On Workload Merging and Filtering Effects in Hierarchical Wireless Media Streaming

Jean Cao Carey Williamson
Department of Computer Science
University of Calgary
Email: {caox,carey}@cpsc.ucalgary.ca

ABSTRACT

In wireless networks, bandwidth is relatively scarce, especially for supporting on-demand media streaming. In wired networks, multicast stream merging is a well-known technique for scalable on-demand streaming. Also, caching proxies are widely used on the Internet to offload servers and reduce network traffic. This paper uses simulation to examine a caching hierarchy for wireless streaming video distribution, in combination with multicast stream merging. The main purpose is to gain insight into the filtering effects caused by caching and merging. Using request frequencies, entropy, and inter-reference times as metrics, we illustrate how merging, caching, and traffic aggregation affect the traffic characteristics at each level. The simulation results provide useful insights into caching performance in a video streaming hierarchy.

General Terms: Design, Performance
Keywords: IEEE 802.11 Wireless LANs, Media Streaming, Network Simulation

1. INTRODUCTION

Scalability is an important issue in any client-server network system. It is especially challenging to support video-on-demand (VoD) applications in IEEE 802.11-based wireless local area networks (WLANs), because VoD has a relatively high bandwidth requirement, and WLAN bandwidth is limited. For example, experiments show that only 8 concurrent video streams of 500 kbps each can be successfully delivered in an IEEE 802.11b WLAN [4]. In an 802.11a WLAN, with a nominal capacity of 54 Mbps, it is estimated that 32 such streams can be supported.

Hierarchical wireless video streaming has been proposed as a solution to this problem, enabling stadium-scale wireless media content delivery [5]. Relying on the use of multi-channel, power control, and caching technologies, the simulation results in [5] show that the proposed multi-level system can support over 1000 concurrent clients using an 802.11a WLAN. However, achieving this level of scalability requires 64 carefully positioned cache-and-relay proxies.

In this paper, we examine the potential of applying multicast stream merging in wireless video streaming hierarchies, to improve the scalability of the system and to ease the task of cache placement. As indicated in [5], in a hierarchical media delivery system, the root is usually the bottleneck. While stream merging is a promising solution, it alone cannot fully offload the server bandwidth requirements in our target system. However, adding a caching proxy can significantly reduce the bandwidth cost (i.e., number of concurrent streams) at the (root) media server [1, 15, 18, 19, 21].

We can further offload traffic from the server by applying *stream merging* in a hierarchical wireless system with multi-level cache proxies. Stream merging is an efficient technique that allows later-arriving clients to receive a portion of a media stream transmitted by the server to an earlier requesting client. The design issues focus on cache management in such a system. Based on previous research, prefix caching is an effective cache replacement policy for merging [1, 18, 19, 21]. However, it is not known if it is still effective in a hierarchical wireless streaming system. Without a good understanding of the workload characteristics within the hierarchy, it is difficult to answer this question.

A good understanding of the reference locality in a workload helps network designers make effective cache management decisions (e.g., cache size, cache replacement policy). Reference locality has two aspects: the popularity profile for requested objects, and temporal correlations among requests. The popularity distribution influences how well cache replacement policies such as LFU (Least Frequently Used) perform. Similarly, temporal correlations affect the performance of policies like LRU (Least Recently Used). We also argue that traditional single-value metrics like entropy, mean inter-arrival time (IAT), or the coefficient of variation (CoV) of IAT do not adequately represent the traffic characteristics in our system, due to the filtering effects of merging. Therefore, we propose two new metrics in this paper – request frequency vector, and inter-reference time (IRT) vector – to reflect the merging and filtering effects.

In this work, we use simulation to illustrate how workload characteristics change within the cache hierarchy for a stadium-scale wireless media streaming system with several thousand clients. Our results show that stream merging changes the workload into a highly skewed popularity distribution for media data units. With prefix caching at the first level cache, the request distribution becomes more evenly distributed at subsequent levels of cache, for either sequen-

WMuNeP'08, October 27, 2008, Vancouver, BC, Canada.

tial or non-sequential media workloads. Furthermore, IRTs are usually large. These characteristics have implications on the effectiveness of caching at subsequent levels of the hierarchy.

The main contributions of this work are not limited to presenting L2 proxy performance. Moreover, it provides:

A means for higher level (above L2) proxy cache management design Previous work on multicast with cache assistance focuses on finding analytical bounds [1, 19, 21]. Their results cannot be applied for L2 proxy design due to the lack of knowledge on the ingress workload at a L2 proxy. This work fills up this gap by studying reference locality at each proxy level. The same method can be applied for any deeper proxy level along the hierarchy.

Metrics that capture the special features caused by merging Merging makes the workload highly related to the time in a *full media* transmission. Some workload properties cannot be captured by single value metrics, but can be easily shown by a metric vector with a value for each data unit. We find that the request frequency vector and the IRT vector are particularly useful in our study.

The rest of the paper is organized as follows. Section 2 provides a brief summary of prior related work. Section 3 presents our system model. Section 4 presents our experimental methodology, including descriptions of the simulator, workloads, and performance metrics. Simulation results appear in Section 5, along with a discussion of workload sensitivities. Finally, Section 6 concludes the paper, and highlights future research directions.

2. BACKGROUND AND RELATED WORK

Many multicast streaming schemes have been proposed in the past decade. To provide immediate service for asynchronous VoD clients, *patching* [12], *tapping* [6] and *stream merging* [8, 9, 10] were proposed.

In patching and tapping, the server transmits the full media stream for the first requesting client. A client that arrives slightly later listens to two streams: its own for the initial part of the media object, and one of the on-going streams for the rest of the required data, which is then pre-buffered for later (future) playback.

