. 23 (1) (2012) 160–167.
[8] C. Tian, H. Jiang, W. Liu, X. Liu, Y. Wang, Towards minimum traffic cost and minimum response latency: a novel dynamic query protocol in unstructured P2P networks, in: Proceedings of the 37th International Conference on Parallel Processing, 2008. ICPP'08, IEEE, 2008, pp. 1–8.

[9] H. Yin, B. Qiao, Y. Luo, C. Tian, Y.R. Yang, Demystifying commercial content delivery networks in china, Concurrency Comput.: Pract. Exp. 27 (13) (2015) 3523–3538.
[10] C. Tian, Y. Wang, Y. Luo, H. Jiang, W. Liu, J. Wu, H. Yin, Minimizing content reorganization and tolerating imperfect workload prediction forcloud-based Video-on-Demand services, IEEE Trans. Serv. Comput. Preprint, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7069258.
[11] M. Thornburgh, Adobe secure real-time media flow protocol (2013).
[12] C. Tian, R. Alimi, Y.R. Yang, D. Zhang, Shadowstream: performance evaluation as a capability in production internet live streaming networks, in: Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, ACM, 2012, pp. 347–358.
[13] J. Yan, C. Tian, J. Sun, H. Mao, Improve distributed client lifecycle control in shadowstream, Int. J. Web Serv. Res. (IJWSR) 11 (4) (2014) 62–78.
[14] R. Kumar, Y. Liu, K. Ross, Stochastic fluid theory for P2P streaming systems, in: Proceedings of the 26th IEEE International Conference on Computer Communications. IEEEINFOCOM 2007., IEEE, 2007, pp. 919–927.
[15] T. Bonald, L. Massoulié, F. Mathieu, D. Perino, A. Twigg, Epidemic live streaming: optimal performance trade-offs, in: Proceedings of the ACM SIGMETRICS Performance Evaluation Review, 36, ACM, 2008, pp. 325–336.
[16] C. Zhao, X. Lin, C. Wu, The streaming capacity of sparsely-connected P2P systems with distributed control, in: Proceedings of the INFOCOM, 2011. IEEE, IEEE, 2011, pp. 1449–1457.
[17] V. Jacobson, Congestion avoidance and control, in: Proceedings of the ACM SIGCOMM Computer Communication Review, 18, ACM, 1988, pp. 314–329.
[18] Y. Pan, J.Y. Lee, Adaptive scheduling of data transfer in P2P applications over asymmetric networks, in: Proceedings of the 2010 IEEE International Conference on Communications (ICC), IEEE, 2010, pp. 1–5.
[19] Q. Yu, D. Chen, Optimal data scheduling for P2P VoD streaming systems, in: Proceedings of the 2010 IEEE 16th International Conference on Parallel and Distributed Systems (ICPADS), IEEE, 2010, pp. 817–822.
[20] Y. He, I. Lee, L. Guan, Distributed throughput maximization in P2P VoD applications, IEEE Trans. Multimedia 11 (3) (2009) 509–522.
[21] S. Sengupta, S. Liu, M. Chen, M. Chiang, J. Li, P.A. Chou, Streaming capacity in peer-to-peer networks with topology constraints, Technical Report, Citeseer, 2008.
[22] Y. Cui, B. Li, K. Nahrstedt, On achieving optimized capacity utilization in application overlay networks with multiple competing sessions, in: Proceedings of the 16th Annual ACM Symposium on Parallelism in Algorithms and Architectures, ACM, 2004, pp. 160–169.
[23] A. Bradai, T. Ahmed, R. Boutaba, R. Ahmed, Efficient content delivery scheme for layered video streaming in large-scale networks, J. Netw. Comput. Appl. 45 (2014) 1–14.
[24] D. Wu, C. Liang, Y. Liu, K. Ross, View-upload decoupling: a redesign of multi-channel P2P video systems, in: Proceedings of the INFOCOM 2009, IEEE, IEEE, 2009, pp. 2726–2730.
[25] B. Cohen, Incentives build robustness in bittorrent, in: Proceedings of the Workshop on Economics of Peer-to-Peer systems, 6, 2003, pp. 68–72.
[26] D. Qiu, R. Srikant, Modeling and performance analysis of bittorrent-like peer-to-peer networks, in: Proceedings of the ACM SIGCOMM Computer Communication Review, 34, ACM, 2004, pp. 367–378.
[27] F. Clévenot-Perronnin, P. Nain, K.W. Ross, Multiclass P2P networks: static resource allocation for service differentiation and bandwidth diversity, Perform. Eval. 62 (1) (2005) 32–49.
[28] S. Sundaresan, W. De Donato, N. Feamster, R. Teixeira, S. Crawford, A. Pescapè, Broadband internet performance: a view from the gateway, in: Proceedings of the ACM SIGCOMM Computer Communication Review, 41, ACM, 2011, pp. 134–145.
[29] N. Magharei, R. Rejaie, Y. Guo, Mesh or multiple-tree: a comparative study of live P2P streaming approaches, in: Proceedings of the 26th IEEE International Conference on Computer Communications INFOCOM 2007, IEEE, IEEE, 2007, pp. 1424–1432.
[30] S. Shalunov, G. Hazel, J. Iyengar, M. Kuehlewind, Low extra delay background transport (LEDBAT), Technical Report, 2012.
[31] J. Schneider, J. Wagner, R. Winter, H.-J. Kolbe, Out of my way-evaluating low extra delay background transport in an ADSL access network, in: Proceedings of the 22nd International Teletraffic Congress (ITC), 2010, IEEE, 2010, pp. 1–8.
[32] H. Schwarz, D. Marpe, T. Wiegand, Overview of the scalable video coding extension of the H.264/AVC standard, IEEE Trans. Circuits Syst. Video Technol. 17 (9) (2007) 1103–1120.
[33] Y. Wang, A.R. Reibman, S. Lin, Multiple description coding for video delivery, Proc. IEEE 93 (1) (2005) 57–70.
[34] H. Hu, X. Zhu, Y. Wang, R. Pan, J. Zhu, F. Bonomi, Proxy-based multistream scalable video adaptation over wireless networks using subjective quality and rate models, IEEE Trans. Multimedia 15 (7) (2013) 1638–1652.
[35] E. Magli, M. Wang, P. Frossard, A. Markopoulou, Network coding meets multimedia: a review, IEEE Trans. Multimedia 15 (5) (2013) 1195–1212.
[36] B. Li, J. Liu, Multirate video multicast over the internet: an overview, IEEE Netw. 17 (1) (2003) 24–29.
[37] T.-Y. Huang, N. Handigol, B. Heller, N. McKeown, R. Johari, Confused, timid, and unstable: picking a video streaming rate is hard, in: Proceedings of the 2012 ACM Conference on Internet Measurement Conference, ACM, 2012, pp. 225–238.
[38] D. Ford, D.R. Fulkerson, Flows in Networks, Princeton University Press, 2010.
[39] B.V. Cherkassky, A.V. Goldberg, On implementing the push-relabel method for the maximum flow problem, Algorithmica 19 (4) (1997) 390–410.
[40] D.A. Bader, V. Sachdeva, A cache-aware parallel implementation of the push-relabel network flow algorithm and experimental evaluation of the gap relabeling heuristic (2006).
[41] R.J. Anderson, J.C. Setubal, Goldberg1s algorithm for maximum flow in perspective: a computational study, Netw. Flows Matching: First DIMACS Implement. Chall. 12 (1993) 1.
[42] B. Hong, A lock-free multi-threaded algorithm for the maximum flow problem, in: Proceedings of the IEEE International Symposium on Parallel and Distributed Processing, 2008. IPDPS 2008., IEEE, 2008, pp. 1–8.
[43] Z. He, B. Hong, Dynamically tuned push-relabel algorithm for the maximum flow problem on CPU-GPU-Hybrid platforms, in: Proceedings of the IEEE International Symposium on Parallel & Distributed Processing (IPDPS), 2010, IEEE, 2010, pp. 1–10.
[44] A. Lahanas, V. Tsaoussidis, Exploiting the efficiency and fairness potential of AIMD-based congestion avoidance and control, Comput. Netw. 43 (2) (2003) 227–245.

