On the Power of Cooperation in Multimedia Caching

Itai Dabran and Danny Raz

Computer Science Department, Technion, Haifa 32000, Israel

Abstract. Real time multimedia applications such as Internet TV, Video On Demand, Distance Learning, and Video Conferencing are becoming more and more popular over the Internet. Streaming media caching is a critical ingredient in the ability to provide scalable real-time service over the best effort Internet. In many cases, bandwidth becomes the system bottleneck and the cache cannot provide the required quality for all streams simultaneously. In this paper we study new algorithms, based on cooperation, which can improve the cache ability to provide service to all of its clients. The main idea is based on the willingness of streams to reduce their used bandwidth and allow other streams that may need it more, to use it. Our extensive simulation study indicates that our algorithms can reduce the pre-caching time by a factor of 3, or increase the probability for adequate service level by 30% using the same pre-caching time.

1 Introduction

A web proxy cache sits between web servers and clients, and stores frequently accessed web objects. The proxy receives requests from the clients and when possible, serves them using the stored objects. When considering streaming media, caching proxies become a critical component in providing scalable real-time services to the end users. Due to the bursty nature of streaming media amplified by compression, providing high quality service requires a considerable amount of bandwidth that may not be always available in the best effort Internet. The main problems that arise in this context are the large size of media items and their variable bit-rate nature: bandwidth consumed by the user is measured by frame per second (frame rate below 24 frames/second is distracting to the human eye) but the size of the frame is variable according to the multimedia visualization and compression method.

Figure 1 depicts an architecture of video delivery from video servers over the Internet to clients connected to a cluster of multimedia cache proxies, over a high speed LAN. The proxy may experience data losses, delays and jitter, while it has to provide the multimedia streams to its customers at high bandwidth and low delay. This is done, in many cases, by keeping a buffer containing several seconds of the stream content per each stream served by the cache. In times when the amount of data needed in the stream increases, or when the available bandwidth reduces due to competitive streams or other traffic load, the cache

still delivers the stream to the client, reducing temporary the amount of buffered data. In current streaming media cache proxies, each stream is handled indepen-

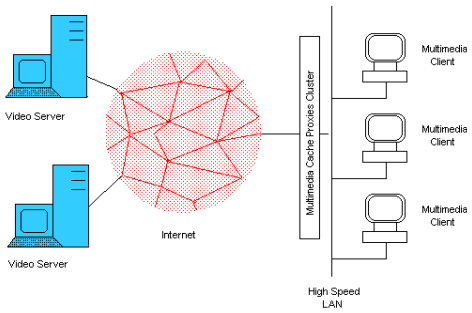

Fig. 1. Video Delivery Architecture

dently, as a separate object. That is, the server opens a TCP connection to the streaming media server, checks the buffer status and possibly other local parameters and then when needed, asks for more streaming data from the server. The downloading rate of the data depends, in addition to the cache requests, on the networking conditions (i.e. available bandwidth and load) and TCP behavior. When a cache proxies' cluster deals with more than one stream, the flows of the different streams may compete on the available bandwidth and they may experience losses. In such a case the TCP mechanism will reduce the flow rate and the amount of data in the flow's buffer may decrease below the ability to serve the client in the desired quality. Note that TCP is often used for media uploading both because unlike UDP it has a congestion control mechanism, and since it is much more resistant to Firewall related blockages.

The main novel idea of this work is to introduce cooperation between the flow management of different streams, all served by the same cache proxy or cache proxies' cluster. When congestion is detected, or when the amount of the buffer's data is decreased, the cache proxy can check the status of other flows and if their buffers are adequately full, reduce their flow by manipulating the TCP flow control. In this way, in scenarios similar to the architecture mentioned before, the load on the bottleneck link from the Internet to the cache proxy is reduced, enabling the needed flow rate to increase by using the TCP normal flow control mechanism. This approach is different from other schemes that are in use today and use the term cooperation. A classic caching cluster may hold multiple copies of the same movie, and cooperate in order to fetch the movie from a neighbor cache instead of fetching it from the server (using ICP [14] for example). On Video On Demand (VOD) systems when more than one client wants to watch the same movie in another time interval, cooperation is used in order to "patch" the needed multimedia stream for the client that joins later.