Multicast stream merging is similar, except that clients can listen to *multiple* on-going streams for more efficient pre-buffering, as long as the client's bandwidth allows. Merging has logarithmic scalability, which is close to the optimal lower bound [10]. If clients can listen to multiple on-going streams, the server bandwidth cost $B_{server} = ln(N_i + 1)$, where $N_i = \lambda_i T_i$ is the average number of requests for the media stream during time T_i with arrival rate of λ_i [10]. For example, with 1000 client requests per media duration, the average server bandwidth cost is about 7 streams.

Previous research on Web workloads shows that reference locality can be exploited in Web cache design. Reference locality has two aspects: the popularity of requested objects, and temporal correlations in the workload [2, 7, 11, 13, 16]. As proposed in [11], we use *entropy* as one of the metrics for popularity. However, most of the single-valued metrics proposed for Web workloads are not suitable for merging traffic. During our study of merging and filtering effects, we found that the request frequencies and IRTs are unevenly distributed among different media data units. Therefore, we propose to use request frequency vector and IRT vector to characterize popularity and temporal correlations.

Filter effects in Web caching hierarchies have been well studied in [20], as a phenomenon that a cache changes the structural characteristics of the workload presented to the next level because only missed requests are included in the egress workload. To gain insight into the filtering effects at a proxy, Fonseca *et al.* [11] have suggested a framework for analyzing Web request streams. They decompose the functions at a proxy into aggregation, filtering, and de-aggregation. We apply this idea to our system, and identify three operations – aggregation, merging, and caching – that specifically fit our scenario. We study the reference locality changes in the ingress and egress workloads due to these effects.

3. SYSTEM MODEL

3.1 Wireless Media Streaming System

We focus on a two-level proxy hierarchy for media streaming, as shown in Figure 1. However, the methodology that we are proposing can be applied to deeper proxy hierarchies.

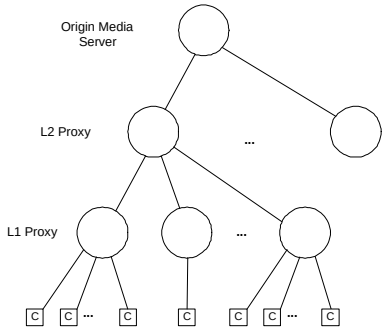


Figure 1: Wireless media streaming system

As shown in Figure 1, the topmost component is the *origin media server*, which is the central repository for all media objects. At the leaf level are the *streaming clients*. Clients send requests for media objects up the hierarchy, while media streams are delivered down the hierarchy to the clients.

Using merging, a client can listen to other ambient transmissions over the same wireless channel, and pre-buffer later data units, if any, for its future use. By the time of playback, if the current data unit is already in the buffer, no transmission is needed from the proxy. The bandwidth can be saved in this way.

To further increase the total number of streaming clients supported, cache proxies are used between the media center and the streaming clients. The proxies perform on-demand caching for media objects, on a data unit basis, where a *data unit* is a fixed-size or fixed-duration piece (e.g., one frame) of a media object.

Two levels of proxies are shown in Figure 1, with L1 proxies closer to the clients. Each client associates with a specific L1 proxy during the media streaming (i.e., no mobility is assumed). Media requests that cannot be satisfied by L1 proxies are propagated to an L2 proxy. In a multi-level proxy system, the request is propagated recursively to a higher level proxy. In our two-level proxy hierarchy, L2 proxies can communicate with the origin media server, where any required media data units are guaranteed to be found.

With *proxy merging* capability, a proxy can also monitor the wireless channel for transmissions to peer proxies. If

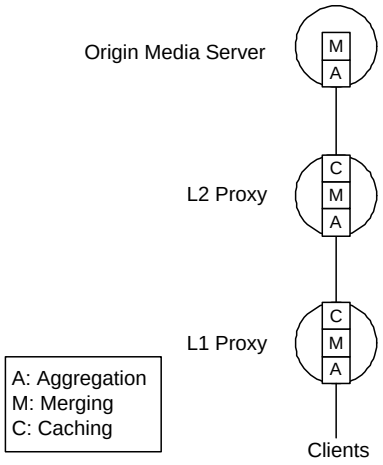


Figure 2: Decomposed operations

a useful future media data unit is observed, the proxy will forward it to its child proxies or clients right away, if there is bandwidth, for potential future use.

3.2 System Operation

To gain insight into the effects of different system configurations on the workload at different proxy levels, we decompose the functionality at a proxy into three operations – aggregation (A), merging (M), and caching (C), as shown in Figure 2. Note that the root node represents the media server where *all* of the media contents are stored. Hence, no cache is implemented at the root node.

Each operation impacts on the traffic flow in a different way. Hence, we analyze them separately.

Aggregation of requests happens at the ingress of each proxy level. At an L1 proxy, requests from all associated clients are aggregated. At an L2 proxy, aggregation happens for requests received from multiple L1 proxies. When multiple request streams are mixed together, some workload characteristics, such as popularity and inter-reference time, might be changed. If so, traffic aggregation might affect the cache performance and the system performance. This paper addresses this issue by studying the changes of workload characteristics due to changing the numbers of clients and proxies.

Merging fundamentally benefits from pre-buffering. If a future data unit to play has been stored in the buffer, no transmission is required for this time slot (i.e., time to transmit a data unit). The more data units can be pre-buffered before playback, the more server bandwidth is saved. Pre-buffering in merging is similar to pre-fetching. The main difference is that in most pre-fetching schemes, data movement is determined by a specific algorithm. With merging, pre-buffering depends on system dynamics; a client or proxy does *not* have control on what specific data and how many data units can be pre-buffered in a time slot.

Several factors affect pre-buffering performance, such as how many on-going transmissions occur, and how many streams a client can receive at the same time. In wireless environments, a client can, ideally, listen to as many streams as available. However, this is not true in wired or mixed environments with heterogeneous client device capabilities. How many on-going transmissions depends on the workload characteristics, which is a main focus of this study.