Chen Tian received the BS, MS and Ph.D degrees from the School of Electronic Information and Communications, Huazhong University of Science and Technology, China, in 2000, 2003, and 2008 respectively. He is an associate professor in the School of Electronic Information and Communications, the Huazhong University of Science and Technology, China. His research interests include data center networks, software defined networks, cloud computing, distributed networks, Internet streaming and network architecture.

Jingdong Sun was born in Shandong, China in 1991. He received the B.E. degree from the School of Electronic Information and Communications, Huazhong University of Science and Technology, Wuhan, China in 2014. He is now a master student in the Department of Electrical Engineering, Missouri University of Science and Technology. His research interests include network communication, signal integrity and power integrity. He is currently an IEEE student member, and the recipient of the IEEE EMC Hardware Design Award in 2015.

Weimin Wu received the B.E. degree in Computer Software from Xidian University, Xi'an, China in 1992, and the M.E. degree in Computer Application from Sichuan University, Chengdu, China in 1995. He obtained the Ph.D degree in Communications and Information Systems from the School of Electronic Information and Communications, Huazhong University of Science and Technology, Wuhan, China, in 2007, where he is currently an associate professor with the School of Electronic Information and Communications, Huazhong University of Science and Technology. As a leader or main participant, he has completed a number of national projects in wireless communications including 863 Projects and National Science and Technology Major Projects. His current research interests are in the areas of broadband wireless communications, multimedia and networking.

Yan Luo is an associate professor of the Department of Electrical and Computer Engineering at the University of Massachusetts Lowell. He earned his Ph.D in Computer Science from the University of California Riverside in 2005 and his BE and ME degrees from Huazhong University of Science and Technology. His research spans broadly computer architecture and network systems. Prof. Luo's current projects focus on heterogeneous architecture and systems, software defined networks and deep learning. He has served on the program committee of numerous international conferences and as a guest editor and referee of premier journals. He is a member of IEEE and ACM.