On-line Stream Merging with Max Span and Min Coverage

Wun-Tat Chan* Tak-Wah Lam† Hing-Fung Ting†‡ Prudence W.H. Wong†

November 25, 2003

Abstract

This paper introduces the notions of span and coverage for analyzing the performance of on-line algorithms for stream merging. It is shown that these two notions can solely determine the competitive ratio of any such algorithm. Furthermore, we devise a simple greedy algorithm that attains the ideal span and coverage, thus giving a better performance guarantee than existing algorithms. The new notions also allow us to obtain a tighter analysis of existing algorithms.

1 Introduction

A typical problem encountered in video-on-demand (VOD) systems is that many requests for a particular popular video are received over a short period of time (say, Friday evening). If a dedicated video stream is used to serve each request, the total bandwidth requirement for the server is enormous. To reduce the bandwidth requirement without sacrificing the response time, a popular approach is to merge streams initiated at different times (see e.g., [1, 3–5, 8–15]).

Stream merging is based on a multicasting architecture and assumes that each client has extra bandwidth to receive data from two streams simultaneously. In such a system, a stream can run in two different states: normal state and exceptional state. A new stream Y is initially in *normal* state and all its clients receive one unit of data from Y in every time unit for immediate playback. Some time later, Y may change to the *exceptional* state and all its clients receive, in every time unit, one unit of data from Y and one unit of data from an earlier stream X (say, initiated Δ time units before Y). When Y's clients have buffered enough data from X (i.e., Δ units), they can synchronize with the playback of X and can switch to X. At this time, Y can terminate and it is said

*Department of Computing, Hong Kong Polytechnic University, Hong Kong, cswtchan@comp.polyu.edu.hk

†Department of Computer Science, University of Hong Kong, Hong Kong, {twlam,hfting,whwong}@cs.hku.hk

‡This research was supported in part by Hong Kong RGC Grant HKU7045/02E.

to *merge* with X. Note that stream merging reduces the total bandwidth requirement. In general, a VOD system may not allow each client to receive up to one extra unit of data per time unit; in this paper a VOD system is characterized by an integer parameter $\lambda \geq 1$, which means that a client can receive a total of $1 + 1/\lambda$ units of data per time unit.

To support stream merging effectively, we need an on-line algorithm to decide how streams merge with each other. The performance of such an on-line algorithm can be measured rigorously using the competitive ratio, i.e., the worst-case ratio between its total bandwidth and the total bandwidth used in an optimal schedule. The literature contains a number of on-line stream merging algorithms, e.g., the greedy algorithm [3] (also called nearest-fit), the Dynamic Fibonacci tree algorithm [3], the connector algorithm [7], and the α-dyadic algorithm [9]. The greedy heuristic is attractive because of its simplicity and ease of implementation; a stream simply merges with the nearest possible stream. Unfortunately, it has been shown to be $\Omega(n/\log n)$-competitive, where n is the total number of requests [3]. The other three algorithms provide much better performance guarantee; in particular, the connector algorithm and the α-dyadic algorithm are known to be 3-competitive [6, 7]. Yet these algorithms are much more complicated than the greedy algorithm. The Dynamic Fibonacci tree algorithm is based on a data structure called Fibonacci merge tree, the connector algorithm needs to pre-compute a special reference tree to guide the on-line algorithm, and the α-dyadic algorithm is recursive in nature.

In reality, it might make more sense for a stream merging algorithm to minimize the maximum bandwidth over time instead of the total bandwidth [2]. In [6], we consider the special case where the extra bandwidth parameter λ is equal to one (i.e., a client can receive 1 unit of normal data and 1 unit of extra data in one time unit) and show that with respect to the maximum bandwidth, the connector algorithm is 4-competitive and the α-dyadic algorithm is 4-competitive when α is chosen to be 1.5. Empirical studies indeed confirm that the connector algorithm and the α-dyadic algorithm do have very similar performance under different measurements [2, 16].

In this paper, we attempt to improve the analysis of existing algorithms and design a new algorithm with a better competitive ratio. With a deeper thought, we also want to identify the key elements in designing a good stream merging algorithm and explain why the connector algorithm and the dyadic algorithm have similar performance.

When designing a stream merging algorithm, there are two conflicting concerns in determining how long a stream should run before it merges. Obviously, we want to merge a stream with an earlier stream early enough so as to minimize the bandwidth requirement. Yet we also want a stream to run long enough so that more streams initiated later can merge with it. Good algorithms such as the connector algorithm and the α-dyadic algorithm must be able to balance these two concerns properly. In this paper we show how these two concerns can be measured rigorously and more importantly, can be used to determine the competitive ratio of an algorithm. More precisely, we introduce the notions of span factor and coverage factor, and show that if a stream merging algorithm has a span factor at most s and a coverage factor at least c, then it is $K_{c,s}$-competitive

	Connector algorithm	Dyadic algorithm	Our greedy algorithm
maximum bandwidth	3 (4 when $\lambda = 1$)	2 (4 when $\lambda = 1$)	2
total bandwidth	3 (3)	2.5 (3)	2.5

Table 1: Competitive ratios of different algorithms. The values enclosed are the previously best results. Unless otherwise specified, the ratios are valid for all possible $\lambda \geq 1$.

with respect to the maximum bandwidth and $(K_{c,s} + \frac{1}{1+\lambda})$-competitive with respect to the total bandwidth, where $K_{c,s} = 1 + \max\left\{\lceil \log_{1+c} \frac{1+2\lambda}{1+\lambda} \rceil, \lceil \log_{1+\frac{1+\lambda}{s\lambda}} \frac{1+2\lambda}{\lambda} \rceil\right\}$. Note that $K_{c,s}$ attains the smallest value of 2 when $s = 1$ and $c = \lambda/(1+\lambda)$.

Another contribution of this paper is a simple greedy algorithm that guarantees the ideal span factor and coverage factor, i.e., 1 and $\lambda/(1+\lambda)$, respectively (or in general, given any number s, guarantees a span factor at most s and a coverage factor at least $s\lambda/(1+\lambda)$). In other words, this greedy algorithm is 2-competitive with respect to the maximum bandwidth, and 2.5-competitive with respect to the total bandwidth. This result improves existing work regarding the competitiveness and generality.

The notions of span and coverage factors also help us obtain a tighter analysis of the existing algorithms. For the connector algorithm, we find that the span factor is at most $\frac{1}{2}\frac{1+\lambda}{\lambda}$ and the coverage factor is at least $1/2$; thus, the connector algorithm is 3-competitive with respect to either the maximum or the total bandwidth. The α-dyadic algorithm has a better performance, its span factor is at most $(\alpha - 1)\frac{1+\lambda}{\lambda}$ and coverage factor at least $\alpha - 1$. When α is chosen as $\frac{1+2\lambda}{1+\lambda}$, the competitive ratio is exactly 2 and 2.5 with respect to the maximum and the total bandwidth, respectively. See Table 1 for a summary.

The technique used in this paper is drastically different from the so-called "schedules-sandwiching" technique used in our previous work to analyze the connector and α-dyadic algorithms. A basic step of the schedules-sandwiching technique is to "enlarge" the on-line schedule to make it more regular for comparison with an optimal schedule. The disadvantage is that the enlarged schedule may increase the bandwidth requirement. Another major reason for not using this technique in this paper is that we have no idea how the actual on-line schedules look like (since our analysis is based on any schedule satisfying the bounds on the span factor and coverage factor). Our analysis is based on a more generic argument which makes use of some deeper structural properties that are guaranteed by the span and coverage bound.

Our paper is organized as follows. In Section 2, we introduce a formal model on stream merging. In Section 3, we define the notions of max span and min coverage of a merging schedule and we show a simple greedy algorithm that always constructs schedules with some given span and coverage. In Section 4, we describe a geometric representation of schedules. We give the competitiveness analysis in Section 5. Note that our algorithm, as well as existing algorithms, only consider request sequences that are compact (to be defined later). In Section 6, we show that if we have an algorithm with a good competitive ratio for compact sequences, we have an algorithm with good competitive ratio for general sequences.

