ReBARC: Recovery-Budget-Aware In-Network Retransmission Control in ICN

Kazuhisa Matsuzono and Hitoshi Asaeda
National Institute of Information and Communications Technology (NICT), Japan.
Email: {matsuzono, asaeda}@nict.go.jp

***Abstract*—Information-centric networking (ICN) is promising in fulfilling future application requirements, such as high-quality, low-latency video streaming. Because data loss considerably affects the user experience of such applications, rapid loss recovery is central to ICN solutions. ICN works with hop-by-hop data transmission, which is beneficial for in-network data retransmission. However, sophisticated in-network retransmission mechanisms that can effectively and efficiently recover data losses have not been fully studied. In this paper, we propose recovery-budget-aware in-network retransmission control (referred to as *ReBARC*) in ICN. ReBARC attempts successful loss recovery while satisfying latency requirements and suppressing duplicate data receptions in a fully distributed manner. Through receiver-driven hop-by-hop communication, receivers (e.g., consumers) allocate a maximum allowable recovery delay (recovery budget) to each link. For effective budget allocation, consumers recognize the budget consumption and request at each link in a network telemetry manner. Based on the allocated budget, each node adjusts the retransmission time-out value and methods of relaying recovery data without exchanging additional control messages with other nodes. Through comprehensive experimental evaluations, we confirm that ReBARC maintains higher video quality than existing methods by achieving more successful loss recoveries and fewer duplicate data receptions.**

I. Introduction

The growth and meaningfulness of applications using high-quality video streaming have been evident in recent years. For instance, low-latency video feedback is needed in common live streaming applications and in teleoperation of robots and unmanned aerial vehicles. Acceptable video streaming in these applications requires network bandwidths of 10–100 Mbps and end-to-end latencies below 200 ms [1], [2]. However, owing to such latency requirements, they often suffer from data losses owing to network impairments.

Lost data are commonly recovered through end-to-end retransmission, such as in the transmission-control protocol (TCP) and QUIC protocol [3], where the involved data sender (or publisher) initiates data retransmission based on acknowledgements (ACKs) from the receiver (or consumer). Another technique is hop-by-hop retransmission, where the node before the hop where data loss occurs initiates data retransmission based on some feedback from the adjacent downstream node (e.g., ACKs and negative ACKs [4]). By recovering data loss locally, hop-by-hop retransmission is expected to accelerate loss recovery. According to early studies [5], [6], hop-by-hop retransmission has better latency performance than end-to-end retransmission.

ISBN 978-3-903176-72-0 © 2025 IFIP

Information-centric networking (ICN) [7], including named data networking (NDN) [8] and CCNx [9], is a new communication paradigm that improves content delivery efficiency. ICN adopts a consumer-driven communication model; a consumer initially sends a data request message (called *interest*), and a node (e.g., router) having the requested content returns as the *data*. When an interest or the corresponding data are dropped en route, the consumer must retransmit the interest packet (hereinafter called *recovery interest*) to recover the lost data. Most existing studies [10]–[13] adopt consumer-based retransmission models; here, the consumer observes the time from sending an interest to obtaining the corresponding data (i.e., round-trip time (RTT)), and sets a certain retransmission time-out (RTO) value based on the observed RTT to determine the timing of issuing a recovery interest to obtain the lost data.

Supporting hop-by-hop transport and in-network caching at intermediate nodes (routers) [14], ICN can accelerate retransmission-based recovery by fetching lost data from data caches at intermediate nodes [15]–[17]. This communication model enables a flow-aware transport control paradigm [18], where intermediate nodes recognize the latency requirement of the consumer application and perform hop-by-hop retransmission as necessary.

However, the nodes on the path cannot easily trigger in-network retransmission in a timely manner and cannot efficiently avoid the unnecessary transmission of recovery interests after detecting the lost data. Generally, ICN does not include loss detection and feedback mechanisms (e.g., ACKs); nevertheless, implementing an ACK mechanism for each link is infeasible because high-bandwidth streaming flows considerably increase bandwidth consumption.

In this paper, we design a new class of transport protocols: recovery-budget-aware in-network retransmission control (referred to as *ReBARC*) in ICN. ReBARC provides effective in-network retransmission and efficiently satisfies various latency requirements especially for high-quality, low-latency streaming applications. The main contributions of this study are summarized as follows:

- We characterize ICN hop-by-hop retransmission schemes and formulate a problem with the twofold objective of satisfying latency requirements and minimizing unnecessary bandwidth use for loss recovery.

