Abstract—In reliable multicast, data packets can be cached at some nodes such as repair servers for future possible retransmission in loss recovery schemes. How to cache packets to optimize the performance of loss recovery is an important issue in reliable multicast protocol design. In this paper, we present a general solution which addresses the main design problems of caching policies. We first formulate the caching policy design as an optimization problem by employing caching utility as a uniform measure. Based on caching utility, we propose an algorithm called Optimal Caching Time (OCT) for configuring the caching time of packets and demonstrate that it solves the optimization problem. Furthermore, we analyze the performance improvement of OCT caching policy compared to the existing caching policies such as FIFO, Probabilistic FIFO (P-FIFO), and Timer-Based Caching Policy (TBCP). We use ns-2 simulations to demonstrate the performance gains brought by OCT caching policy. The numerical results show that OCT caching policy improves the performance significantly, especially for heterogeneous groups of receivers. With the inherent generality of the proposed model and the OCT algorithm, they can be easily applied to the general cache design in reliable unicast or multicast applications, and in both wired and wireless networks.

Index Terms—loss recovery, reliable multicast, caching policy, optimization

I. INTRODUCTION

RELIABLE multicast provides one-to-many reliable data transmission services for many receivers concurrently. For reliable multicast protocols, reliability is guaranteed by data retransmissions to recover possible packet losses. These protocols employ various loss recovery schemes to retransmit the lost packets from near the request receivers [1]–[4]. Generally, the sender takes the ultimate responsibility to provide retransmission service to guarantee the reliability for all receivers. However, scalable reliable multicast requires to distribute the loss recovery responsibility to a number of network nodes. Thus, these nodes perform data caching and packet retransmissions for recovering packet losses locally [1]–[4].

Local loss recovery can be implemented at repair servers or designated receivers in various reliable multicast protocols. In receiver-based local loss recovery schemes, such as SRM [1] and RMTP [2], some or all of the receivers cache received data for some time and retransmit data when receiving requests from other receivers who have experienced packet loss. In server-based local loss recovery schemes such as AER/NCA [4], some Repair Servers (RS) are strategically placed in conjunction nodes with active routers. These active routers can provide network-based storage and processing using active networking technique [5]. Based on that, repair servers can perform data caching, NAK suppression, and scoped retransmissions [3], [4], [6]. Both server-based and receiver-based approaches can reduce loss recovery latency significantly.

For loss recovery purpose, a certain amount of buffer is allocated at the servers or receivers to cache packets for possible retransmissions in the future. Since most of the loss recovery schemes, such as SRM, ARM [3], and AER/NCA, use NAKs for requesting lost packets and this has been shown to be more scalable than the ACK-based approach [7], we consider NAK as the retransmission request in this paper. In NAK-based retransmission mechanism, no servers or receivers have the knowledge of whether any packet has been successfully received by all receivers. As a result, only a limited number of packets can be kept in the caches at repair servers, designated receivers, and even the sender. The study on buffer management issue may have benefits in two aspects. First, the performance of local loss recovery depends on how many lost packets can be retrieved locally. Second, for the two-level caching systems in designated receivers or the sender where packets can be retrieved from the cache or the disk, improving buffer management can reduce the number of data retrieval from the disk. From another angle, we can achieve the same performance but with a smaller buffer or even less retransmission sites with a better caching policy.

In loss recovery schemes, caching policy is employed to determine the condition to accept new arriving packets and the time for caching those packets [8]–[11]. Generally, a good caching policy should always cache those packets which are most probably to be retrieved in the near future. Thus, a good caching policy can significantly enhance the capability of recovering lost packets locally. As a result, loss recovery latency can be reduced and meanwhile the network resource utilization can be improved. Thus, caching policy is actually an essential element of the other buffer management issues such as cache allocation policy and buffer partitioning policy [12]–[14]. In this paper, we will focus on the caching policies and their influences on the performance of loss recovery.

Existing loss recovery schemes employ three categories of
caching policies: First-In-First-Out (FIFO), Probabilistic FIFO (P-FIFO), and Timer-based Caching Policy (TBCP) [8]–[11]. For example, simple FIFO is used for caching received data at repair servers in AER/NCA [4]. In ARM [3], the intermediate routers adopt the timer-based caching policy where the caching time is set to the round-trip time (RTT) from the sender to the farthest receiver. P-FIFO is also proposed for ARM in [9]. In the recent work on ARM [11], the authors compare the performance of local loss recovery with different caching policies, including FIFO, P-FIFO, and TBCP. However, some issues have not been well addressed in existing research on caching policies, which are discussed below.

1). As a component of respective loss recovery scheme, most of the caching policies are targeted at minimizing the loss recovery latency. However, this objective could not be directly used for designing caching policies. First, other components of loss recovery scheme, such as the placement of retransmission points, the NAK aggregation/suppression mechanism, and the retransmission scheme (unicast, multicast, or subcast, etc.), affect the loss recovery latency at the same time [4], [13]. Thus, a caching policy minimizing loss recovery latency in one scenario may not be also good in another scenario. Second, the NAK aggregation/suppression mechanism may prevent the caching policy from achieving the design goal. When the mechanism is adopted, there is no exact information about the number of receivers requesting the data by a single NAK. Without that information, it is impossible to minimize the weighted average of loss recovery latencies of receivers.

2). All existing caching policies are based on some heuristic ideas without solid foundations [8]–[11]. There is no theoretical work which formulates the design problem formally and gives performance analysis of existing caching policies systematically. For example, they do not consider general network topologies including receivers with different loss probabilities and NAK latencies. Let us consider an example that the repair server buffers packets for two receivers simultaneously. Consider two topologies where loss probabilities and NAK latencies of receivers A and B are \{(0.001, 10 ms), (0.01, 30 ms)\} and \{(0.01, 10 ms), (0.001, 30 ms)\}, respectively. Since existing approaches do not consider the influence of different loss probabilities experienced by receivers, their caching policies cannot achieve good performance in both topologies.