2 The model

In a VOD system, a server and a set of clients are connected through a network. We focus on scheduling of one particular popular video, which is ℓ units of length. Let λ be a fixed positive integer. We assume that each client can receive up to $1 + 1/\lambda$ units of video in one unit of time from possibly two streams, and each client can buffer up to $\ell/(1 + \lambda)$ units of the video. At any time, a client may request for the video and a new stream is initiated to serve it immediately.

A stream has two states: *normal* and *exceptional*. Initially, a stream, say Y, is in the normal state and all of its clients will receive one unit of video from Y in one time unit. At any time, Y may change to the exceptional state. Once in the exceptional state, Y is coupled with an earlier stream X and Y's clients will receive, in one time unit, a total of $1 + 1/\lambda$ units of video from Y and X. Y remains in the exceptional state until it *merges* with X, i.e., Y terminates and all its clients switch to listen to X and become X's clients. Intuitively, Y merges with X only when the clients of Y have received the same amount of data as clients of X. This condition for a merging to occur can be formally stated as follows [7]:

Condition 1. *Suppose that X and Y are two streams initiated at time t_X and t_Y, respectively, where $t_X < t_Y$. If Y merges with X, then*

- *Y runs in exceptional state for exactly $\lambda(t_Y - t_X)$ time units and its clients receive an extra of $(t_Y - t_X)$ units of video from X; and*
- *if Y remains in normal state for τ time units, i.e., it merges X at time $t_Y + \tau + \lambda(t_Y - t_X)$, then X must be in normal state at time $t_Y + \tau + \lambda(t_Y - t_X)$ (which can be rewritten as $t_X + \tau + (1 + \lambda)(t_Y - t_X)$).*

Consider a set of request sequences. The stream merging problem is to find a schedule to determine, for every stream multicasted by the system, how long the stream should be in normal and exceptional states and which stream it should merge with. The objective is to minimize either the maximum bandwidth, which is the maximum number of streams running at any time, or the total bandwidth, which is the total duration of all the streams. We say that an on-line stream merging algorithm A is c-competitive with respect to maximum bandwidth (resp. total bandwidth) if A will always produce a schedule with maximum bandwidth (resp. total bandwidth) at most c times that of an optimal schedule. Given any schedule $\mathcal{S}$, let $\texttt{load}(\mathcal{S}, t)$, the *load* of $\mathcal{S}$ at any time t, be the number of streams running at t. The maximum and total bandwidth is equal to $\sum_t \texttt{load}(\mathcal{S}, t)$ and $\max_t \texttt{load}(\mathcal{S}, t)$, respectively.

Consider a request sequence $\mathcal{R} = (t_1, t_2, \ldots, t_n)$, where each t_i denotes the i-th arrival time. We say that $\mathcal{R}$ is *compact* if $t_n - t_1 \le \ell/(1+\lambda)$ (recall that ℓ is the length of the video). A key property of compact sequences is that except for the stream for the first request, which must be a full stream, streams initiated for any other requests can merge with an earlier stream. Note that all previous works first focus on scheduling compact sequences. This is because results on compact sequences can usually be extended easily to general sequences. The following two lemmas capture such generalization for the case of maximum bandwidth and total bandwidth, respectively.

Lemma 1. [6] *Assume that $\lambda = 1$. Let c be a positive number. Suppose that A is an on-line stream merging algorithm such that given any compact sequence C, A produces a schedule $\mathcal{S}$ with $\texttt{load}(\mathcal{S}, t) \le c\,\texttt{load}(T, t)$ for any schedule T for C and any time t. Then, we can construct from A a stream merging algorithm that is c-competitive with respect to maximum bandwidth for any general request sequence.*

Lemma 1 can also be generalized for arbitrary λ. For the sake of completeness, we give the proof in Section 6.

Lemma 2. [7] *Let c be a positive number. Suppose that A is an on-line stream merging algorithm such that given any compact sequence C, A produces a schedule $\mathcal{S}$ for C with $\texttt{load}(\mathcal{S}, t) \le c\,\texttt{load}(T, t)$ for any schedule T for C and any time t. Then, we can construct from A a stream merging algorithm that is* $\max\{3, c\}$*-competitive with respect to total bandwidth for any general request sequence.*

Note that the best on-line algorithm obtained by the technique of Lemma 2 is no better than 3-competitive for general sequences. In Section 6, we give a more elaborate analysis and show that a special class of algorithms for compact sequences, when adapted to general sequences, can circumvent the barrier of 3 and achieve a competitive ratio of 2.5.

In the rest of the paper (except Section 6), we assume that the input sequence is compact, and consider only schedules in which there is only one full stream (i.e., the first stream) and all other streams will eventually merge with some earlier streams.

3 Span and coverage

In this section, we define two measures of a schedule $\mathcal{S}$, namely, the maximum span and minimum coverage, which can capture the load of $\mathcal{S}$. Then we devise a simple greedy on-line algorithm that can construct schedules with desired bounds on the span and coverage.

First, we define the span factor and coverage factor of any stream $X \in \mathcal{S}$ that is not a full stream, i.e., X will change to exceptional state to merge with an earlier stream. Suppose that X is initiated at time t_X, and it runs in normal state and exceptional state for τ_n and τ_e time units, respectively.

The *span factor* $\texttt{Sf}(X)$ of X is defined to be the ratio τ_n/τ_e.

To define the coverage factor of X, we need an additional definition. For any time $t > t_X$, we say that X *covers* t if the following is true: if we initiate a stream at t and let it change immediately to exceptional state, it can merge with X; by Condition 1, it is equivalent to say that X is still in normal state at time $t_X + (1+\lambda)(t - t_X)$. Let

- t_{parent} be the initiation time of the stream with which X merges;
- t_{before} be the initiation time of the stream immediately before X (i.e., the latest time before t_X at which there is a stream initiated);
- t_{miss} be the first time after t_X such that there is a stream initiated at t_{miss}, and X does not cover t_{miss} (we set $t_{\text{miss}} = \infty$ if there is no such stream after X).

The *coverage factor* $\texttt{Cf}(X)$ of X is defined to be the ratio of the length of the two intervals $[t_{\text{before}}, t_{\text{miss}}]$ and $[t_{\text{parent}}, t_{\text{before}}]$, i.e.,

$$\texttt{Cf}(X) = \frac{|[t_{\text{before}}, t_{\text{miss}}]|}{|[t_{\text{parent}}, t_{\text{before}}]|}$$

where $|I|$ denotes the length of the interval I. To understand the meaning of $\texttt{Cf}(\text{X})$, note that $|[t_{\text{before}}, t_{\text{miss}}]| = |[t_{\text{before}}, t_X]| + |[t_X, t_{\text{miss}}]|$, and X covers any time in $[t_X, t_{\text{miss}}]$ at which there is a stream initiated. Thus, a large $\texttt{Cf}(X)$ means that if X is scheduled to merge a stream initiated much earlier than t_{before} (i.e., if $|t_{\text{parent}}, t_{\text{before}}]|$ is large), then either $[t_{\text{before}}, t_X]$ is already large, or the interval $[t_X, t_{\text{miss}}]$ covered by X is large.

The *minimum coverage* of $\mathcal{S}$ is the minimum of $\texttt{Cf}(X)$, and the *maximum span* of $\mathcal{S}$ is the maximum of $\texttt{Sf}(X)$, over all streams $X \in \mathcal{S}$ that are not full streams.

We say that a schedule is (c, s)-bounded if its minimum coverage is at least c and maximum span is at most s. In Section 5, we analyze the competitiveness of (c, s)-bounded schedules. In the rest of this section, we focus on the construction of schedules with desired bounds on the span and coverage factors. In particular, we devise, for any $s > 0$, an on-line algorithm $\mathcal{G}_s$ which, given any request sequence, always returns a schedule for the sequence that is $(\frac{\lambda}{1+\lambda}s, s)$-bounded.