Caching offloads requests from the root server, which tends to be the bottleneck. Cache management (i.e., cache size and cache replacement policy) plays an important role in achieving good caching performance. In addition, previous studies have shown that cache replacement policies perform differently with respect to traffic characteristics. For stream merging, prefix caching is highly effective [1, 18, 19, 21]. However, the interplay is bidirectional. That is, traffic characteristics affect caching performance, while caching and merging also have filtering effects on the traffic characteristics. How does the filtered traffic impact the next level of cache? This is a key question addressed in this study.

Aggregation, merging and caching all have impacts on traffic characteristics. This paper studies the filtering effects from these operations, and how they affect cache replacement policy selection.

3.3 Assumptions

The following assumptions simplify our study. The proxy hierarchy is pre-configured and static. Media streams are CBR, with fixed-size data units. No admission control is applied, which means all client requests are granted in order to study the bandwidth cost. Although we focus on cases that users do not move: a streaming session, once started, stays with the same L1 proxy until it ends, the non-sequential media workload studied in Section 5.6 can also simulate the random request pattern at a proxy due to user mobility. The client's media player has sufficient buffer to store all pre-buffered data units; we leave the limited client playback buffer for future work. At each proxy, incoming and outgoing streams use different wireless channels without interference. Packet loss due to wireless channel conditions can be recovered by the MAC-layer protocol. The bandwidth consumption for sending user requests upstream is ignored due to the small volume.

4. METHODOLOGY

4.1 Simulation Overview

We use simulation to answer the research questions posed earlier. We developed our own event-driven simulator of the system in Figure 1, a two-level proxy hierarchy. The system can be configured using the parameters listed in Table 1. A media streaming workload file is provided as input. For each new user request in the workload, the user is associated with a L1 proxy uniformly at random. The simulator operates at the media stream level; the network protocol stack is not simulated.

The simulator runs the algorithm described in Section 3.1. More specifically, in each time unit, each client monitors the wireless channel and saves all the data units delivered from the proxy with which it is associated. At the same time, each proxy downloads all the data units delivered from its parent proxy. At the end of each time unit, each client checks its playback buffer. If the next required data unit is not in the buffer yet, a request is sent to its proxy. If the data unit is not in the proxy cache, the request is propagated to its parent proxy recursively up to the root, where the request can be guaranteed to be served. In the next data units, all the unique requests at each proxy and the root are served accordingly. Proxy cache is updated as well. For analysis purpose, at each proxy, the number of requests, requested

unique data units, cache hit ratio, and bandwidth cost (i.e., number of data units in each time unit) are recorded.

4.2 Client Workload

Our studies are based on the client workloads that feed into the L1 proxy. All media-streaming workloads for our simulator are generated using GISMO (Generator for Internet Streaming Media Objects) [14].

In this study, we use two sets of workloads: sequential media workloads and non-sequential media workloads. Sequential media workloads are as simple as possible, to better understand system behaviour. These workloads have a *single* read-only media object that is 4 minutes long, consisting of 7200 data units (30 frames per second). Simulation time is normalized to the transmission time of one frame (one time unit). The session arrival process is Poisson, with an average rate of $\lambda = 10$, 100, or 1000 session arrivals per media duration to represent low, medium, or high load, respectively. Each session starts playback at the beginning of the media object, and plays the data units in sequence until the end of the media.

Non-sequential media workloads are on top of the sequential workloads' setup with additional features like early termination and some VCR-like operations such as rewind or fast-forward. It can also simulate the random request pattern at a proxy due to mobility. In the non-sequential media workload used in this paper, 5% clients terminate at the beginning portion of the playback, having the length following a Pareto distribution with shape parameter $\alpha = 1.0$ and scale parameter $k = 0.001$. Other configurations might generate a workload that causes different results. However, because the main purpose of this study is not to exhaust all the possibilities but to provide a general research methodology, only one non-sequential media workload is used as a representative.

4.3 Metrics for Traffic Characterization

For traffic characterization, we use request entropy and frequency vector to study the reference popularity, and use inter-reference time (IRT) vector to study the temporal locality in a traffic flow. These metrics are defined next.

To evaluate at how well these metrics can capture traffic characterization and determine on caching policies, we use bandwidth cost (defined as concurrent number of streams) to evaluate the overall system performance, and use hit ratio for the cache performance.

4.3.1 Entropy

Entropy $H(X)$ was proposed in [7, 11] as a way to measure the uniformity or non-uniformity of the popularity distribution in a workload. More specifically, entropy captures the skew of requests to a set of objects. $H(X)$ is defined as:

$$H(X) = -\sum_{i=1}^{n} p_i \ \ log_2\ (p_i) \tag{1}$$

where p_i is the probability of the ith object among n objects. Low values of $H(X)$ represent highly skewed popularity, while larger values imply more uniformly distributed popularity.

For the same reasons indicated in [11], $H(X)$ is normalized to be able to compare workloads with different numbers of requests. The normalized entropy H^n is calculated as:

$$H^n = H(X)/H_0(N) \tag{2}$$

where N is the total number of references (i.e., requests) in a workload, and $H_0(N) = log_2(N)$ is the largest possible value of H(X).

Furthermore, to highlight differences of H^n values, we use the scaled normalized entropy, H^s, as in [11]:

$$H^s = -log_{10}(1 - H^n). \tag{3}$$

Similar to $H(X)$, a larger H^s value represents more uniformly distributed popularity. A cache replacement policy like LFU, which uses frequency as the main input, may perform poorly for such a workload.