In this paper we use the general term Multimedia Caching Cooperative Device (MCCD) for a streaming media cache proxy or cache proxies' cluster. In our

scheme, when the MCCD shares the same Internet connection, it can optimize the buffer usage in order to increase the scalability of the streaming media, by controling the rate of the incoming streams. The devices in a cluster can exchange information about the storage capacity (or a lack of input bandwidth) in order to enable one device to decrease its rate when it can, and by this to increase the bandwidth towards another one in order to overcome a buffer shortage. The same idea could be used in 3G/4G cellular networks where again, multiple streams compete on limited bandwidth. However in cellular networks, the bandwidth bottleneck is between the cache proxy and the clients that use the cellular devices. In these scenarios, achieving flow control cooperation is more difficult, yet the benefit of cooperation remains very high. We present two algorithms that manipulate the TCP flow control in order to reduce the load on the bottleneck link when needed. The first one, "Lower Threshold Algorithm" (LTA), reduces the TCP receiver window size when the cache buffer's size is decreased below some predefined threshold. The second, "Upper Threshold Algorithm" (UTA) reduces the TCP receiver window size of each connection when its cache buffer's size increases above a certain predefined threshold and by this smoothes the flow control of one recipient. Our extensive simulation study indicates that using our algorithms can reduce the pre-caching time by a factor of 3, or increase the probability for adequate service level by 30% using the same pre-caching time. We also address an implementation approach, an algorithm operated from the application layer, since the TCP Layer generally does not have an application interface for controlling the TCP flow or congestion control.

This paper is organized as follows. Section 2 describes related work used in multimedia caching in order to efficiently cache and use web items. Section 3 introduces the algorithms used in order to cooperate between the MCCD proxies. Section 4 shows the evaluation and the advantages of the proposed scheme, and Section 5 describes a practical approach that can be easily adopted in order to implement our idea. Finally, Section 6 presents our conclusions.

2 Related Work

In order to avoid the mass storage location needed in order to cache multimedia streaming objects, proxies use the fact that it is not necessary to cache all the streaming media. Streaming media nature allows them to retrieve new frames during the time that the user retrieves the cached frames. In [11], a "Prefix Caching" algorithm that caches an initial portion of the streaming media is proposed. The proxy can deliver this "prefix" to the client upon a request and meanwhile to cache the remaining portion of the media. The idea of "Prefix Caching" mechanism consists of caching a group of consecutive frame at the beginning of the stream in order to smooth and reduce the variable bit-rate of the streaming media. Another algorithm, the "Video Staging" algorithm is proposed in [15]. In this algorithm, the proxy caches a portion of bits from all the frames whose size is above some predefined threshold, and uses them in order to smooth bursts when such a frame is transmitted to the user. Efficient line

bandwidth consumption is handled by several caching strategies designed for multimedia caching. Some partial caching strategies are described in [7] [8] and [15] whereas only certain parts of the multimedia stream are cached.

In order to enable a proxy to control the portion of the cached media the authors of [8] propose a "frame wise" selection algorithm in an architecture similar to the one mentioned in Section 1 whereas a proxy cluster is connected to a video server over the Internet, and to its clients over a high speed LAN. This algorithm, "Selective caching for Best effort Networks" (SCB), iteratively selects frames located between the closest buffer peak t_{max} and the risky time t_r afterwards and caches them according to the buffer-space constraints. This algorithm prevents troughs in the proxy buffers and has a better robustness than "Prefix Caching" methods under the same buffer space limitations. In [9, 10] it is proposed to cache more layers of popular videos and by this to overcome congestions that may appear later on the network. The authors of [12] propose few schemes aimed at minimizing the bandwidth needed from the origin server. Each of these schemes is applicable in different scenarios depending on the bandwidth, cache-space tradeoffs and service requirements. For example, partial caching of "Patch" and regular channel is done by re-use of buffers allocated by each "Patching-Window" interval for subsequent intervals.

Video On Demand (VOD) Multimedia stream can be received over multiple channels. A technique mentioned in [5] proposes to use two channels, the first for the complete streaming media, and the second for "patching" the data when a client joins later on to the streaming media. The client receives both channels, caches the data from the first channel, and uses it after playing the data received from the "patch" channel. In [5] it is shown that the "Patching" scheme supports much higher requests rate for VOD.