Roughly speaking, $\mathcal{G}_s$ is just a greedy algorithm with some slight modification. It still schedules a stream to merge with the nearest stream that it can merge with. However, a stream will not change to exceptional state immediately after it is initiated. Instead, it will first run in normal state so that some later streams can merge with it and terminate earlier. The interesting part is to decide how long a stream should run in normal state so that its span and coverage factors will be within the bounds. The algorithm shown below can actually determine the lifespan of every stream at the time it is initiated. See the details below.

Let $\delta = \frac{1+\lambda}{1+\lambda+s\lambda}$. For any time interval $[x, y]$, we say that a time $t \in [x, y]$ is a δ-*checkpoint* for $[x, y]$ if $t = x + \delta^i(y - x)$ for some integer $i \geq 0$. Consider any stream X. Suppose that X is initiated at time t_X. Our algorithm $\mathcal{G}_s$ schedules X as follows:

- If X is the first stream, then it is a full stream.

- Otherwise, let W be the latest stream that covers t_X. (Note that W exists because the first stream must cover t_X.) X will merge with W. Let t_{parent} be the initiation time of W, t_{last} the largest possible time covered by W, and t a δ-checkpoint of $[t_{\text{parent}}, t_{\text{last}}]$ immediately after t_X. X runs in normal state for $(1+\lambda)(t-t_X)$ time units and in exceptional state for $\lambda(t_X - t_{\text{parent}})$ time units. In other words, X runs in normal state long enough to cover t.

Note that we can determine W, t_{parent} and t_{last} when we schedule X because we know the lifespan of all streams before X. Furthermore, X can merge with W successfully because of the fact that W covers t_{last} implies that W is still in normal state at

$$t_{\text{parent}} + (1+\lambda)(t_{\text{last}} - t_{\text{parent}}) \geq t_{\text{parent}} + (1+\lambda)(t - t_{\text{parent}}),$$

which is equal to $t_X + (1+\lambda)(t-t_X) + \lambda(t_X - t_{\text{parent}})$, the time when X terminates. The following theorem characterizes the schedules constructed by $\mathcal{G}_s$.

Theorem 3. *Any schedule $\mathcal{S}$ constructed by $\mathcal{G}_s$ is $(\frac{\lambda}{1+\lambda}s, s)$-bounded.*

Proof. Consider any stream $X \in \mathcal{S}$ that is not a full stream. Suppose that X is initiated at time t_X, and it merges with an earlier stream W initiated at time t_{parent}. Let t_{before} be the initiation time of the stream immediately before X, and t_{miss} be the smallest time such that there is a stream initiated at t_{miss} and X does not cover t_{miss}. Recall that $\texttt{Cf}(X) = (t_{\text{miss}} - t_{\text{before}})/(t_{\text{before}} - t_{\text{parent}})$. We derive its lower bound as follows.

Let t_{last} be the largest time covered by W. Let $I = [t_{\text{parent}}, t_{\text{last}}]$, and $t_\ell = t_{\text{parent}} + \delta^{i+1}|I|$ and $t_r = t_{\text{parent}} + \delta^i|I|$ be the two δ-checkpoints for I that embrace t_X, i.e. $t_\ell < t_X \leq t_r$.

Note that (1) t_r is X's next δ-checkpoint and by construction, X covers t_r, and (2) by definition, X does not cover t_{miss}; thus, we have $t_r < t_{\text{miss}}$. We claim that $t_{\text{before}} \leq t_\ell$. Otherwise, the stream W' initiated at t_{before} will cover t_r, its next δ-checkpoint; this implies that W' also covers t_X, and by the greedy nature of $\mathcal{G}_s$, X will merge with W' instead of the earlier stream W. In summary, we have

$$t_{\text{parent}} \leq t_{\text{before}} \leq t_\ell < t_X \leq t_r < t_{\text{miss}}.$$

Recall that $t_\ell = t_{\text{parent}} + \delta^{i+1}|I|$ and thus $\delta^{i+1}|I| = t_\ell - t_{\text{parent}} \geq t_{\text{before}} - t_{\text{parent}}$. Finally, it can be verified that

$$t_{\text{miss}} - t_{\text{before}} > t_r - t_{\text{before}} \geq t_r - t_\ell = \frac{1-\delta}{\delta}\delta^{i+1}|I| \geq \frac{1-\delta}{\delta}(t_{\text{before}} - t_{\text{parent}})$$

and thus $\texttt{Cf}(X) = (t_{\text{miss}} - t_{\text{before}})/(t_{\text{before}} - t_{\text{parent}}) > \frac{1-\delta}{\delta} = \frac{\lambda}{1+\lambda}s$.

Now, we consider $\texttt{Sf}(X)$. Let τ_n and τ_e be the duration of X in normal and exceptional state, respectively. According to the algorithm, $\tau_n = (1+\lambda)(t_r - t_X)$, which is no greater than $(1+\lambda)(t_r - t_\ell) = (1+\lambda)(\delta^i|I| - \delta^{i+1}|I|)$, and $\tau_e = \lambda(t_X - t_{\text{parent}})$, which is no smaller than $\lambda(t_\ell - t_{\text{parent}}) = \lambda\delta^{i+1}|I|$. Hence,

$$\texttt{Sf}(X) = \frac{\tau_n}{\tau_e} \leq \frac{(1+\lambda)\delta^i|I|(1-\delta)}{\lambda\delta^{i+1}|I|} = \frac{1-\delta}{\delta}\frac{1+\lambda}{\lambda} = s.$$

In summary, for any stream $X \in \mathcal{S}$ that is not a full stream, we have shown that $\mathtt{Cf}(X) > \frac{\lambda}{1+\lambda}s$ and $\mathtt{Sf}(X) \leq s$. The theorem follows. $\square$

Notice that the best choice of s is 1 for dense sequences (there is a request arriving at every time unit). In this case, $\mathcal{G}_1$ is $(\frac{\lambda}{1+\lambda}, 1)$-bounded.

Remarks: As we mentioned in Section 1, we can bound the span and coverage of the α-dyadic algorithm [9] and the connector algorithm [7]. Below we describe how a dyadic schedule looks like and show that its span factor is at most $(\alpha-1)\frac{1+\lambda}{\lambda}$ and its coverage factor at least $\alpha - 1$. The dyadic algorithm decides a schedule recursively. Assume that there is a request X_0 arriving at t_0, and a stream is initiated for it. Consider a set S of requests that arrive in the interval $(t_0, t_1]$. Let X be the earliest request in S that arrives after $t_0 + (t_1 - t_0)/\alpha$, and Y be the last request in S. Suppose X and Y arrive at t_X and t_Y, respectively. According to the α-dyadic algorithm, a stream is scheduled for X that runs in normal state for $(1+\lambda)(t_Y - t_X)$ time units and then in exceptional state for $\lambda(t_X - t_0)$ time units before merging with the stream for X_0. For those requests arriving in $(t_0, \lfloor t_0 + (t_1 - t_0)/\alpha \rfloor]$, as well as those in $(t_X, t_1]$, they are handled recursively (there is no request in interval $(\lfloor t_0 + (t_1 - t_0)/\alpha \rfloor, t_X)$). Notice that

$$\mathtt{Sf}(X) = \frac{(1+\lambda)(t_Y - t_X)}{\lambda(t_X - t_0)} \leq \frac{(1+\lambda)(t_1 - (t_0 + (t_1 - t_0)/\alpha))}{\lambda((t_0 + (t_1 - t_0)/\alpha) - t_0)} = (\alpha - 1)\frac{1+\lambda}{\lambda}.$$

The inequality holds because $t_Y \leq t_1$, and $t_X \geq t_0 + (t_1 - t_0)/\alpha$. On the other hand,

$$\mathtt{Cf}(X) \geq \frac{|[t_X, t_{\mathrm{miss}}]|}{|[t_{\mathrm{parent}}, t_0 + (t_1 - t_0)/\alpha]|} \geq \frac{t_1 - (t_0 + (t_1 - t_0)/\alpha)}{t_0 + (t_1 - t_0)/\alpha - t_0} = \alpha - 1.$$

Therefore, the span factor and the coverage factor of the α-dyadic algorithm is at most $(\alpha - 1)\frac{1+\lambda}{\lambda}$ and at least $\alpha - 1$, respectively. The connector algorithm admits a weaker bounds on the span and coverage. One can also derive directly from the definition of the connector algorithm that the span and coverage is at most $\frac{1}{2}\frac{1+\lambda}{\lambda}$ and at least $1/2$, respectively.