4.3.2 Request frequency vector

The request frequency vector keeps track of how often each media data unit is requested during a specified time interval (e.g., typically a full media playback time). The request frequencies for each media data unit vary depending upon the location of the data unit in the stream. In fact, merging generates a repeating pattern of references in every *full media* play. No single-valued metric can successfully capture the popularity characteristics among data units caused by merging. As an example, H^s values can provide an indication of whether LFU is suitable or not, but it cannot show where the popular units are located within the media object. With frequency vector, one can easily tell if most of the popular units are near the beginning of the media object; if so, simple prefix caching can be applied. For this reason, we use the full vector of reference frequencies together with H^s to represent popularity.

4.3.3 IRT vector

Temporal correlations in a workload can affect the performance of a cache replacement policy such as LRU, which uses time as its decision variable. In our study, we found that the inter-reference time (IRT) captures important features of the temporal locality for a specific data unit. The IRT at a node is the average number of time slots between two consecutive requests to the same media data unit.

In this study, we use the IRT vector to illustrate IRT values across the whole data set. For the same reason stated previously, no single metric, such as average IRT or CoV, can successfully capture the temporal correlations among all data units. The IRT vector shows the mean IRT for each data unit in the media object.

5. SIMULATION RESULTS

This section presents our simulation configuration and the simulation results. Sections 5.2 to 5.5 present results for sequential media streaming workloads. We start by studying simple "caching only" and "merging only" cases, then consider them in combination. Results for non-sequential workloads are presented in Section 5.6. Implications of the results are discussed following each experiment.

5.1 System Configuration

The default system configuration in our experiments has two levels of proxies, with 30 L1 proxies and a single L2 proxy. Each of the L1 proxies handles 100 clients, so the total number of clients in the system is 3000. With no caching or stream merging, naive unicast streaming to each client

Table 1: System Parameters for Each Operation

Operation	Parameters	Value Range
Aggregation	Average arrival rate (client sessions per media duration) Number of proxies at each level	[10, 100, 1000] [1, 10,...60]
Merging	Merging capability Number of simultaneous streams received by a client or a proxy	[on, off] (1 means unicast; ∞ means unlimited streams)
Caching	Caching capability Cache replacement policy at each proxy level Cache size at each proxy level	[on, off] [LRU, LFU, Prefix] 0-10% of the media length

would have a server bandwidth cost of 3000, which provides a baseline comparison point for the simulation results.

When caching is used, the default cache replacement policy is prefix caching (Prefix) at the L1 proxy. The default cache size is 10% of the complete media object size, unless specified otherwise. If Prefix is used at the L2 proxy, it caches the next portion of the media stream following the L1 prefix. For clarity, we refer to this approach as Prefix2.

5.2 Caching Effects

The first simulation experiment considers the use of caching in the wireless streaming hierarchy, with no multicast stream merging. The results are shown in Table 2. The workload has relatively high H^s values at each level, indicating that all data units have similar popularity. Therefore, prefix caching at L1 and LFU at L2, as in the configuration of this test, may not perform well.

The average bandwidth usage confirms that this unicast-based system does not scale well. There is no bandwidth saving at the bottom level, since each L1 proxy delivers about 100 streams to its clients. There is about 26% savings at the L2 proxy, where requests from the 30 L1 proxies are aggregated. This saving is from both L1 cache and the aggregation (i.e., requests for the same data unit are transmitted only once). At the root, we see another small bandwidth savings of about 11%, which comes from the L2 cache. However, these savings are insufficient to make the system scalable: the media server requires a median bandwidth of almost 2000 streams, which is not practical.

Table 2: Performance of a Cache Only System (Prefix/LFU, 30 L1 proxies, medium load: $\lambda = 100$)

Level	H^s	Avg BW*
Root	5.93	1952.21
L2 Proxy	5.66	2197.50
L1 Proxy	3.64	99.25

*: BW is de↓ned as the number of concurrent streams.

These results indicate that if a system does not have multicast capability, the higher levels will be the bottleneck. Hence, only light load can be supported. LFU is not suitable at an L2 proxy, since all media units have similar popularities. Our results also show that IRTs decrease after traffic aggregation, and are fairly evenly distributed across all media data units. Therefore, the LRU scheme could perform better due to the lower IRTs.

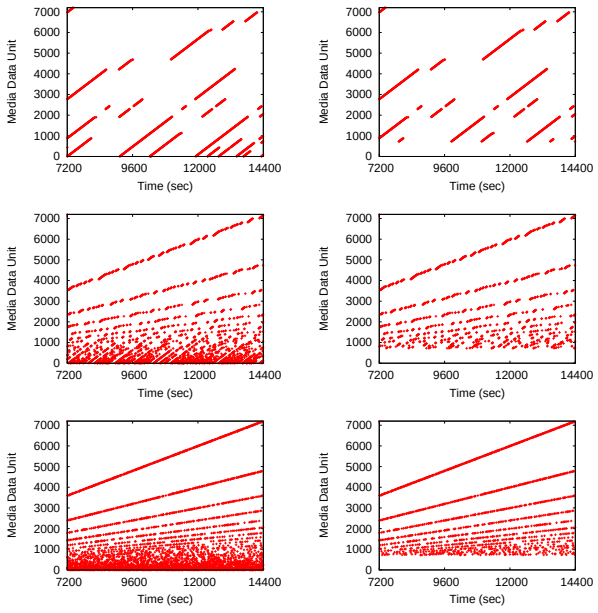
5.3 Merging Effects

5.3.1 *Level 1 Merging Effects*

The next experiment considers multicast stream merging. It is known that merging can achieve high scalability [8, 9, 10]. For example, with 1000 sessions, the server bandwidth cost is about 7 streams.

In our simulation, we found a periodic phenomenon during each full media playback: bandwidth cost is lower at the beginning of the media, and gradually rises and peaks at the end of the media, then drops back to low at the beginning of next full media time. Due to this repeating pattern, all our results are based on one full media time, namely the second such playback time in the system. We use the first full media duration as a warm-up period.