Motivated by the above observations, in this paper we formulate the caching policy design as an optimization problem for maximizing the amount of data retrieved from the cache. The basic idea is that packets should be evicted from the cache not based on their arrival pattern but on the potential benefit that they might have. We propose a uniform measure, namely caching efficiency, in the optimization process. The solution of the problem we present in this paper, named Optimal Caching Time (OCT) algorithm, provides the optimal settings of caching time for caching policies. After that, we investigate two important features affecting the performance of caching policies: the accuracy and responsiveness of caching time control. Based on the investigation, we implement the OCT caching policy by directly comparing the caching time settings and the time of packets stored in the cache. We will demonstrate the benefits of OCT caching policy through analysis and simulations. More importantly, the model we use in this paper is very general and thus our idea can be applied to other areas as well, such as wireless networks, ad hoc networks, and else where the caching policy can have significant impact on the overall performance.

The rest of this paper is structured as follows. In Section II, we review related work on caching policies for local loss recovery. Next, we formulate the optimization problem of caching policies in Section III. In Section IV, we propose Optimal Caching Time (OCT) algorithm and show that it solves the optimization problem. Furthermore, we analyze the performance improvement of OCT compared to existing approaches. In Section V, we implement the optimal caching policy with addressing the problems in P-FIFO and TBCP which harm the accuracy and responsiveness of caching time control. In Section VI, we use ns-2 simulations to demonstrate and validate the performance gains of OCT caching policy compared to existing approaches. Finally, Section VII concludes the paper.

II. RELATED WORK

A. First-In-First-Out (FIFO)

The caching policies used in existing reliable multicast protocols fall into three categories: First-In-First-Out (FIFO) [8], [11], probabilistic FIFO [9], [11] and timer-based caching policies (TBCP) [8], [10], [11]. In this section, we briefly review the related work on caching policy. The FIFO caching policy manages packets in an order of First-In-First-Out. When the cache is full, the oldest packet will be purged for accommodating the new arriving packet. FIFO is the simplest caching policy. However, it is still very useful in most of the environments owing to the following reasons. First, it is very easy to implement and can be applied to various applications. Second, it only needs the buffer occupation information for packet caching. Thus, it is very robust and the resources cost and processing power are minimized. Third, its disadvantage for loss recovery lies in the performance degradation caused by the premature flushing problem [8], [11]. When the packets arrive in a high rate, the caching time for each packet may be so short that cached packets are always flushed before NAKs arrive. Thus, we can expect that FIFO performs badly under heavy traffic load.

B. Probabilistic FIFO

Probabilistic FIFO was proposed for ARM to address the premature flushing problem of FIFO in [9], [11]. The main idea is to drop packets probabilistically so that each cached packet can stay in the cache for sufficient time for possible retrieval request by a NAK. For this purpose, the filter randomly picks the cached packets with probability \(P_{\text{Cache}}\). The control parameter \(P_{\text{Cache}}\) is calculated by:

\[
P_{\text{Cache}} = B/\left[\lambda \times t^{(1)}\right],
\]

where \(\lambda\) is the average packet arrival rate, \(B\) is the buffer size, and \(t^{(1)}\) is equal to the average NAK latency for the first
works and the same setting also works for the timer-based approach. In their simulations, they also found that $RTT_x = 1.2$ might be the best setting. However, they have not explained the reason why this setting works and the same setting also works for the timer-based approach. In Section IV, we will explain the observation theoretically using our theory and we will find that the best setting varies with different scenarios.

### C. Timer-based Caching Policy (TBCP)

Basically the Timer-based Caching Policy, also known as the timer-based approach [8], [10], [11], uses timers to regulate the caching time for all packets. When the caching timer expires, the corresponding packet is pushed out. New arriving packets are only accepted into the cache when required buffer space is available. Otherwise, arriving packets are rejected to be cached. Another variant of TBCP is to maintain the expired packets until they are replaced by new arriving packets for full utilization of the cache. In fact, the number of data packets which can be cached has also been determined when the caching time configurations are given. Thus, the main design problem of TBCP is how to set the caching time. Generally, longer caching time may indicate more NAKs arrive before the cached packets expire. However, the number of cached packets is reduced due to the buffer limit. Thus, a good caching policy should find a balance between these two considerations.

A simple heuristic of the TBCP is to infer the caching time from the round-trip time (RTT) or the NAK latency. However, there is no solid work which suggests how to set the caching time based on RTT. In [8], the authors tried to use the mode, median, and mean of the NAK latency as the caching time. They concluded that TBCP does not significantly improve the performance of FIFO if the NAK latency is highly variable over a large range of values. The reason is that a “good” caching time which optimizes the performance cannot be determined in such cases. However, a more optimistic conclusion is drawn by Yeung and Wong [11]. In their paper, they use the following equation for setting the caching time $t_c$:

$$t_c = RTT \times RTT_x + \frac{1}{\lambda \times (1 - p)}$$  \hspace{1cm} (3)

where $RTT$ is the round-trip time to the furthest receiver, $p$ is the loss probability experienced by that receiver and the last term is the time for that receiver to be able to detect the lost packet. They observed that the optimal caching time can be approximately approached with $RTT_x = 1.2$. However, they again have not explain in principle why $RTT_x = 1.2$ is the best setting.

### III. FORMULATION OF CACHING POLICY DESIGN

We consider a caching system for loss recovery purpose in reliable multicast as shown in Fig. 1. Data packets arrive at the system with average arrival rate $\lambda$. The cache manager determines whether to drop a data packet or cache it for possible data retrieval in the future. After some time, data retrieving requests may arrive. If the caching time of a packet is longer than its request latency, the data can be retrieved successfully. In this paper, we consider that data are requested explicitly by NAKs and there are no ACKs indicating the outdating of packets. Upon receiving requests, the cache manager checks whether the requested data is stored in the cache. We call it a cache hit if some data matches the request or call it a cache miss otherwise. We denote by $h$ the cache hit ratio which is defined as the ratio of the number of data responses and the number of data requests. With a cache hit, the caching system retransmits the requested data. The data retrieval rate of the caching system is defined as the amount of output data per unit time.

For caching policy design, we assume that a certain amount of buffer has been allocated to a multicast group. We also assume that the caching system has no knowledge about whether an arriving data packet will be requested or has a higher probability to be requested. Thus, all arriving packets have equal priority to be cached. If we consider the caching times of the dropped packets to be zero, a caching policy determines the caching times of all arriving packets. Now we are focused on the relationship between the caching times of arriving packets and the performance of the caching system. For this purpose, we evaluate the performance of the caching system using utility functions based on its contribution to the loss recovery scheme. Let $t_i$ denote the caching time for packet $i$ and the utility function $U_i(t_i)$ denote the contribution of caching packet $i$ to the performance of the loss recovery scheme. The design objective can be stated as maximizing the caching utility of the loss recovery buffer $U_B$:

$$\text{maximize } U_B \equiv \sum_i U_i(t_i).$$  \hspace{1cm} (4)

subject to $\sum_i (s \times t_i) \leq s \times B \times T$, \hspace{1cm} (5)

where $s$ is the packet size, $B$ is the buffer capacity measured in packets, and $T$ is the working time of the caching system. We can categorize those packets with the same caching time in a specific class/group. The number of classes can be equal to the total number of packets if the caching times of all packets are different. Let $n_k$ denote the number of packets in class $k$.