Cache clustering is a natural way to scale as traffic increases. Different schemes are used in order to arrange such a cluster, for example, in the *loosely-coupled* scheme proposed in [3], each proxy in the cluster is able to serve every request, independently of the other proxies. Popular protocols for this scheme are ICP (Internet Cache Protocol) [14] and HTCP (Hyper Text Caching Protocol) [13]. By using these protocols, proxies can share cacheable content, and increase the end user performance and the availability of bandwidth towards the server. In [1], "MiddleMan" - cooperating video proxy servers connected by a LAN, and organized by "Coordinators" is presented. A "Coordinator" process keeps track of the files hosted by each proxy and redirects requests accordingly. It was found that when more proxies are used, smaller peak loads in each proxy are created and better load-balancing between them is achieved. Since in a cluster, there is no need to cache the same streaming multimedia object in different copies for more than one user, the cluster should be able to maintain a management of a single copy inside it. Such a solution "The Last-Copy approach" for maintaining one copy of web items in a proxies' cluster in mentioned in [2], where an inter-cluster scheme is used in order to share the proxies' resources, and by this to provide scalable service to the cluster users.

3 Cooperative Protocols

Streaming Multimedia is used today for Internet-TV, Video On Demand, Distance Learning and many other applications. Suppose that MCCD (a cluster of proxies managed by a director for example) who serves several clients via a fast LAN, is connected to the Internet through a broadband connection. In order to overcome the delay and jitter caused by the Internet nature it may use any of the algorithms described before, such as the "Prefix Caching" algorithm [11], or the "Video Staging" algorithm [15]. Each Multimedia session is based on a TCP connection to a video server and has a predefined buffer space. Since the bandwidth of the connection from the MCCD to the Internet is bounded, we focus on balancing of the streaming media bandwidth between the different flows. Suppose a Home-MCCD retrieves several movies for family members, each from a different video server, via the Internet. If the buffer size of one of the proxies becomes empty, the movie transmission towards its specific user fails. Thus, when the content of a buffer decreases below a given threshold, more bandwidth is needed in order to keep the stream alive. Lowering the rate towards other video servers enables more available bandwidth and the TCP mechanism increases the rate, filling faster the buffer that is under risk. In order to implement such a cooperation, the MCCD needs to maintain a database of its buffer status for each multimedia stream. In this paper we focus in the case of a Home-MCCD combined of two proxies' processes, that retrieves two movies, m_i and m_j for family members. In order to rescue a movie whose buffer content is too small, we used two different algorithms that control the TCP receiver window. By reducing the receiver window of the other movie, the server is requested to reduce the input rate, enabling more bandwidth towards for the other connection. In the first algorithm "Lower Threshold Algorithm" (LTA), we check each time if the buffer size of each movie is below some predefined threshold. If it is, we reduce the TCP receiver window size of the second link to half of its value, and then adjust the threshold to half of its original value in order to keep decreasing the TCP receiver window if the buffer size continues to drop down. The pseudo

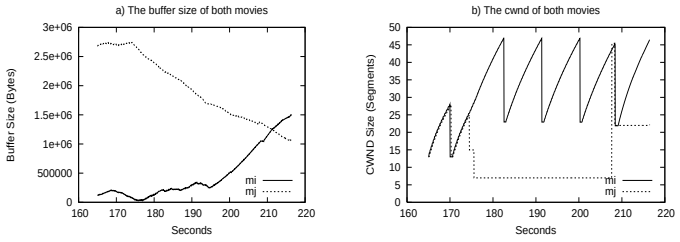

Fig. 2. The behavior of the LTA scheme

code of the "Lower Threshold" algorithm is presented in Algorithm 1. We keep on decreasing the receiver window of the other connection and the temporary