Our simulation also shows that the bandwidth cost varies with time, based on the dynamics of session arrivals and the merging achieved. When $\lambda = 1000$, the highest bandwidth cost observed is 16, which is approximately double the steady-state average. While the scalability of merging is excellent, the average bandwidth cost can still be highly variable. Nonetheless, we use mean values as a simple indicator of the system performance.



(a) before L1 proxy (b) after L1 proxy

Figure 3: Requested data units with time

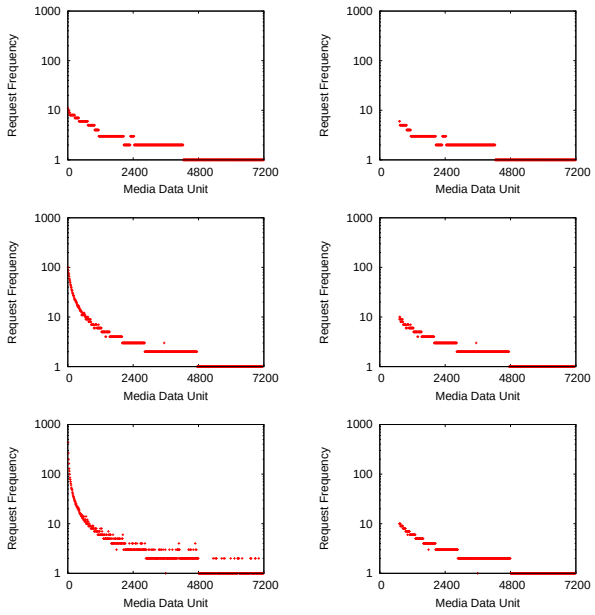
Stream merging (and its pre-buffering strategy) changes the distribution of client requests observed in the system.

Figure 3 plots the requested data units versus time when merging is enabled. The graphs on the left show the ingress requests before the L1 proxy, while the graphs on the right show the egress requests after the L1 proxy. From top to bottom, the graphs represent light, medium, and high load scenarios. The general trend expected in these graphs is an upward sloping line, representing successive media units (on the vertical axis) requested as a function of time (on the horizontal axis).

Figure 3 shows that the transmitted streams are somewhat discontinuous. That is, the streams are rather short-lived pieces at low traffic load, but longer at higher loads. This makes sense based on the arrival rate for client sessions. With few clients active, they are likely to be at different points in the media stream, and few opportunities for merging exist. With many active clients, there are many opportunities for merging, and a given target stream may be long-lived.

Figure 3 also shows that the slope of the lines tends to decrease at higher loads. This phenomenon is explained by the pre-buffering: media units required in the near future have already been buffered, and need not be requested from the proxy. It also shows that the prefix of the media stream is requested more frequently, especially at higher loads.

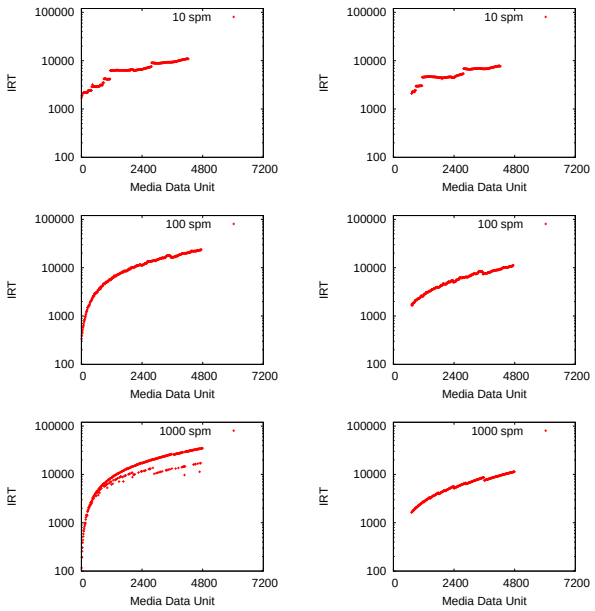
These request patterns mostly remain after filtering by the L1 cache, as shown in Figure 3(b). The primary difference is the reduced demand for low-numbered media units, because of the prefix caching used at the L1 proxies.



(a) before L1 proxy (b) after L1 proxy

Figure 4: Request frequencies

Merging changes the popularity distribution dramatically, from its fairly evenly distributed popularity in Section 5.2 to a skewed popularity in Figure 4. This is why prefix caching is the best to use with stream merging, as claimed in earlier studies [1, 18, 19, 21]. After the L1 cache (Figure 4(b)), the popularity skew is still present, but less pronounced. As a result, the effectiveness of prefix caching (or LFU) at an L2 proxy is reduced compared to an L1 proxy.



(a) before L1 proxy (b) after L1 proxy

Figure 5: IRT vector

Figure 5 shows the IRT for each media data unit. In general, there is a monotonically increasing trend with respect to media data unit number: low-numbered items are requested frequently, at short intervals, while high-numbered items are requested less often, at large intervals. In Figure 5(a), we see that IRT increases quickly from the beginning of the media towards the end. The later portion of the media stream receives few references, since many clients may merge to a few target streams. At high load scenario (Figure 5(a) bottom), requests to some data units might have shorter IRTs. Therefore, the plot shows more than one line other than a single smooth line in the moderate load scenario (Figure 5(a) middle).

5.3.2 *Level 2 Merging Effects*

Stream merging can also be applied at proxies. With *proxy merging*, a proxy listens to all transmissions on the (broadcast) wireless channel, and forwards all relevant data units to its children (clients or proxies). These data units include those requested within its own subtree, and those being transmitted to peer proxies. This scheme is meant to pre-buffer as many data units as possible, as early as possible. Therefore, there is a potential increase in the bandwidth cost at the lower level from "pushing" unrequested data units down the hierarchy, but we expect to see overall improved bandwidth savings in the long run.

The H^s values in Table 3 show that using proxy merging, the popularities are more skewed. This phenomenon would be good for L2 caching policies such as LFU.