![Fig. 1. A caching system](cache.png)
Accordingly, the design objective can be expressed as:

$$\text{maximize } U_B = \sum_{k} n_k U_k(t_k).$$

(6)

Given fixed $s$, $B$ and $T$, we transform (6) to

$$\text{maximize } \frac{U_B}{s \times B \times T}.$$

(7)

Now we define caching efficiency of a given amount of buffer space as the average caching utility achieved by a unit buffer space per unit time. Accordingly,

$$Y_B \triangleq \frac{U_B}{s \times B \times T} \quad \text{and}$$

$$Y_i(t_i) \triangleq \frac{U_i(t_i)}{s \times t_i},$$

(8)

(9)

where $Y_B$ is the caching efficiency of the whole buffer and $Y_i$ is the caching efficiency of packet $i$. Combining (6)–(9) leads to

$$\text{maximize } Y_B = \sum_{k} \left( \frac{n_k t_k}{B \times T} \cdot \frac{U_i(t_i)}{s \times t_i} \right) = \sum_{k} a_k Y_i(t_k),$$

(10)

where $a_k$ is the buffer share of class $k$:

$$a_k = \frac{n_k t_k}{B \times T}.$$  

(11)

We can find from (10) that the caching efficiency is a performance metric which links the performance of caching policy with the performance of caching a single packet. This property makes caching efficiency valuable for making caching decisions. Accordingly, we have Optimization Rule 1, based on which, we can design an algorithm for determining the optimal caching time of packets which can achieve the design objective theoretically in Section IV.

**Optimization Rule 1**: The optimal caching policy should always try to cache packets so as to maximize their caching efficiencies.

As the contribution of the caching system to the performance of the loss recovery scheme, $U_i$ is affected by several factors such as the difference between loss recovery latencies if the packet is locally retransmitted or retransmitted from upstream nodes and the number of beneficiaries of the local retransmission. However, it is very difficult, even if not impossible, to measure how many receivers request the data and/or benefit from the local retransmission, due to the NAK suppression/aggregation and scoped retransmission mechanisms (Referring to Section I). Thus, we simplify $U_i$ by using the amount of data retrieved from the caching system. This implies that each packet retrieved from the caching system contributes equally to the performance of loss recovery scheme. Moreover, it also implies that the caching system does not bias against packets with larger packet size. In this case, the design objective of the caching system is equivalent to maximizing the data retrieval rate of the caching system.

The above objective matches well the objective to minimize the mean number of additional retransmissions from upstream nodes adopted in [8]. Moreover, it is also consistent with other performance metrics used in caching system design and loss recovery schemes. With other conditions being equal, more data retrieved from the caching system means a higher cache hit ratio and more lost packets being locally recovered. Thus, loss recovery latency and sender’s retransmission load can be reduced and network resources utilization can be improved [4], [14].

By simplifying $U_i$ using the amount of data retrieved from the caching system, we find that the caching efficiency is useful in the design and implementation of caching policies. In fact, caching efficiency for caching a packet is the hit frequency of those cache space according to (9) and the caching efficiency of the whole buffer is the average hit frequency of the buffer space. In the following sections, we will investigate the packet caching efficiency and propose an algorithm to solve the optimization problem. Additionally, we will show that caching efficiency is also valuable in evaluating the performance of various caching policies.

**IV. OPTIMIZATION OF CACHING TIMES OF PACKETS**

**A. Packet Caching Efficiency**

We consider a general system model that a number of receivers send NAKs to the repair node where the caching system is located. Suppose a data packet arrives at the caching system which has a chance to be lost before reaching the receivers. Since the caching system may receive NAKs requesting data which has not reached the caching system, we exclude such NAKs from the so-called valid NAKs. We are interested in the packet request ratio $q$ defined to be the ratio between the number of valid requests received and the number of data packets arrived. Another point of interest is the distribution of the request latency defined as the duration from the arrival of data packet $i$ to the arrival of its request. Suppose the NAK latency is randomly distributed but the distribution is deterministic. We define the cumulative distribution function of data retrieval requests $F(t)$ as the probability that the NAK latency is smaller than or equal to $t$ seconds. The corresponding probability density function of data retrieval requests is denoted as $f(t)$. The expected amount of retrieved data if caching a packet for $t$ seconds will be

$$U = q \cdot s \cdot F(t).$$

(12)

Accordingly, the caching efficiency is given by

$$Y = \frac{U}{s \cdot t} = q \cdot \frac{F(t)}{t}.$$  

(13)

From the equation we can see that the caching efficiency is only related with packet request ratio and the NAK latency. Moreover, the optimal caching time (OCT) $t_o$ which maximizes the packet caching efficiency is only determined by $F(t)$.

Now we show some examples of packet caching efficiency in local loss recovery schemes to illustrate the caching efficiency. In these examples, we assume that retransmitted packets are never lost. Accordingly, loss probability $p_l$ of every receiver is also the packet request ratio $q_l$, and $F(t)$, the distribution of packet retransmission request, is also the distribution of the round-trip time from repair node to receiver.
First we consider the simple example (Case I) that there is only one downstream receiver experiencing packet loss. The cdf and pdf of retransmission request are shown by the solid curves in Fig. 2. The NAK latency follows a Weibull distribution $F_0(t)$ and packet loss probability $p_0 = 0.001$. We can easily obtain the caching efficiency from (13):

$$Y(t) = p_0 \cdot \frac{F_0(t)}{t}.$$  \hspace{1cm} (14)

The solid curve of Fig. 3 shows the caching efficiency of Case I as a function of the caching time. As the caching time increases, the caching efficiency increases at first. We can observe that there exists a peak value where the maximal caching efficiency is achieved with the optimal caching time $(OCT) t_o = 50 \text{ ms}$. Note that OCT is much longer than the average NAK latency (equal to 37 ms). When the caching time grows longer than the optimal caching time, the caching efficiency decreases approximately hyperbolically since $F_0(t)$ increases slightly. When the time grows to infinite, the caching efficiency approaches to zero.