threshold, until the problematic buffer recovers. When we recognize such a recovery (lines 24 or 31 of the algorithm) we reset all the parameters and go back to normal mode. Figure 2 shows how this algorithm behaves. Figure 2(a) depicts the amount of data in the buffers and Figure 2(b) depicts the behavior of the TCP cwnd, on the same time interval. Typically, before it drops, the cwnd of both movies is in the congestion control section of 15 to 28 segments. As can be seen when the buffer size of m_i drops below some threshold (and we can see that the buffer of m_j remains high) then the cwnd of m_j starts to be bounded by its receiver window, and drops down to the value of 7 segments. By this, the bandwidth towards the first connection increases, and its cwnd changes up to 45 segment, filling quick its buffer until it increases above the threshold. The threshold taken should be optimized according to the input bandwidth and to the caching algorithm in use. If enough bandwidth is available then the threshold should be high, since the buffer of one movie can be filled quick when we increase back its receiver window, and we can reduce the risk of a failure of the other one when its buffer will drop below zero. The caching algorithm also influence the decision about the threshold size. If the "Prefix-Caching" algorithm is used for example, then frame-bursts may appear later and the threshold should be high enough in order to prevent it. If a caching algorithm that smoothes the frame-bursts is used, then the threshold can be lower. However, when there is a limited amount of bandwidth, if the threshold is too low when both buffers are mostly empty and at least one of them is still below the threshold, then the other one may slow the connection too early without enough data to overcome this action. In contrary if the threshold is too high, then the algorithm may slow the connection of one movie too many times, increasing its failure risk. In the second algorithm "Upper Threshold Algorithm (UTA) we limit the buffers capacity by using a pre-defined upper threshold for the buffers. This prevents each of the connections to consume more bandwidth than needed. This solution has some disadvantage, since some of the available bandwidth may not be consumed. A pseudo code of this algorithm is presented in Algorithm 2.

4 Performance Evaluation

We evaluate the performance of our proposed scheme, by simulating it using NS (Network Simulator) [6] over a configuration consists of a home broadband network with an MCCD that uses two proxies' processes. In our first experiment we used the "Prefix Caching" algorithm proposed in [11] and in the second one a variant of the "Video Staging" algorithm proposed in [15]. In both cases we checked how each of our protocols, the UTA and the LTA, affects the MCCD success rate (i.e. full delivery of both movies). The trace we used was taken from [4]. It consists of 174,136 integers representing the number of bits per video frame over about two hours of the Star-Wars movie MPEG1 decoding. The inter-frame time of the StarWars trace is 1/24 sec. That means that the inter-arrival time for the arriving bits is 1/24 seconds. We split the frames of the first two hours (172,800 frames) into 8 short movies of 15 minutes each. Our experiments were

Algorithm 1 "Lower-Threshold Algorithm"

$pthold \leftarrow$ predefined threshold parameter
$ReceiverWindow_i \leftarrow$ the size of the the TCP receiver window of m_i
$ReceiverWindow_j \leftarrow$ the size of the the TCP receiver window of m_j
For each incoming frame do the following:
$s_{i(t)} \leftarrow$ the current size of the buffer of m_i at time t
$s_{j(t)} \leftarrow$ the current size of the buffer of m_j at time t
Normal-State:
if $s_{i(t)} < s_{j(t)}$ and $s_{i(t)} < pthold$ **then**
  $OldReceiverWindow_j \leftarrow ReceiverWindow_j$
  $ReceiverWindow_j \leftarrow ReceiverWindow_j/2$
  $thold_i \leftarrow pthold/2$
  Change into Recovery-State
end if
if $s_{j(t)} < s_{i(t)}$ and $s_{j(t)} < pthold$ **then**
  $OldReceiverWindow_i \leftarrow ReceiverWindow_i$
  $ReceiverWindow_i \leftarrow ReceiverWindow_i/2$
  $thold_j \leftarrow pthold/2$
  Change into Recovery-State
end if
Recovery-State:
if $s_{i(t)} < s_{j(t)}$ and $s_{i(t)} < thold_i$ **then**
  $ReceiverWindow_j \leftarrow ReceiverWindow_j/2$
  $thold_i \leftarrow thold_i/2$
else if $s_{i(t)} > pthold$ **then**
  $ReceiverWindow_j \leftarrow OldReceiverWindow_j$
  Change into Normal-State
end if
if $s_{j(t)} < s_{i(t)}$ and $s_{j(t)} < thold_j$ **then**
  $ReceiverWindow_i \leftarrow ReceiverWindow_i/2$
  $thold_j \leftarrow thold_j/2$
else if $s_{j(t)} > pthold$ **then**
  $ReceiverWindow_i \leftarrow OldReceiverWindow_i$
  Change into Normal-State
end if