4 Geometric representation of schedules

Our competitive analysis is based on a geometric representation of schedules introduced in [7] and a generalization of the notion of timeline introduced in [6] to arbitrary λ. This paper further introduce the concept of no waste schedules for analyzing the maximum load of any schedule. In the geometric representation, the streams are represented by some rectilinear lines on the plane. These lines are arranged in such a way that the load of the schedule at any time t is equal to the number of intersections between these rectilinear lines and some particular line $\mathcal{L}_t$. Thus, the representation enables us to reduce our problem of estimating the load to the combinatorial problem of estimating the number of intersections among some particular lines.

The geometric representation of a schedule $\mathcal{S}$ is constructed as follows. Let $X \in \mathcal{S}$ be a stream initiated at time t_X. Suppose that it runs in normal state and exceptional state

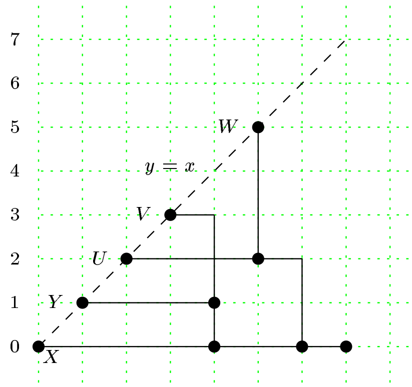

Figure 1: A schedule for streams X, Y, U, V and W, initiated at time $0, 1, 2, 3$, and 5, respectively.

for τ_n and τ_e time units, respectively. Then, starting from the point (t_X, t_X), we draw a crook $\mathcal{C}(X)$ on the plane, which is a right-going horizontal line of length $\tau_n/(1+\lambda)$ followed by a down-going vertical line of length τ_e/λ.

An important property of this representation is that if a stream X merges with a stream Y, then $\mathcal{C}(X)$ must terminate at some point on $\mathcal{C}(Y)$ (see [7] for a proof). Furthermore, note that all crooks start on the line $y = x$, and a full stream is a horizontal line terminating at $(t_1 + \ell/(1+\lambda), t_1)$, where t_1 is the initiation time of the stream. (Recall that ℓ is the length of the video.) Together with our assumption that the input is compact and that the first stream is the only full stream in the schedule, we observe that the geometric representation has the following property.

> If we put a node at each endpoint of a crook, the crooks in the representation form a tree in which all the leaves are on the line $y = x$, and the root is at $(t_1 + \ell/(1+\lambda), t_1)$ where t_1 is the initiation time of the first stream. Furthermore, the path from a leaf to the root is a monotonic rectilinear path in which every horizontal segment is right-going and every vertical segment is down-going.

See Figure 1 for an example. Now, we explain how to find the load of $\mathcal{S}$ from this representation.

Lemma 4. *For any stream X, it is still running at time t if and only if its crook $\mathcal{C}(X)$ passes through some point (a, b) satisfying $(1+\lambda)a - \lambda b = t$.*

Proof. Let t_X be the initiation time of X. Thus, $\mathcal{C}(X)$ starts at (t_X, t_X). First, suppose that $\mathcal{C}(X)$ passes through some (a, b) with $t = (1+\lambda)a - \lambda b$. Then it has a horizontal segment and a vertical segment of length at least $a - t_X$, and $t_X - b$, respectively. This implies that X runs in normal state and exceptional state for at least $(1+\lambda)(a - t_X)$ and $\lambda(t_X - b)$ time units, respectively, and thus it is still running at time $t_X + (1+\lambda)(a - t_X) + \lambda(t_X - b) = (1+\lambda)a - \lambda b = t$.

Now, suppose that X is still running at t. Assume first that X is in normal state at t. By construction, the horizontal segment of $\mathcal{C}(X)$ passes through the point $(t_X + (t-t_X)/(1+\lambda), t_X)$. Let $a = t_X + (t-t_X)/(1+\lambda)$, and $b = t_X$, it can be verified that $(1+\lambda)a - \lambda b$ is equal to t. The other case where X is in exceptional state at t can be proved similarly. $\square$

Note that the set of points (a, b) with $t = (1+\lambda)a - \lambda b$ forms the line $\mathcal{L}_t : y = ((1+\lambda)/\lambda)x - t/\lambda$. Thus, for any stream X, X is still running at t if and only if $\mathcal{C}(X)$ intersects this timeline $\mathcal{L}_t$. Let $\mathcal{S} \cap \mathcal{L}_t$ denote the set of intersections between the crooks in $\mathcal{S}$ and $\mathcal{L}_t$. We have the following important corollary.

Corollary 5. $|\mathcal{S} \cap \mathcal{L}_t|$ *equals to* $\texttt{load}(\mathcal{S}, t)$*, the load of* $\mathcal{S}$ *at* t.

We derive an upper bound on $|\mathcal{S} \cap \mathcal{L}_t|$ in Section 5. In the rest of this section, we introduce the concept of no waste schedules, which simplifies our competitive analysis by avoiding pathological instances of representation. We say that the schedule $\mathcal{S}$ is no waste if for any stream $X \in \mathcal{S}$ that is not a full stream, it satisfies the following two properties.

- No waste in normal state: X changes to exceptional state as soon as there are no later streams merging with it. (The stream U in Figure 1 does not have this property; it should have changed to exceptional state immediately after W merges with it.)

- No waste in exceptional state: When X changes to exceptional state, it will decide to merge with the nearest stream that it can merge so as to be in the exceptional state the least amount of time. (The stream V in Figure 1 does not have this property; it should have merged with U instead of Y.)

Note that all known competitive algorithms guarantee no waste in exceptional state. Furthermore, the 2-dyadic and the α-dyadic algorithms also guarantee no waste in normal state, but this is not true for the connector algorithm and our greedy algorithm $\mathcal{G}_s$. However, by applying a technique of Coffman *et al.* [9], we can modify these two algorithms such that they construct on-line schedules that satisfy the two "no waste" properties while still maintaining the max span and min coverage bounds. The idea is roughly as follows. To be no waste in normal state, a stream X can run in exceptional state whenever possible, and if some later stream decides to merge with it, X can easily correct its mistake by discarding the extra data in its buffer and running as if it is still in normal state. We refer to [9] for more details.

Now, we prove some structural properties on no waste schedules. We say a point (t, t) is a request point if at time t there is a request. Thus, a stream is initiated at every request point. For any point $p = (x, y)$, let $\texttt{above}(p)$ and $\texttt{left}(p)$ denote the point (x, x) and (y, y), respectively. (Note that $\texttt{above}(p)$ and $\texttt{left}(p)$ are just the points on the line $y = x$ that are directly above and to the left of p, respectively.) Given any crook $\mathcal{C}$, let $h(\mathcal{C})$ and $v(\mathcal{C})$ denote respectively the horizontal and vertical segments of $\mathcal{C}$. We say that a crook $\mathcal{C}$ is *in the shadow* of another crook $\mathcal{C}'$ if $\mathcal{C}$ is lying completely above $h(\mathcal{C}')$.