- ReBARC achieves this objective, and it does not require additional traffic for exchanging control messages between nodes and enables each node to perform self-regulating transmissions of recovery interests considering the latency requirement and network conditions.
- We implement ReBARC and experimentally validate its performance.

The remainder of this paper is organized as follows. Section II introduces the related work. Section III presents the background of this study as well as the network and system models. We characterize the performance of in-network retransmission and formulate a problem in Section IV. We describe the design of our scheme in Section V and the performance evaluation of ReBARC in Section VI. In Section VII, we discuss some potentials of ReBARC. Finally, Section VIII concludes our work.

II. Related Work

Abu et al. studied consumer-based retransmission schemes using in-network caches in ICN and analyzed their recovery performance [15]. They then considered the sending of recovery interests at intermediate nodes, where in-network retransmissions increase network traffic load due to unnecessary retransmissions. However, they did not discuss how to develop in-network retransmission mechanisms that satisfy the acceptable latency specified by the involved application. Another previous scheme [17] also did not consider how to achieve the acceptable latency. These proposed schemes primarily focus on improving throughput performance and may not successfully recover lost data within the acceptable latency.

To suppress the adverse impact of interest losses on latency, Carofiglio et al. proposed the wireless loss detection and recovery mechanism [10], which promptly identifies and recovers channel losses occurring at wireless access links. The main idea is to implement additional sequencing on the interests to detect data loss at these links. An upstream node that detects a loss uses explicit feedback to trigger interest retransmission at the downstream node. However, although the authors addressed the channel losses of interests at wireless access links, they did not discuss how to deal with data losses occurring in a network.

Chen et al. designed the in-network proactive loss recovery (PLR) scheme, which offloads loss detection and recovery to intermediate nodes [19]. An intermediate node detects congestion-induced data loss in the uplink by monitoring its queue and then stores detected lost data for later retransmission. The lost data are retransmitted by the node without receiving a recovery interest. The node also uses explicit loss notification to inform the consumer about the data loss. PLR reduces the number of interest retransmissions and decreases the completion time compared with a consumer-based retransmission scheme. However, when data loss occurs during retransmission, PLR requires the consumer to issue a recovery interest, thus prolonging recovery delays.

In our previous work, we proposed an in-network recovery mechanism called L4C2 [16], which exploits the ICN feature of flow-aware hop-by-hop forwarding for timely data delivery while considering the latency requirement of the involved application. A node aims to quickly recover lost data from the caches at the upstream nodes by issuing a recovery interest according to the latency requirement. However, L4C2 does not provide any in-network retransmission control mechanism to suppress unnecessary retransmissions.

III. In-Network Retransmission

A. ICN Basics and Background

ICN is proposed to improve content delivery efficiency. In ICN, content is retrieved by name. If content cannot be packed into a single data packet, it is divided into multiple chunks uniquely identified by the content name and chunk number (e.g., /Room1/Robot1-Camera/chunk=3) [20]. A consumer sends interest packets specifying the content name (and chunk number, if necessary) to retrieve content, and a publisher delivers them in data packets. An intermediate node (commonly a router) along the forwarding path holds three primary data structures: (1) a forwarding information base, a lookup table that maps name prefixes and outgoing interfaces (called *faces*) to forward received interests; (2) a pending interest table (PIT), which contains the name specified by the received interests and outgoing faces (interest arrival faces) to forward received data; and (3) a content store (CS) for caching data. If an intermediate node is already storing the data specified by the received interest in the CS, then the data are immediately sent to the downstream node.

According to ICN's semantics [9], consumers send interests per chunk to retrieve data. Such an interest is called *regular interest (RGI)*. To increase throughput, consumers simultaneously send multiple RGIs and control the number of outstanding interests. However, especially for high-bandwidth streaming content, numerous chunks are transmitted in bursts; network congestion may occur and compromise network performance.

ICN's consumer-driven hop-by-hop transport allows consumers to flexibly determine transport strategies according to application characteristics. For example, *symbolic interests (SMIs)* [21] were proposed to smoothly fetch real-time streaming data from a publisher. Unlike RGIs, SMIs enable the wildcard specification of chunk numbers and provide bulk data transfer. Because SMIs can suppress interest traffic, they can minimize the risk of network congestion and packet loss in real-time streaming.