done with all the possible 28 pairs that can be created from these 8 movies. These movies have a very bursty nature. The maximum frame size in this trace is 185267 bits, and the average one is 15611 bits. Hence, in order to to be able to pass the maximum frame (each $1/24 seconds$), the needed bandwidth is 4.4Mbps, but the average bandwidth needed for all the movies is only 374Kbps, more than 12 times less than the maximum. In our most bursty movie derived from this trace the ratio between the peak bandwidth to the average one is 10.7, while in our moderate one, this ratio is 6.7.

We used a bandwidth interval between 600Kbps to 1.2Mbps for both MCCDs. The lower threshold used in our LTA protocol was 1.2M bytes when the "Prefix-Caching" algorithm was used, and $\alpha * _{Input-Bandwidth}$ when the "Video-Staging" algorithm was used. We used $\alpha = 6.4$ (a threshold of 6.4Mbit for 1Mbps for example). We decreased the lower threshold in this case since the "Video-Staging" algorithm smooths and reduces the variable bit-rate of the future frame-bursts, thus less overhead should be taken. In our UTA experiments we used 2M bytes as our upper threshold. The "Prefix Caching" algorithm was implemented over the first 15 seconds, whereas the prefixes of both movies were cached before start consuming them from their buffers. The "Video Staging" algorithm was implemented for frames larger than 60K bits. The total average size of the portion of

Algorithm 2 "Upper-Threshold Algorithm"

1: $ReceiverWindow_i \leftarrow$ the size of the the TCP receiver window of m_i
2: $ReceiverWindow_j \leftarrow$ the size of the the TCP receiver window of m_j
3: $OldReceiverWindow_i \leftarrow ReceiverWindow_i$
4: $OldReceiverWindow_j \leftarrow ReceiverWindow_j$
5: $pthold \leftarrow predefinedthresholdparameter$
6: For each incoming frame do the following:
7: $s_{i(t)} \leftarrow$ the current size of the buffer of m_i at time t
8: $s_{j(t)} \leftarrow$ the current size of the buffer of m_j at time t
9: **if** $s_{i(t)} > pthold$ **then**
10: $ReceiverWindow_i \leftarrow ReceiverWindow_i/2$
11: **else**
12: $ReceiverWindow_i \leftarrow OldReceiverWindow_i$
13: **end if**
14: **if** $s_{j(t)} > pthold$ **then**
15: $ReceiverWindow_j \leftarrow ReceiverWindow_j/2$
16: **else**
17: $ReceiverWindow_j \leftarrow OldReceiverWindow_j$
18: **end if**

bits of frames larger than 60K bits, in our movies is 16Mbit. Thus, in order to pre-fetch it we need to consumes an average of 16 seconds for the staging phase in 1Mbps input rate. One condition was added to this algorithm: we assume that the cached "frame peaks" are accumulated in a different buffer. We check the influence of our schemes over the run-time buffer of the input stream.

We will use the terms m_a and m_b for the movies that are under consideration. On each case there is a "weak" movie m_a that may crash as a result of bursty frames interval, and a "strong" movie m_b that decreases its connection rate towards the server, in order to enable m_a to continue, using one of our schemes. We start with checking the overall validity of the scheme. Figure 3 shows the behavior of m_a and m_b when no algorithm is in use and when the LTA and the UTA algorithms are used. When no algorithm is used, after only 90 seconds the buffer of m_a gets empty, and the streaming media fails. However, when we use the LTA algorithm, the TCP receiver-window of m_b connection is decreased when the amount of data in m_a buffer drops below the predefined threshold. Then, the buffer of m_a does not get empty, and the streaming media continues with no interruptions. When the UTA algorithm is used, both m_a buffer and m_b buffer cannot increase beyond the upper threshold, and when one of them does, its TCP receiver-window is decreased in order to balance the transfer rate of the shared connection between the two connections. Again, we see that m_a does not crash, simply because m_b is not using all the available bandwidth, and by this enable m_a to compensate the buffer temporary shortage. Figure 4 depicts the improvement of the success to view both m_a and m_b, when the LTA algorithm is operated. These results are obtained when using the "Prefix Caching" scheme and the "Video Staging" scheme. We can see for example, that when the bandwidth varies between the ranges of 750Kbps to 850Kbps the LTA algorithm enables between 25% to 57% of the movie pairs to succeed, while without it there is only 10% to 35% pairs success. When the bandwidth is above 1050Kbps using the LTA algorithm causes all movies to be successful, while without it only 75% of the pairs are successful.