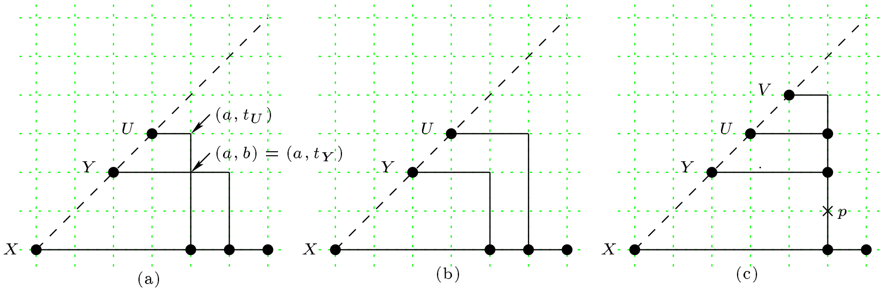

Figure 2: Proof of the properties on no waste schedule.

Lemma 6. *Suppose that the schedule $\mathcal{S}$ is no waste. Then its representation has the following properties.*

1. *No two crooks cross each other.*
2. *If a crook $\mathcal{C}$ covers a request point (t, t), then the crook for (t, t) is in the shadow of $\mathcal{C}$.*
3. *Let p be a point on some crook $\mathcal{C}$. If p is on $h(\mathcal{C})$, then* `left`(p) *is a request point. If p is on $v(\mathcal{C})$, then* `above`(p) *is a request point.*

Proof. For the first property, suppose that there are streams U and Y, initiated at t_U and t_Y, where their crooks $\mathcal{C}(U)$ and $\mathcal{C}(Y)$ cross each other. Then, the vertical segment of one of them, say $\mathcal{C}(U)$, must intersect the horizontal segment of the other, i.e., $\mathcal{C}(Y)$, at some point (a, b). See Figure 2(a). Note that the length of $v(\mathcal{C}(U))$ is greater than $t_U - b$ and thus U runs in exceptional state for more than $\lambda(t_U - b) = \lambda(t_U - t_Y)$ time units, which is the time U needs to be in exceptional state if it merges with Y instead. This implies that U is not "no waste" in exceptional state and contradicts our assumption that $\mathcal{S}$ is no waste.

Now, we prove the second property. Suppose that the crook $\mathcal{C}(Y)$ covers the request point (t, t), but the crook $\mathcal{C}(U)$ at (t, t) is not in the shadow of $\mathcal{C}(Y)$. See Figure 2(b). From the first property that we have proved above, there are no crooks that cross $h(\mathcal{C}(U))$ and terminate on $\mathcal{C}(Y)$. This implies that Y can change to exceptional state at the earlier time $t_Y + (1 + \lambda)(t_U - t_Y)$ and thus Y is not no waste in normal state; a contradiction.

For the third property, consider any point p on some crook $\mathcal{C}$. If p is on its horizontal segment $h(\mathcal{C})$, then by construction, `left`(p) is a request point. Suppose that p is on the vertical segment $v(\mathcal{C})$. If `above`(p) is not a request point, then the top crook in the stack of crooks above p is not no waste in normal state (e.g., $\mathcal{C}(V)$ in Figure 2(c)); a contradiction. □

5 The competitiveness of (c, s)-bounded schedules

In this section, we analyze the competitiveness of a (c, s)-bounded no waste schedule $\mathcal{S}$. Recall that $\texttt{load}(\mathcal{S}, t)$, the load of S at t is equal to $|\mathcal{S} \cap \mathcal{L}_t|$. In Section 5.1, we introduce a counting argument that relates $\texttt{load}(\mathcal{S}, t)$ to the load of an optimal schedule. We complete the analysis in Section 5.2.

5.1 A counting argument for bounding $|\mathcal{S} \cap \mathcal{L}_t|$

Let $\mathcal{O}$ be any schedule serving the same sequence of requests served by $\mathcal{S}$. Now, we bound $|\mathcal{S} \cap \mathcal{L}_t|$ in terms of $|\mathcal{O} \cap \mathcal{L}_t|$.

Note that the representation of both $\mathcal{S}$ and $\mathcal{O}$ has their root r at $(t_1, t_1 + \ell/(1+\lambda))$, where t_1 is the initiation time of the first stream. In our argument, we associate every point $p \in \mathcal{S} \cap \mathcal{L}_t$ with a unique point $q \in \mathcal{O} \cap \mathcal{L}_t$ as follows.

Suppose that p is on the crook $\mathcal{C}$. By Lemma 6 Property 3, we conclude that if p is on the horizontal line of $\mathcal{C}$, then $\texttt{left}(p)$ is a request point; otherwise, $\texttt{above}(p)$ is a request point. Let u be this request point. Since p is on $\mathcal{L}_t$, u and the root r are on the different sides of $\mathcal{L}_t$ and this implies that the path from u to r in the optimal schedule $\mathcal{O}$ must intersect $\mathcal{L}_t$ at some unique point q. Thus, $q \in \mathcal{O} \cap \mathcal{L}_t$. We associate p with this q.

For ease of future reference, we say that q is the *boss* of p, and p is turned to be a *slave* of q through u. Figure 3 gives an example. Now, we are ready to relate $|\mathcal{S} \cap \mathcal{L}_t|$ and $|\mathcal{O} \cap \mathcal{L}_t|$. Consider the following table $T_{\mathcal{SO}}$, in which there are $|\mathcal{S} \cap \mathcal{L}_t|$ rows and $|\mathcal{O} \cap \mathcal{L}_t|$ columns. The rows of $T_{\mathcal{SO}}$ are labeled by the points in $\mathcal{S} \cap \mathcal{L}_t$, and the columns by those in $\mathcal{O} \cap \mathcal{L}_t$. Every row has a single entry equal to 1, and all the other entries are equal to 0. More precisely, for a row p, we have

$$T_{\mathcal{SO}}[p, q] = \begin{cases} 1, & \text{if } q \text{ is the boss of } p; \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Let N be the total number of 1's in $T_{\mathcal{SO}}$. Obviously $|\mathcal{S} \cap \mathcal{L}_t| = N$. In the next section, we show that if $\mathcal{S}$ is (c, s)-bounded, we can derive an upper bound $K_{c,s}$ on the number of 1's in every column of $T_{\mathcal{SO}}$. Then, we have

$$|\mathcal{S} \cap \mathcal{L}_t| = N \leq K_{c,s} |\mathcal{O} \cap \mathcal{L}_t|. \tag{2}$$

5.2 Finding the upper bound $K_{c,s}$

Suppose that $\mathcal{S}$ is (c, s)-bounded. Consider any column q of $T_{\mathcal{SO}}$. Denoted by $\texttt{Slave}(q)$ the set of all slaves of q. By definition, the total number of 1's in column q is equal to $|\texttt{Slave}(q)|$. Below, we derive an upper bound on $|\texttt{Slave}(q)|$.

Given any two points u and v, let $\overline{uv}$ denote the line segment joining u and v, and let $\|\overline{uv}\|$ be the length of $\overline{uv}$. For the sake of simplicity, we let $q_a = \texttt{above}(q)$ and

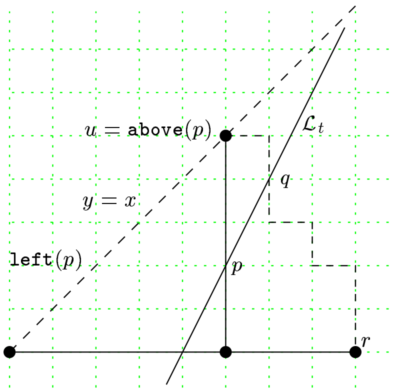

Figure 3: The position of the boss of p. The dotted line is the path in $\mathcal{O}$ from $\texttt{above}(p)$ to r.

$q_l = \texttt{left}(q)$. Let q_h be the intersection point of $\mathcal{L}_t$ and the horizontal line passing through q_a. Let q_v be the intersection point of $\mathcal{L}_t$ and the vertical line passing through q_l (see Figure 4). Below, we give two easy lemmas that are useful in our analysis.