However, because an SMI does not specify a chunk number, it cannot recover lost chunks. An RGI can be used with an SMI as the recovery interest to request specific chunk retrieval during data transmission using SMIs. To deal with data recovery in a network, a consumer or an intermediate node should quickly detect data loss and send recovery interests (RGIs) to ask upstream nodes to retransmit lost chunks.

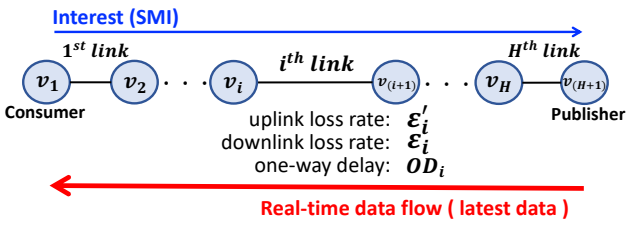


Fig. 1. Network model where publisher sends real-time content flow to consumer via H links.

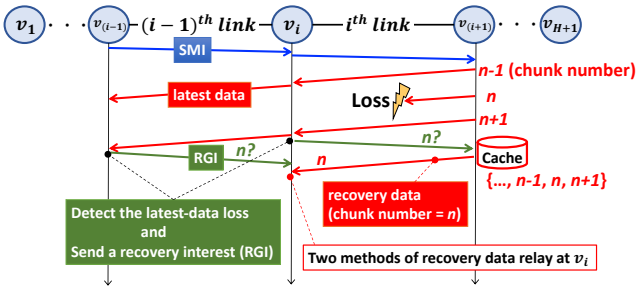


Fig. 2. Overview of latest-data loss detection and recovery. This figure omits the process of relaying the received recovery data at v_i.

B. Network and System Models for Real-Time Streaming Data Acquisition and Recovery

Network Model: To design real-time streaming data acquisition and recovery, we first consider a chain topology with H hops, including one consumer and one publisher, as illustrated in Fig. 1. Node $v_{i,1\leq i\leq H}$ is connected to its upstream node $v_{(i+1)}$ via the i^{th} link. The network primarily comprises the publisher $v_{(H+1)}$, which generates a real-time content flow; the intermediate nodes $v_{i,2\leq i\leq H}$; and the consumer v_1, which requests the real-time data (latest data) by issuing SMIs. The interest and data packets traversing the i^{th} link are subject to packet loss rates of ϵ_i' and ϵ_i, respectively. The one-way propagation delay from v_i to $v_{(i+1)}$ or from $v_{(i+1)}$ to v_i is OD_i, and the RTT of the i^{th} link (denoted by RTT_i) is $2{\cdot}OD_i$. For simplicity, the queuing delay for packet forwarding at each node is assumed constant.

System Model: The publisher generates real-time data packets of the flow and assigns a unique name with a chunk number to each piece of data. The consumer requests the latest data by sending an SMI per 100 ms. The PIT entry created by the SMI at each node remains for a few seconds. Therefore, even if SMI loss occurs at a link, each node can promptly and continuously receive and relay the latest data as long as data loss does not occur at the links.

As shown in Fig. 2, if the latest data are lost at the i^{th} link, then the downstream node v_i must detect this loss and recover the lost data by sending recovery interests (RGIs) to obtain the lost data from the CS at $v_{(i+1)}$. Thus, nodes $v_{i=\{2,\ldots,H+1\}}$ can cache the received latest data within the acceptable end-to-end latency at most (e.g., 200 ms) to respond to the received recovery interests. Each node employs the gap in the chunk number of the received latest data to detect and confirm the loss of latest data. After the loss is detected, the downstream nodes $v_{j=\{1,\ldots,i\}}$ immediately send recovery interests to the upstream nodes; these recovery interests specify the chunk number of the lost data. Because v_j sets the hoplimit of the recovery interest to 1, only $v_{(j+1)}$ receives it. Recovery data are relayed through the *stop-and-wait* and *fast-relay* methods, which are discussed in Section IV.