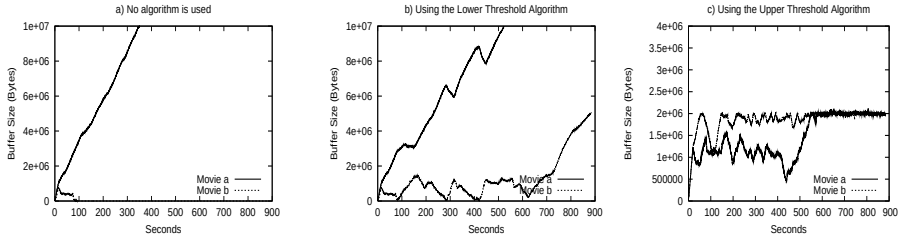

Fig. 3. Two movies behavior

Figure 4(b) shows that the "Video Staging" scheme improves the success of both movies more than the "Prefix Caching" one, but still, our LTA algorithm gets 20% to 30% better results than when no algorithm is used.

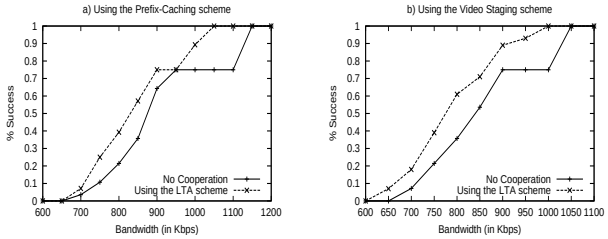

Fig. 4. The success of two movies when using the LTA Algorithm

5 A Practical Approach

Since the TCP Layer generally does not have an application interface for controlling the TCP flow or Congestion-control, implementing our algorithm in practice may be problematic. A possible solution is to temporary stop the TCP connection of one movie. However, this is not a smooth process, due to the TCP slow-start algorithm used in order to restart. Since bursts in a movie data do not last long, we can assume that time periods in which the buffer content decreases rapidly are short. We also assume that we should deal only with the case that during this time interval, only one movie is problematic, since if both of them are, we cannot do much. This discussion leads to the conclusion that stopping another TCP connection for a short time interval may solve the problem. When using a short time inteval we can use a "risky" lower threshold for m_a, lower than the one we used in the "Lower Threshold" algorithm. This reduces the possibility that m_b will crash also, since we fill its buffer more time. The "Disconnect by Lower Threshold Algorithm" (DLTA) is presented in Algorithm 3.

Algorithm 3 "Disconnect by Lower Threshold Algorithm"

```
1: pthold ← predefinedthresholdparameter
2: For each incoming frame do the following:
3: s_i(t) ← the current size of the buffer of m_i at time t
4: s_j(t) ← the current size of the buffer of m_j at time t
5: if s_i(t) < pthold then
6:    if m_i connection is active then
7:       stop the connection of m_j
8:    end if
9: else
10:   if m_j connection is stopped then
11:      start the connection of m_j
12:   end if
13: end if
14: if s_j(t) < pthold then
15:   if m_j connection is active then
16:      stop the connection of m_i
17:   end if
18: else
19:   if m_i connection is stopped then
20:      start the connection of m_i
21:   end if
22: end if
```

Note that we never stop both connections, and the buffer size of the stopped connection can drop temporary below the predefined threshold.