Lemma 7. *We have* $\|\overline{q_a q}\| = \|\overline{q_l q}\|$, $\|\overline{q_a q_h}\| = \frac{\lambda}{1+\lambda}\|\overline{q_a q}\|$ *and* $\|\overline{q_l q_v}\| = \frac{1+\lambda}{\lambda}\|\overline{q_a q}\|$.

Proof. Note that q_a is on the line $y = x$, and q and q_h are on the line $\mathcal{L}_t : y = ((1+\lambda)/\lambda)x - t/\lambda$. The lemma follows from simple calculation from analytic geometry. □

Lemma 8. *Consider any point $p \in \texttt{Slave}(q)$. Suppose that p is turned to be a slave of q through the point u. Then u must lie on the segment $\overline{q_l q_a}$.*

Proof. By definition, the path π from u to the root of $\mathcal{O}$ passes through q. Since π is a right- and down-going monotonic rectilinear line, it cannot pass through q if u is outside $\overline{q_l q_a}$. □

Now, we are ready to estimate $|\texttt{Slave}(q)|$. As a warm-up, we first consider a special but important case, namely when $c = \lambda/(1+\lambda)$ and $s = 1$.

Let A be the set of points in $\texttt{Slave}(q)$ that are on or above q, and $B = \texttt{Slave}(q) - A$.

Lemma 9. *Suppose that $\mathcal{S}$ is (c, s)-bounded with $c = \lambda/(1+\lambda)$ and $s = 1$. If $|A| \geq 2$ then we can conclude $|A| = 2$ and $|B| = 0$.*

Proof. Suppose that $|A| \geq 2$. Let p be the highest point in A. Recall that p is some point in $\mathcal{S} \cap \mathcal{L}_t$, and thus must lie on a crook $\mathcal{C}$ in $\mathcal{S}$. Note that p must lie on the horizontal segment $h(\mathcal{C})$ of $\mathcal{C}$ (otherwise, p is on $v(\mathcal{C})$ and by definition, p is turned to be a slave of q through $\texttt{above}(p)$, which lies outside $\overline{q_l q_a}$; this is impossible because of Lemma 8). Thus, $\|h(\mathcal{C})\| \geq \|\overline{q_a q_h}\|$ (see Figure 4). Since $\mathcal{S}$ is (c, s)-bounded, it can be verified that $\|h(\mathcal{C})\| \leq \frac{\lambda}{1+\lambda} s \, \|v(\mathcal{C})\|$. Together with Lemma 7, we have

$$\|v(\mathcal{C})\| \geq \frac{1}{s}\frac{1+\lambda}{\lambda}\|h(\mathcal{C})\| \geq \frac{1+\lambda}{\lambda}\|\overline{q_a q_h}\| = \|\overline{q_a q}\|.$$

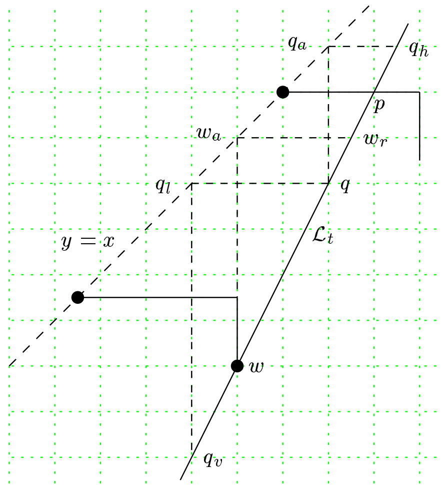

Figure 4: The position of the points in $\texttt{Slave}(q)$.

Now consider the other point $p' \in A$ that is immediately below p on $\mathcal{L}_t$. Suppose that p' is on the crook $\mathcal{C}'$ in $\mathcal{S}$. Note that $\mathcal{C}'$ covers $\texttt{left}(p)$, and since $\mathcal{S}$ is no waste, by Lemma 6 Property 2, $\mathcal{C}$ is in the shadow of $\mathcal{C}'$. It follows that $h(\mathcal{C})$ and $h(\mathcal{C}')$ must be at least $\|v(\mathcal{C})\| \geq \|\overline{q_a q}\|$ apart. This is only possible when $\texttt{left}(p)$ and $\texttt{left}(p')$ are at the two endpoints of $\overline{q_l q_a}$ (Lemma 8 asserts that they cannot lie outside $\overline{q_l q_a}$). Thus, there are no other points on $\overline{q_h q}$ that are in $\texttt{Slave}(q)$, and together with Lemma 8, we can conclude that $|A| = 2, |B| = 0$. $\square$

Lemma 10. *Suppose that $\mathcal{S}$ is (c, s)-bounded with $c = \lambda/(1+\lambda)$ and $s = 1$. If $|B| \geq 2$, then we can conclude that $|B| = 2$ and $|A| = 0$.*

Proof. Suppose that $|B| \geq 2$. Let w be the lowest point in B, and $\mathcal{C}_w$ be the crook in $\mathcal{S}$ on which w lies. Using an argument similar to the one we use in Lemma 9, we can show that $w_a = \texttt{above}(w)$ is a request point. Let $w_a = (t_{\text{before}}, t_{\text{before}})$ for some t_{before}. Let X be the stream in $\mathcal{S}$ that is initiated immediately after t_{before}. Suppose that X merges with a stream initiated at time t_{parent}. Since $\mathcal{S}$ is no waste, we have (1) all crooks above $h(\mathcal{C}_w)$ are in the shadow of $\mathcal{C}_w$ and (2) no crooks can cross $v(\mathcal{C}_w)$. This implies that $\mathcal{C}(X)$ can only terminate on some horizontal line below w, and it follows that

$$t_{\text{parent}} < t_{\text{before}} - \|\overline{w_a w}\|.$$

Let t_{miss} be the smallest time such that there is a stream initiated at t_{miss} and X does not cover t_{miss}. Since $\mathcal{S}$ is (c, s)-bounded, we have

$$t_{\text{miss}} \geq t_{\text{before}} + c(t_{\text{before}} - t_{\text{parent}}) > t_{\text{before}} + c\|\overline{w_a w}\|.$$

We claim that $t_{\text{before}} + c\|\overline{w_a w}\| \geq t_a$, where $q_a = (t_a, t_a)$. Then, we have $t_{\text{miss}} > t_a$ and thus there are only two crooks starting in the line segment $\overline{q_l q_a}$, namely $\mathcal{C}_w$ and $\mathcal{C}(X)$, that can intersect $\mathcal{L}_t$. As we assume $|B| \geq 2$, we conclude $|B| = 2$ and $|A| = 0$.

To prove our claim, let w_r be the intersection point of $\mathcal{L}_t$ and the horizontal line that passes through w_a. We have $t_{\text{before}} + c\|\overline{w_a w}\| = t_{\text{before}} + \frac{\lambda}{1+\lambda}\|\overline{w_a w}\| = t_{\text{before}} + \|\overline{w_a w_r}\| \geq t_a$. Note that the second equality follows from simple calculation from analytic geometry. $\square$

We immediately have the following theorem and corollary.

Theorem 11. *Suppose that $\mathcal{S}$ is (c, s)-bounded with $c = \lambda/(1 + \lambda)$ and $s = 1$. Then for any time t, $|\mathcal{S} \cap \mathcal{L}_t| \leq 2|\mathcal{O} \cap \mathcal{L}_t|$.*

Proof. Consider any column q of $T_{\mathcal{SO}}$. The number of 1's in this column equals $|\texttt{Slave}(q)|$, which, by Lemmas 9 and 10, is no greater than 2. Together with Inequality (2), the theorem follows. $\square$

The following corollary, which follows directly from Theorems 3 and 11, gives the first on-line algorithm for stream merging which is 2-competitive.

Corollary 12. *The on-line algorithm $\mathcal{G}_s$ with $s = 1$ is 2-competitive.*

The following theorem states our result for the general case.