When the recovery interest issued by v_i or the recovery data sent by $v_{(i+1)}$ are lost at the i^{th} link, the node v_i must retransmit the recovery interest to fetch the recovery data from the cache at $v_{(i+1)}$. When sending the first recovery interest after detecting the latest-data loss, v_i sets a time-out value of recovery interest retransmission. It retransmits the recovery interest if the RTO event occurs. If loss recovery continuously fails, then v_i repeats recovery interest retransmission several times. The RTO value at v_i is as follows, which is the representation commonly employed in TCP [22]:

$$RTO_{v_i} = RTT_i + \alpha \qquad (\alpha = 4 \cdot var), \tag{1}$$

where RTT_i and var are the smoothed RTT and RTT variation, respectively. To measure RTT_i, v_i sends a measurement packet such as ICN-ping [23] to $v_{(i+1)}$ periodically. Node $v_{(i+1)}$ responds by sending the corresponding response data, and v_i retains RTT_i by measuring the difference between the interest sending time and the response reception time. The link loss rates, ϵ_i' and ϵ_i, are also measured using the measurement packet and corresponding response packets. A sequence number is assigned to the measurement packet by v_i and assigned to the response data by $v_{(i+1)}$. v_i and $v_{(i+1)}$ determine the sampled packet loss rate using the gap in the sequence number of the received packets. The packet loss rates are calculated using an exponential weighted moving average with a smoothing factor $\alpha = 0.8$ to adapt to the loss condition changes rapidly.

IV. Recovery Delay

In this section, we characterize the recovery delay performance of in-network retransmission and formulate the problem.

A. Definitions

We consider two methods for relaying recovery data: the *stop-and-wait* and *fast-relay* methods. Unlike the latter, the former does not allow v_i to immediately relay recovery data. To relay the recovery data to $v_{(i-1)}$, v_i should receive a recovery interest from $v_{(i-1)}$ after obtaining and caching the recovery data from $v_{(i+1)}$.

Recovery Delay: Consider latest data that are lost at the i^{th} link first (i.e., delivered up to $v_{(i+1)}$ without loss), as shown in Fig. 3. Assuming that v_i recovers and obtains the recovery data, the recovery delay at the i^{th} link is defined as follows:

$$R_i = t_{v_i}^{recv} - t_{v_{(i+1)}}^{recv} - OD_i, \tag{2}$$

where $t_{v_i}^{recv}$ is the time v_i receives the recovery data and $t_{v_{(i+1)}}^{recv}$ is the time $v_{(i+1)}$ receives and caches the data. Similarly,

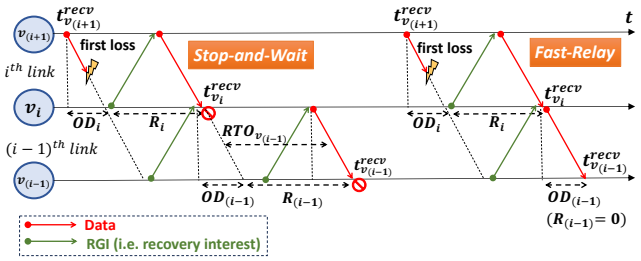


Fig. 3. Recovery delay in the stop-and-wait and fast-relay methods. In case of the fast-relay method, because the recovery data relayed to $v_{(i-1)}$ are not lost, the recovery delay $R_{(i-1)}$ is zero according to Eq. (2).

$R_{j,1\leq j\leq i-1}$ is defined as Eq. (2). Thus, the total recovery delay for the data to arrive at the consumer v_1 is as follows:

$$R^{tot}_{(1,i)} = \sum_{l=1}^{i} R_l. \tag{3}$$

Acceptable Recovery Delay: The one-way delay from $v_{(i+1)}$ to v_1 is $OD_{(1,i)} = \sum_{l=1}^{i} OD_l$. Let T_{app} be the application-specified acceptable latency. Assuming that $T_{app} > OD_{(1,i)}$, the *acceptable recovery delay* T is defined as follows:

$$T = T_{app} - OD_{(1,i)}. \tag{4}$$

Successful Loss Recovery: If $R^{tot}_{(1,i)} \leq T$, then loss recovery is successful.