Next we consider the cases of multiple lossy receivers. In Case II.A, one repair node serves for $n$ receivers with the same RTT distribution $F_0(t)$. Their packet losses are independently distributed with equal probability $p_0$. Suppose there is no NAK suppression mechanism and the repair node unicasts the retransmission to the request receiver. In this way, we can obtain the total amount of retrieved data

$$U(t) = n \cdot p_0 \cdot s \cdot F_0(t).$$  \hspace{1cm} (15)

The corresponding caching efficiency is

$$Y(t) = n \cdot p_0 \cdot \frac{F_0(t)}{t}.$$  \hspace{1cm} (16)

Thus, the optimal caching time $t_o$ will be the same as that of Case I but the maximal caching efficiency will be $n$ times greater.

For the second multi-receivers case (Case II.B), suppose the NAKs can be suppressed or aggregated before forwarding to the cache and the repair packet is multicast to all downstream receivers so that for one data packet only the first NAK reaches the caching system. With other conditions being the same as Case II.A, we have

$$F(t) = 1 - [1 - F_0(t)]^n,$$  \hspace{1cm} (17)

and

$$q = 1 - (1 - p_0)^n.$$  \hspace{1cm} (18)

From (13) we can obtain

$$Y(t) = [1 - (1 - p_0)^n] \cdot \frac{1 - [1 - F_0(t)]^n}{t}.$$  \hspace{1cm} (19)

The dotted curves of Fig. 2 show the cdf and pdf of NAK latency with $n = 2$. We can observe that the curves have similar shapes as those with $n = 1$ (Case I). However, the NAKs arrive earlier and their arrivals are more concentrated around the peak arrival time. Correspondingly, the cdf of NAK latency increases more rapidly. Its caching efficiency shown by the dotted curve of Fig. 3 also has similar shape as that of Case I. However, both the increase and the decrease of caching efficiency becomes steeper, the maximal caching efficiency is higher, and the maximal caching efficiency can be achieved with less caching time. As expected, the caching efficiency is higher with a larger group, i.e., it is more efficient to cache a packet being multicast to a larger group.

Besides the above homogeneous scenarios, now we consider a heterogeneous scenario (Case III). There are two receivers at the downstream of the repair node with loss probabilities 0.0015 and 0.001. The second receiver has a longer propagation delay. As a result, the pdf of its NAK latency is shifted to the right of the $x$ axis for 60 ms. Suppose there is no NAK suppression mechanism and the repair node unicasts the retransmission to the request receiver. The distribution of NAK latency seen by the caching system is shown in Fig. 4. The packet caching efficiency is shown by Fig. 5. The optimal caching time $t_o = 50 \text{ ms}$ maximizes the caching efficiency. However, the caching efficiency does not decrease monotonically to zero after caching time grows larger than $t_o$. A second maximum of caching efficiency exists to the right.
Actually, multiple maximums may appear frequently for a region with multiple receivers. This observation has an essential impact on solving the optimization problem.

Now we define an important concept residual caching efficiency as a complement of caching efficiency to analyze the situation after a packet has been cached for $\tau$ seconds. The time $\tau$ is called the age of a packet. If this packet is cached for another $t - \tau$ duration, the caching efficiency during these $t - \tau$ seconds is the residual caching efficiency. Using previous notations, the caching utility for the period $(\tau, t)$ is

$$U(\tau, t) = q \cdot s \cdot [F(t) - F(\tau)] .$$  \hspace{1cm} \text{(20)}

The residual caching efficiency is thus equal to

$$Y(\tau, t) \triangleq \frac{U(\tau, t)}{s \cdot (t - \tau)} = q \cdot \frac{F(t) - F(\tau)}{t - \tau} .$$ \hspace{1cm} \text{(21)}

Fig. 6 shows the residual caching efficiency for Case I after the packet has been cached for $\tau$ unit time, where $\tau$ equals to 0, 10, 20, 30, 40, 50, 60 ms. The packet with age zero is actually a new packet. When the age $\tau$ is small, the residual caching efficiency increases with the caching time $t$ at first. After reaching a peak value, the residual caching efficiency decreases. When the time grows to infinite, it decreases to zero. When $\tau$ is large, the residual caching efficiency decreases monotonically until it approaches zero when the time grows to infinite.

We can obtain the maximal residual caching efficiency (MRCE) as a function of $\tau$:

$$A(\tau) = \max_t Y(\tau, t) .$$ \hspace{1cm} \text{(22)}

Fig. 7 shows the maximal residual caching efficiencies of Cases I and II.B. We can observe that the maximal residual caching efficiency increases with the caching time at first before reaching a peak value. After that, it decreases rapidly to zero. Similarly, Fig. 8 shows the maximal residual caching efficiency of Case III. Due to the heterogeneity of receivers,
there are two peak values of maximal residual caching efficiency.

B. Optimizing the Caching Time

Maximal residual caching efficiency provides an effective method to decide which packet should be dropped when necessary. When packet dropping is required, there are a number of packets of different ages (including the new packet) available for caching. These packets may have different MRCEs. Since a higher MRCE indicates a higher achievable caching efficiency, it is reasonable to cache a packet with a higher MRCE. Thus, we have Optimization Rule 2.

Optimization Rule 2: When packet dropping is required, the packet with the lowest MRCE, even if it is the new arrival packet, should be dropped.

Take the MRCE of Case I as shown by the solid curve in Fig. 7 as an example. Suppose there are 61 packets with ages ranging from 0, 1, 2, ⋯ to 60 ms available to be cached. However, the buffer limit is 60 and one packet has to be dropped. Accordingly, the oldest packet of age 60 ms should be dropped since it is the packet with the lowest MRCE. Suppose a different situation where we must drop a packet among 41 packets of ages ranging from 0, 1, 2, ⋯ to 40 ms. The new arriving packet with age zero should be dropped because of its lowest MRCE. For the two cases shown in Fig. 7, either dropping the new arriving packet (Drop-Tail) or dropping the oldest packet (Drop-Front) is the optimal decision. In fact, this observation is the underlying principle on which existing caching policies are designed. However, the above observation is not always true for the heterogeneous case shown in Fig. 8. In that case, sometimes dropping the packet in the middle of the queue is the optimal decision. From Figs. 7 and 8, we can observe that there exists an age \( \tau \) which satisfies \( A(\tau) = A(0) \) and \( \tau' > \tau \) is a necessary condition for \( A(\tau') < A(0) \). Since \( A(\tau') < A(0) \) is a necessary condition for dropping an old packet upon the arrival of new data, we call it push-out age (POA) denoted by \( \tau_o \).