Theorem 13. *Let $K_{c,s} = 1 + \max\left\{\lceil \log_{1+c} \frac{1+2\lambda}{1+\lambda} \rceil, \lceil \log_{1+\frac{1+\lambda}{s\lambda}} \frac{1+2\lambda}{\lambda} \rceil\right\}$. Suppose that $\mathcal{S}$ is (c, s)-bounded. Then we have for any time t, $|\mathcal{S} \cap \mathcal{L}_t| \leq K_{c,s}|\mathcal{O} \cap \mathcal{L}_t|$.*

Proof. The proof is a direct generalization of that of Theorem 11. We still need to bound $|A|$ and $|B|$. For $|A|$, since we have a different span factor, a crook may be shorter and thus there may be more crooks stacking above q; hence we may have a larger $|A|$. For $|B|$, if the coverage factor is different, then the crooks may be closer and thus there may be more points falling into B. However, using a very similar analysis, we can show that the worst case is when either all points fall in A or all points fall in B and $|\texttt{Slave}(q)|$ is upper bounded by the constant $K_{c,s}$.

We now explain briefly why $|\texttt{Slave}(q)|$ is upper bounded by $K_{c,s}$ for the worst case that all points fall in A. The case that all points fall in B is similar. Suppose there are r points in A, lying on crooks $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_r$ such that $\mathcal{C}_i$ is above $\mathcal{C}_{i+1}$. Using similar notations in the proof of Lemma 9, we know that $\|v(\mathcal{C}_1)\| \geq \frac{1}{s}\frac{1+\lambda}{\lambda}\|h(\mathcal{C}_1)\|$. Then $\|h(\mathcal{C}_2)\| \geq \|h(\mathcal{C}_1)\| + \|v(\mathcal{C}_1)\| \geq (1 + \frac{1}{s}\frac{1+\lambda}{\lambda})\|h(\mathcal{C}_1)\|$, and $\|v(\mathcal{C}_2)\| \geq (\frac{1}{s}\frac{1+\lambda}{\lambda})\|h(\mathcal{C}_2)\| \geq (\frac{1}{s}\frac{1+\lambda}{\lambda})(1 + \frac{1}{s}\frac{1+\lambda}{\lambda})\|h(\mathcal{C}_1)\|$. Similarly, we can show that for $1 < i \leq r$,

$$\|v(\mathcal{C}_i)\| \geq (\frac{1}{s}\frac{1+\lambda}{\lambda})(1 + \frac{1}{s}\frac{1+\lambda}{\lambda})^{i-1}\|h(\mathcal{C}_1)\|.$$

On the other hand,

$$\sum_{1 \leq i \leq r} \|v(\mathcal{C}_r)\| \leq \|\overline{q_a q}\| \leq \frac{1+\lambda}{\lambda}\|h(\mathcal{C}_1)\|.$$

Combining the two inequalities, we have $|\texttt{Slave}(q)| = r \leq \lceil \log_{1+\frac{1+\lambda}{s\lambda}} \frac{1+2\lambda}{\lambda} \rceil$. $\square$

6 Reduction from general to compact sequence

In this section, we are going to elaborate the discussion in Section 2 that if we have an on-line algorithm that performs well for compact sequences, then we can construct another on-line algorithm that performs well for general sequences. By Corollary 12, in the case of compact sequences, our greedy algorithm $\mathcal{G}_s$ always produces a schedule in which the load at any time is at most twice of the load of the optimal schedule. Then, by Lemma 1, when $\lambda = 1$, $\mathcal{G}_s$ is 2-competitive with respect to maximum bandwidth for general sequences. By Lemma 2, for arbitrary $\lambda \geq 1$, $\mathcal{G}_s$ is 3-competitive with respect to total bandwidth for general sequences. Below we will generalize Lemma 1 so that for arbitrary $\lambda \geq 1$, $\mathcal{G}_s$ is 2-competitive with respect to maximum bandwidth for general sequences (see Lemma 14). Furthermore, we will show in Lemma 16 that a special class of algorithms (including $\mathcal{G}_s$) are 2.5-competitive with respect to total bandwidth for general sequences.

First of all, we describe how to handle a given general sequence based on an on-line algorithm for compact sequence. We divide any general sequence R into maximal and non-overlapping compact sequences $C_1, C_2, \ldots, C_m$, and then find a schedule S_i for each C_i. Then the schedule for R is the union of these schedules S_i. Note that the geometric representation of S_i forms a tree. Thus, the geometric representation of the schedule for R forms a forest where all the leaves lie on the line $y = x$ and all roots lie on the line $y = x - \ell/(1+\lambda)$. The following lemma is a generalization of Lemma 1 for arbitrary $\lambda \geq 1$.

Lemma 14. *Consider any $\lambda \geq 1$. Let c be a positive number. Suppose that A is an on-line stream merging algorithm such that given any compact sequence C, A always produces a schedule $\mathcal{S}$ for C with $\texttt{load}(\mathcal{S}, t) \leq c\,\texttt{load}(T, t)$ for any schedule T for C and any time t. Then, we can construct from A a stream merging algorithm that is c-competitive with respect to maximum bandwidth for any general request sequence.*

Proof. Before getting into the analysis, we give some definition. Consider any compact sequence $C = (t_1, t_2, \ldots, t_k)$. Note that the geometric representation of a schedule for C forms a tree in which all the leaves are on the line $y = x$ and its root is at $(t_1 + \ell/(1+\lambda), t_1)$. In other words, the representation must lie on the triangle $\triangle$ bounded by the three points (t_1, t_1), $(t_1 + \ell/(1+\lambda), t_1)$ and $(t_1 + \ell/(1+\lambda), t_1 + \ell/(1+\lambda))$. We call this triangle $\triangle$ the *bounding triangle* of C.

Let R be any general sequence. Suppose that R can be divided into m maximal and non-overlapping compact sequences C_1, C_2, $\ldots$, C_m. Let S_i be the schedule (and its representation) constructed by the algorithm A for C_i, and P be the union of these S_i. Let O be an optimal schedule for R. First, we associate O with a set of m schedules, one for each compact sequence and then we use these schedules to relate the load of P and O. Let $\triangle_i$ be the bounding triangle of C_i. Denote by Z_i the portion of O that lies in the interior of $\triangle_i$. Let T_i be the tree comprising the lines in Z_i together with the bottom and right boundary of $\triangle_i$. Observe that T_i corresponds to a schedule for C_i, and $\texttt{load}(T_i, t) \leq \texttt{load}(Z_i, t) + 2$. Now, we relate $\texttt{load}(O, t)$ and $\texttt{load}(S_i, t)$ through these T_i.

From the assumption of the lemma, we have $\mathtt{load}(S_i, t) \leq c\,\mathtt{load}(T_i, t)$. We claim that there are at most $(1+\lambda)$ T_i's, say $T_{i_1}, T_{i_2}, \ldots, T_{i_{1+\lambda}}$, such that $\mathtt{load}(T_i, t) \neq 0$. Then, $\mathtt{load}(P, t) = \sum_{1 \leq i \leq m} \mathtt{load}(S_i, t) \leq c \sum_{1 \leq i \leq m} \mathtt{load}(T_i, t) \leq c \sum_{1 \leq j \leq 1+\lambda} (\mathtt{load}(Z_{i_j}, t) + 2) \leq c\mathtt{load}(O, t) + 2c(1+\lambda)$, and the lemma follows.

To prove the claim, consider the smallest i such that $\mathtt{load}(T_i, t) \neq 0$. Suppose that the first request in C_i arrives at t_i. Note that all streams scheduled by T_i terminate before $t_i + \ell$ and to contribute some load at t, we must have $t \leq t_i + \ell$. On the other hand, since $C_1, C_2, \ldots, C_m$ are maximal and non-overlapping compact sequences, the arrival times of the first request of any two consecutive C_j and C_{j+1} differ by more than $\ell/(1+\lambda)$. This implies that for all $j \geq i + 1 + \lambda$, all streams scheduled by T_j run after $t_i + \ell$, and thus after t. Hence, only $T_i, T_{i+1}, \ldots, T_{i+\lambda}$ can have positive load at t. □

Corollary 15. *The on-line algorithm $\mathcal{G}_s$ with $s = 1$ is 2-competitive with respect to maximum bandwidth for general sequences.*

Recall that $K_{c,s}$ is the competitive ratio of an algorithm which always produces (c, s)-bounded schedules in the case of compact sequences. The next lemma is an improvement of Lemma 2 with respect to total bandwidth.