B. Methods

Stop-and-Wait Method: Assume that the latest data are lost for the first time at the i^{th} link, and consider a round where v_i sends a recovery interest to $v_{(i+1)}$. Assuming the data recovery succeeds in the x^{th} round, the recovery delay (Eq. (2)) is approximated as follows:

$$R_i(x) = x \cdot RTO_{v_i} \quad (x \geq 1). \tag{5}$$

The probability that v_i obtains the recovery data in the x^{th} round is as follows:

$$P^{suc}_i(x) = \{1-(1-\epsilon'_i)(1-\epsilon_i)\}^{x-1}(1-\epsilon'_i)(1-\epsilon_i). \tag{6}$$

Consider the recovery delay at the $(i-1)^{th}$ link, which depends on the time at which v_i receives the recovery interest from $v_{(i-1)}$. The left part of Fig. 3 shows v_i receiving the recovery interest just before caching the recovery data. We consider this case because of (1) the delayed timing for v_i to relay the recovery data and (2) the importance of considering such an extreme scenario to satisfy applications' latency requirements. As in Eq. (5), the recovery delay of the downstream node v_j is approximated as $R_j(x) = x \cdot RTO_{v_j} (1 \leq j \leq i-1)$. Therefore, the expected total recovery delay is as follows:

$$E[R^{tot}_{(1,i)}] = \sum_{l=1}^{i}\sum_{x=1}^{\infty} P^{suc}_l(x) \cdot R_l(x). \tag{7}$$

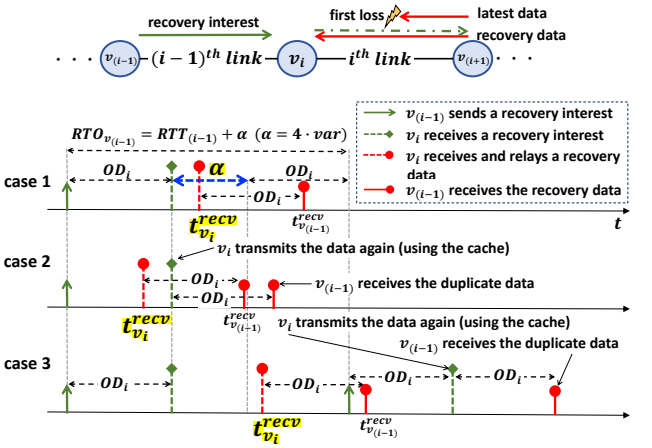


Fig. 4. Fast relay without duplicate transmission (case 1) and with duplicate data transmission and reception (cases 2 and 3).

Let x^{max}_l denote the maximum x of the l^{th} link $(1 \leq l \leq i)$ or the minimum number of rounds for v_l to obtain the recovery data from $v_{(l+1)}$ with a probability of over 99%.

$$x^{max}_l = min\{x | P^{suc}_l(x) \geq 0.99\} \quad (1 \leq l \leq i). \tag{8}$$

Given x^{max}_l, the maximum total recovery delay is approximated as follows:

$$R^{tot\_max}_{(1,i)} = \sum_{l=1}^{i} R_l(x^{max}_l). \tag{9}$$

Fast-Relay Method: Unlike the stop-and-wait method, the fast-relay method allows nodes $v_k (2 \leq k \leq i)$ to relay the received recovery data immediately to the downstream node $v_{(k-1)}$. Therefore, considering the probability that the relayed recovery data are lost at the link (denoted by $\epsilon_{(k-1)}$), the expected total recovery delay is as follows:

$$E[R^{tot}_{(1,i)}] = \sum_{x=1}^{\infty} P^{suc}_i(x) \cdot R_i(x) + \sum_{l=1}^{i-1}\sum_{x=1}^{\infty} \epsilon_l \cdot P^{suc}_l(x) \cdot R_l(x). \tag{10}$$

The maximum total recovery delay is approximated using Eq. (9).

C. Problem Formulation

In-network retransmission primarily aims to satisfy the acceptable recovery delay T (or the application-specified acceptable latency T_{app}). In the stop-and-wait method, a node does not relay the received recovery data until it receives a recovery interest from the downstream node. Therefore, it is not likely to cause duplicate data reception at the downstream node but likely to prolong recovery delay. The fast-relay method can reduce recovery delays relative to the stop-and-wait method but may unnecessarily duplicate data reception, increasing the network traffic load. Such an increase in the network traffic load will cause network congestion, so an effective, efficient mechanism should be developed to satisfy T while suppressing duplicate data receptions by bridging the gap between the stop-and-wait and fast-relay methods.