The above example suggests that we can try to optimize the caching policy by comparing the MRCEs of packets. However, this method needs to calculate the MRCE of each packet and thus requires a high computation load. Now we will try to find another method based on the optimal caching time of packet caching efficiency. We start from investigating the relationship between the push-out age \( \tau_o \) and the optimal caching time \( t_o \). From the definition of residual caching efficiency in (21), we have

\[
Y(0, t) = \frac{t}{t - \tau} \cdot Y(0, \tau) + \frac{t - \tau}{t} \cdot Y(\tau, t).
\]

(23)

From

\[
Y(0, t_o) = \max_t Y(0, t),
\]

(24)

we can obtain

\[
Y(0, t) \leq Y(0, t_o), \forall t \geq t_o.
\]

(25)

Combining (23) and (25) leads to

\[
Y(t_o, t) \leq Y(0, t_o), \forall t \geq t_o.
\]

(26)

Moreover, from (24) we have

\[
Y'(0, t)\big|_{t_o} = 0.
\]

(27)

Integrating (21) to the above equation leads to

\[
q \cdot \frac{t F'(t) - F(t)}{t^2} \big|_{t_o} = 0.
\]

(28)

Thus we have

\[
F'(t_o) = \frac{F(t_o)}{t_o}.
\]

(29)

Additionally,

\[
Y(t_o, t) = q \cdot \frac{F(t) - F(t_o)}{t - t_o}.
\]

(30)

leads to

\[
\lim_{t \to t_o} Y(t_o, t) = q \cdot F'(t_o).
\]

(31)

Combining (29) and (31) yields

\[
\lim_{t \to t_o} Y(t_o, t) = q \cdot \frac{F(t_o)}{t_o} = Y(0, t_o).
\]

(32)

From (26) and (32) we have

\[
A(t_o) = \max_t Y(t_o, t) = Y(0, t_o) = A(0).
\]

(33)

Meanwhile, from (23) and (24) we know that \( \tau' > t_o \) is a necessary condition for \( A(\tau') < A(0) \). Thus, the optimal caching time \( t_o \) is equal to the push-out age \( \tau_o \). This finding is also shown in Figs. 3, 5, 7 and 8.

In fact, the equivalence of optimal caching time and push-out age indicates that Optimization Rule 1 based on maximizing the caching efficiency and Optimization Rule 2 based on comparing the maximal residual caching efficiencies of packets are equivalent. According to Rule 1, we should try to cache packets for the optimal caching time \( t_o \). From Rule 2, we know that we can drop a packet if its age is greater than \( \tau_o \). Thus, we can choose the suitable one for making caching decisions in various situations. Based on the above investigations, we can try to solve the optimization problem based on (10). To maximize \( Y_B \), we should maximize the
number of packets which can achieve the highest caching efficiencies. Suppose the optimal caching time $t_0$ maximizes $Y(t)$, i.e., $Y(t_o) = \max Y(t)$. We categorize all packets with caching time $t_o$ as class 0. Accordingly, $t_0 = t_o$ and $Y_0 = Y(t_o)$. Thus, $\max Y_B = Y(t_o)$ if and only if the buffer is always occupied by packets belonging to class 0:

$$a_0 = \frac{n_0 \times t_0}{B \times T} = 1.$$  

(34)

As $n_0/T \leq \lambda$, from the above equation we have $B \leq \lambda \times t_0$. Thus, we have found the optimal solution for the optimization problem when $B \leq \lambda \times t_0$. If $B > \lambda \times t_0$, there is extra buffer space even after all packets have been cached for $t_0$ seconds. We consider this case as the optimization problem for the extra buffer space. From (10) we have

$$\text{maximize } Y_B = \frac{\lambda \times t_0}{B} Y_0 + \sum_{k>0} a_k Y_k^{(1)}(t_k),$$

where $Y_k^{(1)}(t)$ is the residual caching efficiency for packet with age $t_0$, i.e., $Y_k^{(1)}(t) = Y(t_0, t_0 + t)$. Notice that we consider the packets with age $t_0$ are “new” packets. Suppose $t_1^{(1)}$ maximizes $Y^{(1)}(t)$. We categorize all such packets with caching time $t_1^{(1)}$ as class 1. Accordingly, $t_1 = t_1^{(1)}$ and $Y_1 = Y(t_0, t_0 + t_1)$. Thus, $\max Y_B = a_0 Y_0 + a_1 Y_1$ if and only if:

$$a_1 = \frac{n_1 \times t_1}{B \times T} = 1 - \frac{\lambda \times t_0}{B}.$$  

(36)

As $n_1/T \leq \lambda$, from the above equation we have $B \leq \lambda(t_0 + t_1)$. Combining with the previous result for $B \leq \lambda(t_0)$, we have found the optimal solution for the optimization problem for $B \leq \lambda(t_0 + t_1)$. However, it is necessary to add some mechanism considering the case that the maximal value of $Y(t_k, t_k + t)$ may be approached when $t \to 0$. In this case, we can add a constraint that each class of packets must be cached at least for time $\delta$. Following this method, we can obtain the complete near-optimal solution for the optimization problem iteratively. Since the algorithm is based on searching the optimal caching time (OCT), we call it OCT algorithm (Fig. 9). After running the algorithm, the caching times of packets are randomly set to $t_0$ or $t_1$ with probability $a_0$ or $a_1$, respectively. The caching efficiency of the whole buffer is given by

$$Y_B = a_0(\lambda T_0/B)Y(T_0) + a_1(\lambda T_1/B)Y(T_1)$$

(37)

In real systems, the cdf of NAK latency is only known at a number of points (e.g., 50) due to the measurement limit. Thus, the computation load of OCT algorithm is acceptable.