We evaluated the advantages of the proposed scheme using the same simulation and input as before. The buffer size threshold we used was a function of the input bandwidth, the caching time (in the "Prefix Caching" scheme), and the caching scheme in use (since we need a lower threshold with the "Video Staging" scheme). For example, we used a threshold of 600K bytes when the input bandwidth was 1Mbps, and the "Prefix Caching" time was 15 seconds. Figure 5 depicts the success rate when the DLTA algorithm is used. Figure 5(a) results are obtained when the "Prefix Caching" scheme was used, and Figure 5(b) results are obtained using the "Video Staging" scheme. In figure 5(a) we can see for example, that when the marginal bandwidth for both movies is 800 Kbps, only in 21% of the cases we succeed to see both m_a and m_b when there is no cooperation, while when the DLTA algorithm is used, we more than double it to 53% of the time. The same can be seen in Figure 5(b), whereas the "Video Staging" scheme improve the success rate over the "Prefix Caching" algorithm, but our scheme still gets about 30% better. Figure 6 depicts the improvement with respect to the no-cooperation case of all our proposed algorithms. We more than double the success rate when the link has a limited bandwidth, whereas in the lower bandwidth interval (below 800Kbps) the DLTA algorithm even triple this success. Another interesting result is the overall percentage of success to see movies with and without our scheme. As depicted in Figure 7, the DLTA schemes improves meaningfully the number of movies that can be seen via the MCCD for both caching algorithm while the other schemes do not. The main reason for this is that in a bandwidth shortage when in all other schemes both movies crash, the DLTA algorithm disables one of the connections, causes this movie to crash, but increases the bandwidth of the other, and let it to be finished.

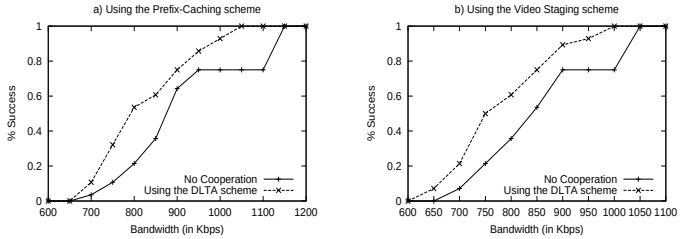

Fig. 5. The success of two movies when using the DLTA Algorithm

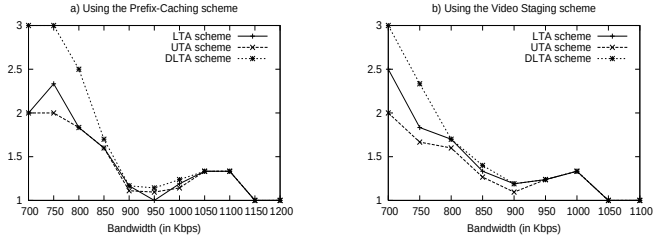

Fig. 6. The Improvement of the success of two movies

A different way to evaluate our schemes is to compare the pre-caching time needed in order to be able to deliver both movies to the clients in all 28 cases. Figure 8 compares the needed pre-caching time in our schemes against the case where no cooperation is used. It can be seen that the caching time needed in order to achieve 100% success is 90 seconds when using input bandwidth of 1Mbps, and it is decreased to 5 second when the input bandwidth is 1200Kbps. However, when using our LTA scheme, it takes only 35 seconds, less than 34%, when the input bandwidth is 1Mbps. The DLTA scheme also shows a great improvement whereas the pre-caching time that guaranties success when the input bandwidth is 1Mbps is only 40 seconds and it drops to 15 seconds instead of 60 seconds when the input bandwidth is 1050Kbps.