The bandwidth costs in Table 3 show the gains from proxy merging. Without proxy merging, the bandwidth requirements are as high as 140 at L2 proxy and the root due to the aggregation of 30 L1 proxies. However, using proxy merging, there is a 20-fold bandwidth savings at higher levels. It is especially important to the root, which tends to be the bottleneck. In the remaining experiments, proxy merging is

Table 3: Performance with and without Merging at Proxy (30 L1 proxies, medium load: $\lambda = 100$)

Proxy Merging	Level	H^s	Avg BW
Without	L2 Proxy	1.04	140.21
	L1 Proxy	1.19	5.04
With	L2 Proxy	0.66	7.24
	L1 Proxy	0.94	7.24

used to enable better overall system performance.

5.4 Merging and Caching

In this section, we study the combination of caching and multicast stream merging. In Section 5.4.1, we study the combined filtering effects of merging with prefix caching, and discuss the potential impacts on L2 cache design. In Section 5.4.2, we compare three cache replacement schemes (LFU, LRU, and Prefix) at the L2 proxy.

5.4.1 L1 Prefix Caching and Merging Effects

As illustrated in the last section, merging changes the workload to be highly skewed toward the beginning of a media stream (see Figure 4(a)). After filtering by the L1 proxy cache, requests for the most popular media units are removed from the workload, as shown in Figure 4(b). However, some skew still exists in the workload, favouring the earlier portions of the media.

We expect to see LFU or Prefix2 perform well in this context, with comparable performance in terms of bandwidth saving and hit ratio. However, with increased cache size at the L1 proxy, the skew disappears from the workload entering the L2 proxy. Also, if clients are allowed to do fast-forward or other VCR-like operations, the popularity distribution might not be as skewed. In these cases, prefix caching might not perform as well as LFU.

Figure 5(b) shows that after the L1 proxy filtering effect, all IRTs are larger than 720 time units (the effective size of the L1 cache). Therefore, we can conclude that LRU is a poor choice for the L2 proxy; in fact, its hit ratio will be 0.

5.4.2 L2 Cache Management Policies

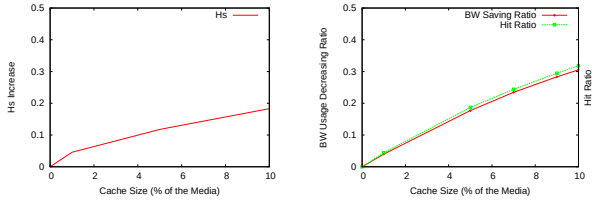
We conducted a simulation experiment with LFU, LRU, and Prefix2 cache replacement policies at the L2 proxy. From the previous section discussing the workload characteristics after merging and caching at L1 proxy, we expect to see LFU and Prefix2 perform equally well at L2, with both outperforming LRU.

Table 4 presents results of a scenario with 30 L1 proxies. The results confirm our prediction: LFU and Prefix2 both have about 30% bandwidth savings, with a hit ratio of 30%. LRU performs the worst, with no bandwidth savings at all.

Table 4: Performance of Different Cache Replacement Policies (30 L1 proxies, medium load: $\lambda = 100$)

	Level	H^s	Avg BW	Hit Ratio
LFU	Root	1.81	1.57	0.30
	L2 Proxy	1.52	2.26	
LRU	Root	1.57	2.26	0.0
	L2 Proxy	1.52	2.26	
Prefix2	Root	1.81	1.57	0.32
	L2 Proxy	1.52	2.26	

By examining different L2 proxy cache sizes, we can see how cache sizes affect the egress workload characteristics of L2. In Figure 6(a), we see that the entropy H^s increases gradually with cache size. It illustrates that the L2 cache has a filtering effect on the egress workload: a larger cache size makes the request distribution in the egress workload more uniform. At the same time, a larger cache size, with a proper cache replacement policy (such as Prefix2), achieves a higher hit ratio and correspondingly better bandwidth savings (see Figure 6(b)),



(a) Entropy (b) Bandwidth Savings

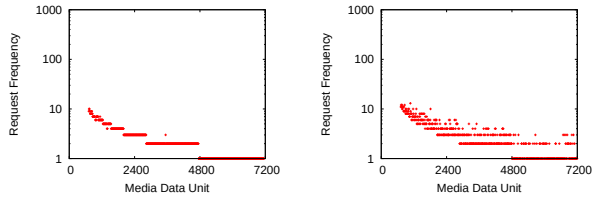
Figure 6: Effect of different cache sizes at L2 proxies (Prefix2) on entropy of egress workload and network performance

5.5 Aggregation Effects

Figure 4 and Figure 5 illustrated the skewed popularity of media units after merging and caching at a L1 proxy. In that test scenario, the H^s in the egress workload is 1.03.

The L2 proxy aggregates the egress workloads from multiple L1 proxies. The distribution of media units requested becomes more evenly distributed. Our results show that with 10 to 60 L1 proxies, the aggregated H^s is about 1.55. Therefore, aggregation generates a more uniform distribution of requests to the individual blocks of the media object.

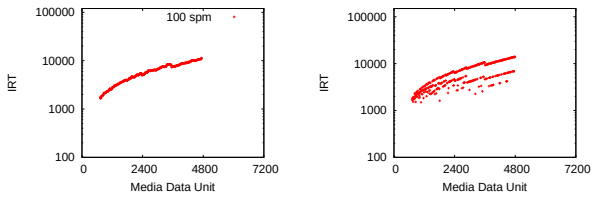
Figure 7(a) presents the request frequency results for the egress workload from a L1 proxy. For the same client request rate (100) at each L1 proxy, the aggregated workload from 30 L1 proxies, which is the ingress workload of L2 proxy, has a "flatter" request frequency distribution, as shown in Figure 7(b). In the aggregated traffic flow, there are more requests for the later units in a media stream.