C. Performance Improvement of OCT Algorithm

Given the caching times of all packets, we can calculate the caching efficiency of the loss recovery buffer according to the definition of (8). If the average used cache space measured in packets is $C$ and the caching times for all cached packets are equal to $t$, we have

$$Y_B = U_B \frac{s \cdot C \cdot t \cdot Y(t)}{s \cdot B \cdot T} = \rho \cdot Y(t),$$  

(38)

OCT Algorithm:

$k = 0$; $t_{-1} = 0$; while($\lambda k_{-1} < B$) {

Find $t_o \geq \delta$ so that $Y(t_o) = \max Y(t_{k-1}, t_{k-1} + t)$;

$t_k = t_{k-1} + t_o$;

$k + ++$;

} if($t_o == \delta$) {

$T_0 = T_k = B/\lambda$;

$Y_B = Y(T_0)$;

} else {

$T_0 = t_{k-2}$, $T_1 = t_{k-1}$;

$a_0 = (\lambda T_1 - B)/[\lambda(T_1 - T_0)]$;

$a_1 = (B - \lambda T_0)/[\lambda(T_1 - T_0)]$;

$Y_B = a_0(\lambda T_0/B)Y(T_0) + a_1(\lambda T_1/B)Y(T_1)$;

}

Fig. 9. OCT algorithm

where $\rho$ is the cache utilization ratio. If the cache is fully utilized,

$$Y_B = Y(t).$$  

(39)

It means that the caching efficiency of the buffer is equal to the packet caching efficiency. Now we analyze the caching efficiency of various caching policies, including FIFO, P-FIFO, and TBCP.

1) Caching Efficiency of FIFO: For FIFO, the cache is fully utilized and all packets are cached with time $t = B/\lambda$. Thus, the caching efficiency of FIFO will be

$$Y_B = Y(B/\lambda).$$  

(40)

It means that the caching efficiency of FIFO follows the curve of packet caching efficiency exactly.

2) Caching Efficiency of P-FIFO: For P-FIFO, the cache is always fully utilized and $Y_B = Y(t)$. If $B < \lambda \times t(1)$, packets are cached with a probability $P_{\text{cache}}$. Thus, the caching time of packets will be

$$T = B/(\lambda P_{\text{cache}}) = t(1).$$  

(41)

In this case, the caching efficiency of P-FIFO will be

$$Y_B = \begin{cases} Y(T) & \text{if } B \leq \lambda \times T \\ Y(B/\lambda) & \text{if } B > \lambda \times T. \end{cases}$$  

(42)

In the implementation of P-FIFO, the caching time for each packet may be randomly distributed as a consequence of the probabilistic caching mechanism. We call this problem as the inaccuracy of caching time control. However, this problem can be avoided by using a periodic dropping mechanism with the same dropping probability. In that case, the above analysis is still applicable to the caching time settings of P-FIFO. We will analyze the performance difference caused by the problem in Section V.

3) Caching Efficiency of TBCP: For TBCP, we first consider the policy for which a cached packet is flushed out upon the expiration of the cached packet. If the caching time is set to be equal to $T$, $B \leq \lambda \times T$ means that all buffer space is used
for caching packets. Thus, the caching efficiency of TBCP in this case will be $Y(T)$. Or if $B > \lambda \times T$, it means that some of the buffer space is wasted. Thus, the caching efficiency is equal to

$$Y_B = Y(T) \times \frac{\lambda T}{B}.$$  (43)

Now we consider the policy for which the expired packet is still retained in the cache until the new packet arrives. We can easily obtain the caching efficiency as

$$Y_B = \begin{cases} Y(T) & \text{if } B \leq \lambda \times T \\ Y(B/\lambda) & \text{if } B > \lambda \times T. \end{cases}$$  (44)

Obviously $Y(B/\lambda) > Y(T) \times \lambda T/B$ since $F(B/\lambda) > F(T)$. Thus, the second policy is more efficient than the first one. Hereafter we consider this policy as the default policy of TBCP. Please note that (44) has the same expression as (42). It means that the performance of P-FIFO can be equal to that of TBCP with (41) satisfied.

Now we can explain the observation in [11] that the performance of both P-FIFO and TBCP are the best when $RTT_x = 1.2$. Given the above relationship between the performance of P-FIFO and TBCP, we only need to find the configuration of $T$ which maximizes (44). According to the formula, $t_o$ is the optimal setting of caching time when $B \leq \lambda \times t_o$ as $Y(t_o)$ is the maximum of $Y(t)$. For the case that $B > \lambda \times t_o$, the same performance of $Y(B/\lambda)$ is achieved by configuring $T \leq B/\lambda$. If configuring caching time $T > B/\lambda$, $Y(T) < Y(B/\lambda)$ is true under the condition that caching efficiency decreases monotonically when $t > t_o$. This condition is likely to be satisfied for the homogeneous network topology employed in [11]. Thus, configuring the caching time according to $t_o$ can maximize the caching efficiency of the whole buffer. Based on the analysis, we know that the reason for the observation in [11] is that the caching time $T$ is set to be approximately equal to the optimal caching time $t_o$ with $RTT_x = 1.2$. Hence we can estimate $t_o$ for their simulation topology from (41):

$$t_o \approx 1.2 \times t^{(1)},$$  (45)

where $t^{(1)}$ is the average NAK latency for the first NAK or the average RTT from the repair node to the receiver. Moreover, we can draw conclusions that $t_o$ is always the optimal setting of $T$ in TBCP when $B \leq \lambda \times t_o$ and/or $Y(t)$ is a decreasing function of $t$ when $t > t_o$. Otherwise, a different optimal setting of $T$ may exist in the range that $T > B/\lambda$.

4) Performance Improvement of OCT Algorithm: Now we use Case III to illustrate the performance improvement of OCT algorithm compared to FIFO, P-FIFO, and TBCP. Since P-FIFO may achieve the same performance as TBCP, we plot the performance curves of OCT, FIFO, and TBCP in Fig. 10. The caching time of TBCP is set to be the RTT from the server to the furthermost receiver [3], [11]. We can observe that OCT algorithm always achieves the highest caching efficiency. When $B/\lambda < 110$ ms, OCT algorithm can improve the performance of caching system significantly.