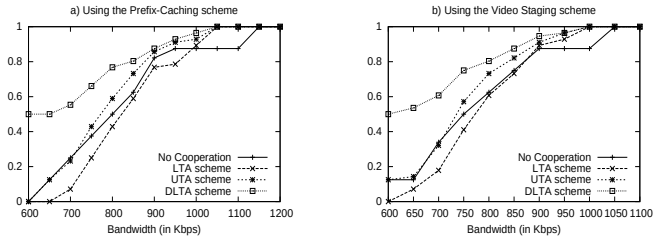

Fig. 7. The Improvement of movies success

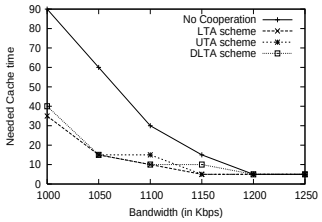

Fig. 8. Pre-Caching time in order to achieve 100% success for both movies

6 Conclusions

In this paper we proposed an efficient scheme for decreasing the pre-caching time, or increasing the probability for adequate service level of multiple streams using the same cache proxy. The heart of the proposed scheme, is an algorithm that utilizes cooperation between the flow management of different streams, which can improve the ability of the cache to provide service to all its clients. The idea is based on the willingness of streams to reduce their used bandwidth and allow other streams that may need it more to use it. In such a case, when congestion is detected, or when the amount of the buffer's data is decreased, the cache proxy can reduce other flows by manipulating the TCP flow control. Then, the load on the bottleneck link is reduced and the needed flow rate increases (or at least is not reduced), by using the TCP normal flow and congestion control mechanism. We also investigated a more practical approach that blocks the TCP connection for a short time interval, and showed that it may solve the problem. Our simulation results indicate that the proposed schemes achieve much better success rates (i.e. full delivery of both movies) than schemes in which no cooperation is used. Using our algorithms can reduce the pre-caching time by a factor of 3, or increase the probability for adequate service level by 30% using the same pre-caching time. A similar idea can be used in 3G/4G cellular networks where multiple streams compete on a limited bandwidth. In such networks, the bandwidth bottleneck is between the cache proxy and the clients that use the cellular devices, thus achieving flow control cooperation is a bit more complex, yet the benefit of cooperation remains very high.

References

1. S. Acharya and B. Smith. Middleman : A video caching proxy server. In *Proceedings of 10th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, June 2000.
2. R. Cohen and I. Dabran. The "Last-Copy" Approach for Distributed Cache Pruning in a Cluster of HTTP Proxies. In *7'th International Workshop for High-Speed Networks (PfHSN 2002)*, Apr. 2002.

3. I. Cooper, I. Melve, and G. Tomlinson. Internet Web Replication and Caching Taxonomy. RFC-3040, Jan. 2001.
4. M. W. Garrett and A. Fernandez. Variable bit rate video bandwidth trace using mpeg code. Available at: thumper.bellcore.com/pub/vbr.video.trace/MPEG.description, Nov. 1994.
5. K. Hua, Y. Cai, and S. Sheu. Patching: A multicast technique for true video-on-demand services. In *ACM Multimedia*, pages 191–200, Sept. 1998.
6. S. McCanne and S. Floyd. ns-LBL Network Simulator. Available at: http://www-nrg.ee.lbnl.gov/ns/.
7. Z. Miao and A. Ortega. Proxy caching for efficient video services over the Internet. In *9th International Packet Video Workshop*, Apr. 1999.
8. Z. Miao and A. Ortega. Scalable proxy caching of video under storage constraints. IEEE J. Selected Areas in Communications, 20(7):1315– 1327, Special issue on Internet Proxy Services., Sept. 2002.
9. R. Rejaie, M. Handley, and D. Estrin. Quality adaptation for congestion controlled video playback over the internet. In *SIGCOMM*, pages 189–200, 1999.
10. R. Rejaie, H. Yu, M. Handley, and D. Estrin. Multimedia proxy caching mechanism for quality adaptive streaming applications in the internet. In *IEEE INFOCOM*, pages 980–989, Mar. 2000.
11. S. Sen, J. Rexford, and D. F. Towsley. Proxy prefix caching for multimedia streams. In *IEEE INFOCOM*, pages 1310–1319, Mar. 1999.
12. O. Verscheure, P. Frossard, and J.-Y. L. Boudec. Joint smoothing and source rate selection for guaranteed service networks. In *IEEE INFOCOM*, pages 613–620, Apr. 2001.
13. P. Vixie and D. Wessels. Hyper Text Caching Protocol (HTCP/0.0). RFC-2756, Jan. 2000.
14. D. Wessels and K. Claffy. Internet Cache Protocol (ICP). RFC-2186, Sept. 1997.
15. Z.-L. Zhang, Y. Wang, D. H. C. Du, and D. Shu. Video staging: a proxy-server-based approach to end-to-end video delivery over wide-area networks. *IEEE/ACM Transactions on Networking*, 8(4):429–442, 2000.