(a) Single (b) Aggregated

Figure 7: Aggregation effects of 30 proxies on request frequency

We can also see the effect of aggregation on the IRTs. The IRT vector of the egress workload from a L1 proxy is shown in Figure 8(a), and the aggregated result is shown in Figure 8(b). The egress workload from a single L1 proxy generally has monotonically increasing IRTs, which are strongly correlated with relative data unit positions. When multiple streams are aggregated at L2, the IRTs become more diverse, especially across earlier units.



(a) Single (b) Aggregated

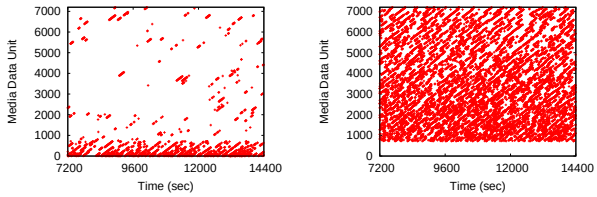
Figure 8: Aggregation effects of 30 proxies on IRT

Additional simulation experiments (not included here) show that the diversity of IRTs increases with the number of L1 proxies. The IRTs always exceed that associated with the L1 cache size, because of L1 cache filtering.

5.6 Non-sequential Media Workloads

We close our experiments with a discussion of sensitivities to client workload characteristics, particularly with respect to sequential stream viewing versus VCR-like functionality (i.e., pause, rewind, fast forward, random jump, early termination). The latter functionality alters the sequential nature of the requests for media data units, adding greater randomness into the request workload.

Figure 9 illustrates the differences in requested data units for non-sequential media workloads (compared to sequential workloads in Figure 3). As shown in Figure 9 (a), there are more short continuous lines, with parallel rather than decreasing slopes. These two phenomena are related: due to the VCR-like operations, playbacks are in many short sections, therefore, clients cannot benefit from merging as much as in sequential playback. Without pre-buffering, requests are sequential, which causes the lines to be approximately parallel. At moderate traffic load ($\lambda = 100$), the traffic flow is highly skewed ($H^s = 0.91$), partially due to early terminations which cause the first 5-10% of data units to be frequently requested, and also due to random jump operations.

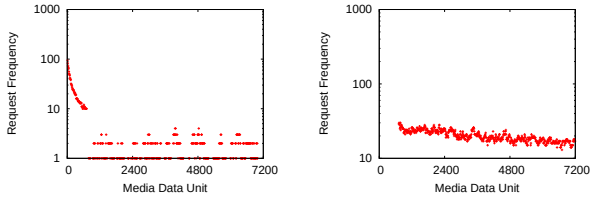


(a) before L1 proxy $H^s = 0.91$ (b) before L2 proxy $H^s = 2.81$

Figure 9: Requested data units with time

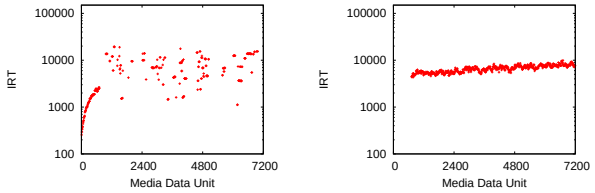
Figure 9 (b) shows the aggregated requests by 30 L1 proxies. It shows the request are more evenly distributed across all data units ($H^s = 2.81$), except that the first 10% are removed by L1 proxy's Prefix cache.

Figure 10 and Figure 11 depict the frequency vector and IRT vector before and after the L1 proxy. As shown, the most frequently requested prefix units are removed by L1 proxy. Although 30 aggregated traffic flows are fairly evenly distributed (Figure 10(b)), there is a slight decreasing trend which can favour prefix caching. However, due to the overall even distribution, we should not expect a significant benefit



(a) before L1 proxy (b) after L1 proxy

Figure 10: Request frequencies: before (a) and after (b) L1 proxy



(a) before L1 proxy (b) after L1 proxy

Figure 11: IRTs: before (a) and after (b) L1 proxy

from Prefix2 and LFU. This conclusion is confirmed by the average bandwidth cost and cache hit ratios in Table 5. With a 10% cache, the bandwidth saving is only about 13.50%.

Figure 11 also shows that LRU would be ineffective, since most IRTs exceed 4000. Unless a very large cache size is used, LRU cannot save any bandwidth. This is confirmed by the average bandwidth cost in Table 5: for an LRU cache at L2, there is bandwidth saving of only about 0.6%.

Table 5: Performance at Each Level (30 L1 proxies, medium load: $\lambda = 100$)

	Level	H^s	Avg BW	Hit Ratio
LFU	Root	2.73	15.51	0.1351
	L2 Proxy	2.63	17.93	
LRU	Root	2.83	17.80	0.0063
	L2 Proxy	2.63	17.93	
Prefix2	Root	2.73	15.50	0.1355
	L2 Proxy	2.63	17.93	

Although the non-sequential media workloads in this section show that there is no benefit in using any of LRU, LFU or Prefix2, this conclusion may not be true for all non-sequential media workloads. A different workload might have different characteristics that might benefit LRU if the IRTs are small, or LFU if there are some "hot spots" in a media object. Our study shows how our three metrics can be used to characterize workloads, and assist with the selection of proper caching policies.

6. CONCLUSION

In this paper, we study filtering effects caused by aggregation, caching, and merging in a wireless video streaming hierarchy. Reference locality in the workload at different proxy levels is analyzed using extensive simulations. Popularity and temporal locality are captured using entropy, request frequency vector, and IRT vector as metrics. The simulation results provide useful insights on L2 cache per-

formance. Our key findings include:

- For sequential media workloads, merging generates highly skewed requests, which are smoothed by prefix caching. With aggregation, although traffic becomes slightly more uniform, it is still skewed towards the prefix. Therefore, LFU and Prefix2 are useful at L2 proxies.