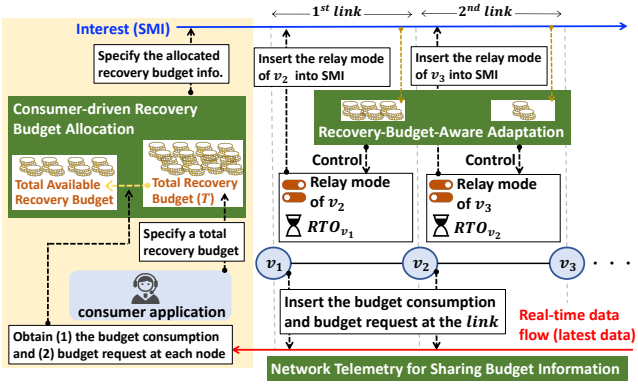


Fig. 5. Functions enabling ReBARC.

Fig. 4 illustrates three fast-relay result cases: one without duplicate data reception (case 1) and two with duplicate data reception (cases 2 and 3). We focus on $t_{v_i}^{recv}$ (the time at which v_i receives and relays the recovery data obtained from $v_{(i+1)}$) and $\alpha(=4 \cdot var)$ (which determines $RTO_{v_{(i-1)}}$). As shown in case 1, v_i should receive the recovery data within α after receiving a recovery interest (i.e., within the blue dotted line) to prevent v_i from transmitting duplicate data after receiving a recovery interest from $v_{(i-1)}$. The larger the value of α ($RTO_{v(i-1)}$), the higher the likelihood of suppressing duplicate data reception.

Because the latest data are lost at the i^{th} link first, it is not likely to cause duplicate data reception at v_i. Assuming a uniform timing of recovery data reception at the nodes, the probability of duplicate data reception at $v_j, (1 \leq j \leq i-1)$ is represented as follows:

$$P_j^{dup} = (1 - \frac{\alpha}{RTT_j + \alpha}) \cdot (1 - \epsilon_j) \cdot (1 - \epsilon_j')(1 - \epsilon_j). \quad (11)$$

The bandwidth cost of recovery interests is minimal because the size of the recovery interest is small relative to the data packet [24]. We thus formulate the following problem:

$$\begin{aligned} \text{Minimize} \quad & \sum_{j=1}^{i-1} N_{dup}^{tot}(j) \\ \text{subject to} \quad & R_{(1,i)}^{tot} \leq T, \end{aligned} \quad (12)$$

where $N_{dup}^{tot}(j)$ is the number of duplicate data receptions at v_j.

V. Protocol Design

This section presents the proposed scheme for ReBARC. The ReBARC mechanism follows the ICN protocol principle (consumer-driven communication and two-way exchange using interest and data) and fully exploits the features of hop-by-hop communication. Fig. 5 illustrates the functions enabling the recovery-budget-aware in-network retransmission.

A. ReBARC Features

The fundamental concept behind ReBARC is to execute retransmission-based loss recovery per link for locally recovering data loss. Keeping the total recovery delay within an acceptable latency is necessary. Thus, ReBARC controls the recovery delay per link by exploiting hop-by-hop communication in ICN. To enable self-regulating in-network retransmission control, we propose and introduce a metric called *recovery budget*, which corresponds to the maximum allowable recovery delay. The consumer allocates the total available recovery budget to each link (Section V-B).

According to the allocated budget, the recovery-budget-aware adaptation mechanism switches between the stop-and-wait and fast-relay methods (Section V-C). Then, the RTO value of the downstream node at each link is adjusted. For suppressing duplicate data receptions, the relay mode is set to the stop-and-wait method if the allocated budget is sufficient. Otherwise, the RTO value in the fast-relay method is increased by as much as the allocated budget allows.

Why Simple Loss Detection of Latest Data?: If each downstream node can determine where and what data were lost, the scope of recovery control can be limited to the link of data loss. In this case, only the downstream node at the link issues recovery interests using an adequate RTO value so that duplicate data reception does not occur. However, considering the existence of lossy links, quickly and reliably notifying all downstream nodes of data loss locations is difficult. Using additional sequence numbers to indicate lost data (as in [10]) increases the computational cost on each node.