V. IMPLEMENTING OCT CACHING POLICY

A. Measuring the Cumulative Distribution Function of NAK Latency

In this subsection we address the implementation of the OCT algorithm. First, we should measure the packet caching efficiency for various caching time. For this purpose, we measure the cumulative distribution function of NAK latency by the following method. We set a number of objective values of $F(t_i), f_i$, which are equally distributed within $[0, 1]$. We are trying to find the time $t_i$ with which $F(t_i) = f_i$. This can be realized by applying smoothing techniques. The basic idea is using the smoothing technique to find the probability that NAK latency is smaller than some time $t_x$ and then adjusting the time $t_x$ by comparing $F(t_x)$ and $f_i$.

$$\begin{array}{l}
\text{if}(t_{nak} \leq t_i)
\quad h_i = (1 - \alpha) \times h_i + \alpha \\
\text{else}
\quad h_i = (1 - \alpha) \times h_i
\end{array}$$

$$\begin{array}{l}
\text{if}(h_i < i/k \ & \text{&} \ & t_i < l_{nak})
\quad t_i = (1 - \alpha) \times t_i + \alpha \times l_{nak}
\quad t_i = \min(t_i, t_{i+1})
\end{array}$$

$$\begin{array}{l}
\text{else if}(h_i > i/k \ & \text{&} \ & t_i > l_{nak})
\quad t_i = (1 - \alpha) \times t_i + \alpha \times l_{nak}
\quad t_i = \max(t_i, t_{i-1})
\end{array}$$

Fig. 11 shows the core algorithm for measuring the cdf of NAK latency. We keep $k$ pairs of variables $t_i$ and $h_i$ with $i$ ranging from 1 to $k$, where $h_i$ is used to maintain the value of $F(t_x)$ for searching $t_i$. The objective value of $F(t_i)$ is set to be equal to $i/k$. During initialization, we set $h_i$ to 0.5 and $t_i$ to the first NAK latency $l_{nak}$ obtained. When a new $l_{nak}$
is obtained, we compare \( t_i \) and \( l_{nak} \) and use the following equation to recalculate \( h_i \):

\[
  h_i' = \begin{cases} 
    (1 - \alpha)h_i + \alpha & \text{if } l_{nak} \leq t_i \\
    (1 - \alpha)h_i & \text{if } l_{nak} > t_i 
  \end{cases} \tag{46}
\]

After obtaining the new \( h_i \), we compare it with the objective probability \( i/k \). If \( h_i \) is smaller than the objective probability and \( t_i \) is smaller than the new NAK latency, we increase \( t_i \) by \( \alpha \times (l_{nak} - t_i) \). However, the new \( t_i \) should not be greater than \( t_{i+1} \). Otherwise if \( h_i \) is greater than the objective probability and \( t_i \) is greater than the new NAK latency, we decrease \( t_i \) by \( \alpha \times (t_i - l_{nak}) \). However, the new \( t_i \) should not be smaller than \( t_{i-1} \). Therefore, we can expect that \( t_i \) is approaching the time with which \( F(t) = i/k \) with the arrival of NAKs.

Now we have obtained the cumulative distribution function of \( F(t) \) approximately. With the measured values of \( t_i \) and \( h_i \), we can calculate the caching efficiency directly by \( h_i/t_i \) and find the maximal \( h_i/t_i \) and the corresponding \( t_{max} \). According to our mechanism, \( t_{max} \) can be used for representing the optimal caching time \( t_o \) in theory. Next, we can follow the OCT algorithm shown in Fig. 9 to find other caching time configurations. In the implementation of OCT algorithm, \( \delta \) is actually the difference between \( t_i \) and \( t_{i+1} \) and the residual caching efficiency is \( (h_j - h_i)/(t_j - t_i) \) for caching packet from \( t_i \) to \( t_j \) \((j > i)\). In the implementation, we use the parameters \( k = 50 \) and \( \alpha = 0.01 \). In our simulation experiments, the mechanism works well.

### B. Problems in Caching Time Control of P-FIFO and TBCP

By implementing the OCT algorithm with the above mechanisms, we can obtain the configuration of caching times for packet caching. Now we consider the caching time control issue which affects the actual performance we can achieve with a caching policy. The control mechanism for caching time determines whether the caching policy can adjust the caching time promptly and whether the caching time can be controlled accurately. In the following parts, we elaborate the problems in existing caching time control mechanisms.

1) The Accuracy of Caching Time Control: Further Analysis on Caching Efficiency of P-FIFO

We analyze the performance of P-FIFO with probabilistic caching mechanism. Arriving packets may be dropped when \( P_{Cache} < 1 \), where \( P_{Cache} = B/(\lambda T) \) when \( B < \lambda T \). Now we consider an arbitrary packet \( i \) which arrives at the caching system and is accepted into the buffer. According to the caching policy, packet \( i \) will be flushed after \( B \) new packets are accepted into the buffer. We are trying to find how many new arriving packets are needed so that \( B \) packets are successfully cached. We denote by \( d \) the number of required new arriving packets \((d \geq B)\). \( d \) follows the Pascal Distribution:

\[
P(X = d) = \binom{d - 1}{B - 1} p^B (1 - p)^{d - B}, \tag{47}
\]

where \( p = P_{Cache} \) and \( q = 1 - p \). Its mean and variance are given by

\[
  \mu = B/p = \lambda T \quad \text{and} \quad \sigma^2 = Bq/p^2 = (\lambda T - B)\lambda T/B. \tag{48}
\]

\[
  2) The Responsiveness of Caching Time Control: Since the distribution of NAK latency may vary according to the network dynamics, the caching time configurations need to catch up with the variations instantly. As a result, it is important to investigate the responsiveness of caching time control. P-FIFO is insensitive to the shift of NAK latency since it has no caching time control mechanism. For TBCP based on caching timers, the adjustment of caching time only takes effects on new arriving packets. In other words, the packets already in cache are not affected by the change of caching time. Thus, the caching time control in TBCP is delayed. For P-FIFO, the adjustment of caching probability takes effect on accepting new arriving packets into the cache. As a result, the caching times of packets managed by P-FIFO are affected by the adjustment immediately but partially.

C. OCT Caching Policy

To address the problems in P-FIFO and TBCP, we implement the OCT algorithm by comparing the caching time settings and the age of packets directly. This caching time control mechanism is not only accurate but the caching times of packets can be adjusted promptly. In the proposed OCT caching policy, a field is used to maintain the arrival time of each cached packet. In this way, we can calculate the age of packets easily when necessary. When the buffer is full, we need to determine the packet to be dropped upon the arrival of new packet. Suppose the age of packet \( i \) is \( t_i \) at that time. The OCT caching policy compares \( t_i \) with \( t_o \). If \( t_o \geq t_i \), the oldest packet is dropped else we check if there are packets which have been cached over time \( T_0 \). If so, we drop the youngest packet whose caching time is longer than or equal to \( T_0 \). Otherwise, we compare \( t_{max} \) with the optimal caching time \( t_o \). We will drop the oldest packet if \( t_{max} \geq t_o \) or drop the new arriving packet if \( t_{max} < t_o \).