- For non-sequential media workloads, the request frequencies are more evenly distributed across all data units after the L1 proxy filters out the popular prefix. LFU and Prefix2 have little benefit in the workloads tested. At the same time, LRU performs even worse due to the high IRTs.

- Using request frequency vector and IRT vector can effectively depict the traffic characteristics and contribute to proper decision making in L2 caching policies in a merging/caching wireless streaming hierarchy.

- Separating the operations merging, caching and aggregation can identify the impacts from different operations in a caching hierarchy. Among these three, merging has the main effect in skewing the popularity towards the prefix of a media object.

Although our study focuses on a 2-level wireless proxy hierarchy, the methodology proposed can be applied to a hierarchy with any levels and/or with a wired or mixed connections .

Future work will focus on considering limited client playback buffer and comparing merging with other multicast schemes and on studying scenarios with multiple media objects.

7. REFERENCES

[1] J. Almeida, D. Eager, M. Ferris, and M. Vernon, "Provisioning Content Distribution Networks for Streaming Media", *Proceeding of IEEE INFOCOM Conference*, New York, NY, pp. 1746-1755, June 2002.

[2] G. Bai and C. Williamson, "Workload Characterization in Web Caching Hierarchies " *Proceedings of IEEE/ACM MASCOTS Conference*, Fort Worth, TX, pp. 13-22, October 2002.

[3] A. Bestavros and S. Jin, "OSMOSIS: Scalable Delivery of Real-Time Streaming Media in Ad-Hoc Overlay Networks", *Proceedings of IEEE ICDCS Workshop on Data Distribution in Real-Time Systems*, Providence, RI, pp. 184-195, May 2004.

[4] X. Cao, G. Bai, and C. Williamson, "Media Streaming Performance in a Portable Wireless Classroom Network", *Proceedings of European Workshop on Internet Multimedia Systems and Applications (EuroIMSA)*, Grindelwald, Switzerland, pp. 246-252, February 2005.

[5] X. Cao and C. Williamson, "Towards Stadium-Scale Wireless Media Streaming", *Proceedings of IEEE/ACM MASCOTS Conference*, Monterey, CA, September 2006.

[6] S. Carter, and D. Long, "Improving Video-on-Demand Server Efficiency through Stream Tapping", *Proceedings of International Conference on Computer Communications and Networks*, 1997.

[7] F. Duarte, F. Benevenuto, V. Almeida, and J. Almeida, "Locality of Reference in an Hierarchy of Web Caches", *Networking 2006 LNCS 3976*, pp. 344-354, April 2006.

[8] D. Eager, M. Vernon, and J. Zahorjan, "Optimal and Efficient Merging Schedules for Video-on-Demand Servers", *Proceedings of ACM Multimedia Conference*, Orlando, FL, pp. 199-202, November 1999.

[9] D. Eager, M. Vernon, and J. Zahorjan, "Bandwidth Skimming: A Technique for Cost-Effective Video-on-Demand", *Proceedings of MMCN Conference*, San Jose, CA, pp. 206-215, January 2000.

[10] D. Eager, M. Vernon, and J. Zahorjan, "Minimizing Bandwidth Requirements for On-Demand Data Delivery", *IEEE Transaction on Knowledge and Data Engineering*, Vol. 13, No. 5, pp. 742-757, September/October 2001.

[11] R. Fonseca, V. Almeida, M. Crovella, and B. Abrahao, "On the Intrinsic Locality Properties of Web Reference Streams", *Proceeding of IEEE INFOCOM Conference*, San Francisco, CA, pp. 448-458, April 2003.

[12] K. Hua, Y. Cai, and S. Sheu, "Patching: A Multicast Technique for True Video-on-Demand Services", *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, 1999.

[13] S. Jin and A. Bestavros, "Sources and Characteristics of Web Temporal Locality", *Proceedings of the 8th IEEE MASCOTS*, San Francisco, CA, August 2000.

[14] S. Jin and A. Bestavros, "GISMO: A Generator of Internet Streaming Media Objects and Workloads", *ACM Performance Evaluation Review*, Vol. 29, No. 3, pp. 2-10, December 2001.

[15] S. Jin and A. Bestavros, "Cache-and-Relay Streaming Media Delivery for Asynchronous Clients", *Proceedings of the 4th International Workshop on Networked Group Communication*, Boston, MA, October 2002.

[16] A. Mahanti, D. Eager, and C. Williamson, "Temporal Locality and its Impact on Web Proxy Cache Performance", *Performance Evaluation*, Vol. 42, No. 2/3, pp. 187-203, Septmeber 2000.

[17] A. Mahanti, A. Mahanti, and C. Williamson, "Locality Characteristics of Web Streams Revisited" *Proceedings of SCS SPECTS Conference*, Philadelphia, PA, pp. 795-803, July 2005.

[18] S. Ramesh, I. Rhee, and K. Guo, "Multicast with Cache (Mcache): An Adaptive Zero-Delay Video-on-Demand Service", *Proceedings of IEEE INFOCOM Conference*, Anchorage, AL, April 2001.

[19] B. Wang, S. Sen, M. Adler, and D. Towsley, "Optimal Proxy Cache Allocation for Efficient Streaming Media Distribution", *IEEE Transactions on Multimedia*, Vol. 6, No. 2, pp. 366-374, April 2004.

[20] C. Williamson, "On Filter Effects in Web Caching Hierarchies". *ACM Transactions on Internet Technology*, Vol. 2, No. 1, pp. 47-77, Feburary 2002.

[21] L. Zhu, G. Cheng, N. Ansari, Z. Sahinoglu, A. Vetro, and H. Sun, "Proxy Caching for Video on Demand Systems in Multicasting Networks", *Proceedings of the John Hopkins University Conference on Information Sciences and Systems (CISS)*, March 2003.