Lemma 16. *Suppose that A is an on-line stream merging algorithm such that given any compact sequence C, A always produces a (c, s)-bounded schedule for C with $s \leq (1+\lambda)$. Then, we can construct from A a stream merging algorithm that is $(K_{c,s} + 1/(1+\lambda))$-competitive with respect to total bandwidth for any general request sequence.*

Proof. Suppose that the general request sequence $\mathcal{R}$ is divided into m maximal and non-overlapping compact sequences $C_1, C_2, \ldots, C_m$. Let S_i be the (c, s)-bounded schedule constructed by the algorithm A for C_i. Let $\mathcal{S} = \bigcup_{i=1}^{m} S_i$. Let $\mathcal{O}$ be an optimal schedule for $\mathcal{R}$. For any time t, the table $T_{\mathcal{SO}}$ is constructed in the same way as the case for compact sequence. However, in this case it is not necessary that all rows have a single entry equal to 1. There may be some rows that have all entries equal to 0. Let W_t^0 denote the set of rows in $T_{\mathcal{SO}}$ that have all entries equal to 0 and W_t^1 denote the set of rows in $T_{\mathcal{SO}}$ that have a single entry equal to 1. A row in W_t^0 corresponds to the case that $\mathcal{L}_t$ intersects some point p in $\mathcal{S}$ but $\mathcal{L}_t$ does not intersect any point in the path from $u = \{\mathtt{above}(p), \mathtt{left}(p)\}$ in $\mathcal{O}$. Note that $W_t^0 \cup W_t^1 = \mathcal{S} \cap \mathcal{L}_t$. Theorem 13 can be rephrased to $|W_t^1| \leq K_{c,s}|\mathcal{O} \cap \mathcal{L}_t|$ for any time t. We thus obtain the following inequality.

$$\sum_t |W_t^1| \leq K_{c,s} \cdot (\text{total bandwidth used in } \mathcal{O}). \tag{3}$$

We claim that

$$\sum_t |W_t^0| \leq \frac{1}{1+\lambda} \cdot (\text{total bandwidth used in } \mathcal{O}). \tag{4}$$

With Inequalities (3) and (4), we obtain the stated competitive ratio of $(K_{c,s} + 1/(1+\lambda))$.

We prove the claim of Inequality (4). The case that there is a row in W_t^0 having all entries 0 must happen at time $\lambda\ell/(1+\lambda)$ units after the request arrives at u because even if the path from u in $\mathcal{O}$ is composed of only exceptional state, it runs for at least $\lambda\ell/(1+\lambda)$ time units. In $\mathcal{S}$, only a full stream runs for more than $\lambda\ell/(1+\lambda)$ time units in normal state. For any other stream in $\mathcal{S}$, if the stream runs in normal state for more than $\lambda\ell/(1+\lambda)$ time units, with span factor less than or equal to $(1+\lambda)$, the stream must also run in exceptional state for more than $\lambda\ell/(1+\lambda)^2$ time units. Hence, the total number of video units received will be more than ℓ, which is a contradiction. For each of the full streams in $\mathcal{S}$, suppose that v is the request point, the case that there is a row in W_t^0 having all entries 0 only happens at time $\lambda\ell/(1+\lambda)$ unit after the request arrives at v. Hence, the full stream contributes at most $\ell - \lambda\ell/(1+\lambda)$ points in $\bigcup_t W_t^0$. Since $\mathcal{R}$ consists of m compact sequences, there are exactly m full streams in $\mathcal{S}$. Therefore, $\sum_t |W_t^0| \leq m(\ell - \lambda\ell/(1+\lambda)) = m\ell/(1+\lambda)$. As $\mathcal{R}$ consists of m compact sequences, $\mathcal{O}$ must consist of at least m full streams and hence the total bandwidth of $\mathcal{O}$ is at least $m\ell$. As a result, we have $\sum_t |W_t^0| \leq 1/(1+\lambda)$ times the total bandwidth of $\mathcal{O}$. □

Corollary 17. *The on-line algorithm $\mathcal{G}_s$ with $s = 1$ is 2.5-competitive with respect to total bandwidth for general sequence.*

Proof. Notice that $K_{c,s} = 2$ for $\mathcal{G}_s$ with $s = 1$; and $1/(1+\lambda) \leq 1/2$. Thus, by Lemma 16, the corollary follows. □

References

[1] C. C. Aggarwal, J. L. Wolf, and P. S. Yu. On optimal piggyback merging policies for video-on-demand systems. In *Proceedings of ACM Sigmetrics Conference*, pages 200–209, 1996.

[2] A. Bar-Noy, J. Goshi, R. E. Ladner, and K. Tam. Comparison of stream merging algorithms for media-on-demand. In *Proceedings of Conference on Multimedia Computing and Networking*, pages 115–129, 2002.

[3] A. Bar-Noy and R. E. Ladner. Competitive on-line stream merging algorithms for media-on-demand. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 364–373, 2001.

[4] Y. Cai, K. A. Hua, and K. Vu. Optimizing patching performance. In *Proceedings of Conference on Multimedia Computing and Networking*, pages 204–215, 1999.

[5] S. W. Carter and D. D. E. Long. Improving bandwidth efficiency of video-on-demand. *Computer Networks*, 31(1-2):99–111, 1999.

[6] W. T. Chan, T. W. Lam, H. F. Ting, and P. W. H. Wong. Competitive analysis of on-line stream merging algorithms. In *Proceedings of the Twenty-Seventh Annual International Symposium on Mathematical Foundations of Computer Science*, pages 188–200, 2002.

[7] W. T. Chan, T. W. Lam, H. F. Ting, and P. W. H. Wong. A unified analysis of hot video schedulers. In *Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing*, pages 179–188, 2002.

[8] W. T. Chan, T. W. Lam, H. F. Ting, and P. W. H. Wong. On-line stream merging in a general setting. *Theoretical Computer Science*, 296(1):27–46, 2003.

[9] E. Coffman, P. Jelenkovic, and P. Momcilovic. The dyadic stream merging algorithm. *Journal of Algorithms*, 43(1):120–137, 2002.

[10] D. Eager, M. Vernon, and J. Zahorjan. Bandwidth skimming: A technique for cost-effective video-on-demand. In *Proceedings of Conference on Multimedia Computing and Networking*, pages 206–215, 2000.

[11] D. Eager, M. Vernon, and J. Zahorjan. Minimizing bandwidth requirements for on-demand data delivery. *IEEE Transactions on Knowledge and Data Engineering*, 13(5):742–757, 2001.

[12] L. Golubchik, J. C. S. Lui, and R. R. Muntz. Adaptive piggybacking: A novel technique for data sharing in video-on-demand storage servers. *ACM Journal of Multimedia Systems*, 4(3):140–155, 1996.

[13] K. A. Hua, Y. Cai, and S. Sheu. Patching: A multicast technique for true video-on-demand services. In *Proceedings of the Sixth ACM International Conference on Multimedia*, pages 191–200, 1998.

[14] S. W. Lau, J. C. S. Lui, and L. Golubchik. Merging video streams in a multimedia storage server: Complexity and heuristics. *ACM Journal of Multimedia Systems*, 6(1):29–42, 1998.

[15] S. Sen, L. Gao, J. Rexford, and D. Towsley. Optimal patching schemes for efficient multimedia streaming. In *Proceedings of the Ninth International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 44–55, 1999.

[16] P. W. H. Wong. *On-line Scheduling of Video Streams*. PhD thesis, The University of Hong Kong, 2002.