The above OCT caching policy is designed with the following considerations. First, all packets must be cached over
the optimal caching time $t_o$ according to Optimization Rule 1. Second, we try to cache packets so that their caching times follow the caching time settings indicated by the OCT algorithm. Additionally, our design is also consistent with Optimization Rule 2. In the next section, we will demonstrate the benefits of the caching time control mechanism of OCT caching policy and the OCT algorithm separately.

VI. PERFORMANCE EVALUATION BY SIMULATIONS

In this section, we use ns-2 [15] simulations to evaluate the performance of various caching policies, including FIFO, TBCP, P-FIFO, and OCT. For performance comparison, we use caching efficiency, cache hit ratio and loss recovery latency as performance metrics. Loss recovery latency is the time elapsed between the first detection of a packet loss at a receiver and the receipt of the first repair for that packet. The last two performance metrics are widely used in the literature [4], [9], [11].

We run Pragmatic General Multicast (PGM) [16] implemented with ns-2.26 [15] as the reliable multicast protocol. The buffer management function is added as a component of the Designated Local Repairer (DLR). DLR is also known as the repair server (RS). When RS receives a NAK, it first tries to retrieve the requested data packet from the buffer. If successful, the retrieved data packet is sent to the receivers using the retransmission mechanism in PGM. Otherwise, RS may send the NAK to upstream nodes with the NAK suppression and forwarding mechanism in PGM.

First, we use a symmetric topology shown in Fig. 12 to examine the performance of various caching policies. For the simulation, the multicast sender transmits data to four receivers. Data traffic is generated at the sender by a Poisson process with a rate of 448 Kbps. Each packet has a length of 226 Bytes including the header. All links cause a transmission delay of 20 ms. All packets transmitting through the links connecting receivers with repair servers (tail links) may be randomly lost with a probability of $p = 0.01$. At the same time, eight TCP sessions are competing with the multicast session on the tail links so that each tail link is shared by two TCP sessions and the multicast session. Accordingly, packets experience both random losses and congestion losses. Other links are assumed to be loss free.

Fig. 13 shows the caching efficiencies of the compared caching policies. We can observe that OCT and FIFO achieve the highest and lowest caching efficiencies, respectively. OCT significantly outperforms all other caching policies with $B < 40$. This is caused by the optimized caching time settings in OCT. The performance difference between OCT and P-FIFO decreases with the buffer size and the difference diminishes after buffer size grows larger than 50. Since OCT, P-FIFO, and TBCP avoid the premature flushing problem in FIFO, they all improve the performance of FIFO significantly when the buffer size is small. We also observe that TBCP performs not so good as OCT and it degrades to FIFO as buffer size increases to 30. That is caused by the poor responsiveness property of TBCP analyzed in Section V-B.

Figs. 14 and 15 show the cache hit ratios and loss recovery latencies of the caching policies, respectively. The performance of the caching policies is consistent with that in terms of caching efficiency. For example, OCT achieves the best
performance in terms of the highest caching efficiency, the highest cache hit ratio and the lowest loss recovery latency. From another angle, a good caching policy decreases the buffer requirements of repair nodes for local loss recovery. For example, to maintain the cache hit ratio above 95%, OCT and P-FIFO decrease the buffer requirements of FIFO from 60 to 50.

Next, we use a heterogenous topology to examine the performance of the caching policies. We design this topology by modifying the properties of links from the two repair servers to receivers based on the previous topology in Fig. 12. For each repair server, the delay and loss probability for the two receivers are set to (10 ms, 0.025) and (100 ms, 0.005). Fig. 16 shows the caching efficiencies of the four caching policies. Again, we observe that OCT achieves the highest caching efficiency. The performance improvement of OCT compared to other policies is more significant. Different from the observations for the homogeneous topology, FIFO achieves the second highest caching efficiency with $10 < B < 40$. This difference is caused by the different curves of packet caching efficiency. Moreover, TBCP performs worst in all caching policies as its poor responsiveness property degrades the performance more severely. Figs. 17 and 18 show the cache hit ratios and loss recovery latencies of the compared caching policies. The observations are also consistent with the performance of caching policies in terms of caching efficiency. In summary, the heterogeneous topology highlights the performance gains brought by the OCT algorithm. We have also conducted the simulations with a CBR source and observed similar observations.

VII. CONCLUSIONS

Loss recovery schemes are employed to guarantee reliable data disseminations for multicast transport services. For scalability considerations, most of the packet losses are recovered by retransmissions from the repair servers or designated
receivers in local loss recovery schemes. In those schemes, data packets can be cached at those nodes for future local loss recovery. In this paper, we have addressed the caching policy issue which determines the packets to be cached and the caching time for cached packets to optimize the performance of loss recovery based on a general model.

We have formulated the caching policy design as an optimization problem for maximizing the amount of data retrieved from the buffer. To solve the optimization problem, we proposed an algorithm called Optimal Caching Time (OCT) for configuring the caching time of packets. Furthermore, we analyzed the performance improvement of OCT algorithm compared to all existing settings of caching time in FIFO, Probabilistic FIFO (P-FIFO), and Timer-Based Caching Policy (TBCP). After that, we realize the OCT caching policy by directly comparing the caching time settings and the age of packets. This proposed caching time control mechanism avoids the problems in P-FIFO and TBCP that damage the accuracy and responsiveness of caching time control. Finally, we use ns-2 simulations to demonstrate and validate the performance gains brought by the OCT caching policy. The simulation results show that OCT caching policy can improve the performance significantly in terms of higher caching efficiency and cache hit ratio and lower loss recovery latency for both homogeneous and heterogeneous scenarios.

The proposed OCT algorithm for optimizing caching policies is a general solution. It can be used in various reliable multicast protocols, including the recently emerged Application Layer Multicast (ALM) approaches such as REALM [17], to provide performance improvement. Moreover, the mathematical formulation leading to the optimization rules and the OCT algorithm may provide some insights for general caching systems such as the cache for wireless data disseminations.

REFERENCES