ReBARC simply uses the gap in chunk number of the received latest data to confirm the loss. Each downstream node does not determine where the latest-data loss occurred, issuing the same RGIs (as in v_i and $v_{(i-1)}$ of Fig. 2); therefore, duplicate data receptions can occur. However, owing to the recovery-budget-aware adaptation, ReBARC suppresses duplicate data receptions at the nodes.

Why No Use of ACK?: By adopting an ACK mechanism for data reception by the downstream node, the upstream node can trigger data retransmission such that the downstream node avoids duplicate data receptions. However, a large amount of ACK or negative ACK messages increase the bandwidth consumption and may lead to network congestion. ReBARC, therefore, follows the ICN principle without using ACK.

B. Consumer-Driven Recovery Budget Allocation

All upstream nodes adopt the stop-and-wait method to avoid duplicate data reception. However, owing to the long recovery delays caused by not immediately relaying recovery data, loss recovery at a consumer can fail. In contrast, the fast-relay method enables faster loss recovery but results in increasing duplicate data reception. To manipulate the relay methods for effective and efficient loss recovery, in-network retransmission needs a criteria to determine which to use; therefore, the consumer allocates a recovery budget to each link, enabling each downstream node to perform self-regulating transmissions of recovery interests and determine the relay method of the upstream node. To notify each downstream node of the allocated budget, the consumer sends an SMI including the allocated budget information.

Algorithm 1 Recovery Budget Allocation at Consumer

1: **Require:**
2: T: total recovery budget (acceptable recovery delay).
3: H: total hop number from consumer to publisher.
4: **Output:**
5: $budget\_alloc(\mathcal{L})$: budgets allocated to each link,
6: $\mathcal{L} = \{l_1, ..., l_H\}$.
7: **At Data_Reception**($name, budget\_info$)
8: **if** Data has $budget\_info$ **then**
9: /* Update budget info. for links $\mathcal{L}$ */
10: Update_$budget_use(l \in \mathcal{L})$
11: Update_$budget_req(l \in \mathcal{L})$
12: /* Compute total budget consumption */
13: $total\_budget\_use \leftarrow$ Sum_$budget_use(l \in \mathcal{L})$
14: /* Compute total available budget */
15: $budget\_avail \leftarrow T - total\_budget\_use$
16: **if** $budget\_avail > 0$ **then**
17: **for** $l \in \mathcal{L}$ /* in ascending order */ **do**
18: /* Budget allocation for l */
19: **if** $budget\_avail > budget\_req(l)$ **then**
20: $budget\_alloc(l) \leftarrow budget\_req(l)$
21: $budget\_avail \leftarrow$
22: $budget\_avail - budget\_req(l)$
23: **else if** $budget\_avail == 0$ **then**
24: $budget\_alloc(l) \leftarrow 0$
25: **else**
26: $budget\_alloc(l) \leftarrow budget\_avail$
27: $budget\_avail \leftarrow 0$
28: **else**
29: $budget\_alloc(\mathcal{L}) \leftarrow 0$

Algorithm 2 Recovery Budget based Adaptation at v_l

1: **Require:**
2: RTT_l: RTT at l^{th} link.
3: $RTO^{min}_{v_l}$: minimum value of RTO_{v_l}.
4: ϵ'_l: loss rate of l^{th} uplink.
5: ϵ_l: loss rate of l^{th} downlink.
6: **Updated Output:**
7: RTO_{v_l}: The RTO value of v_l.
8: $m_{v_{(l+1)}} \in \mathcal{M}$: relay method at $v_{(l+1)}$,
9: $\mathcal{M}$ ={Stop-and-Wait, Fast-Relay}.
10: **At Interest_Reception**($name, budget\_info$)
11: **if** SMI has allocated budget info. **then**
12: /* Get budget allocated for l^{th} link */
13: $budget\_alloc(l) \leftarrow$
14: Get_Budget_Alloc($name, l, budget\_info$)
15: /* Determine the relay method of $v_{(l+1)}$ */
16: $budget\_req(l) \leftarrow RTO^{min}_{v_l} \cdot x^{max}_l$
17: **if** $budget\_alloc(l) \geq budget\_req(l)$ **then**
18: $m_{v_{(l+1)}} \leftarrow$ Stop-and-Wait
19: $RTO_{v_l} \leftarrow RTO^{min}_{v_l}$
20: **else**
21: $m_{v_{(l+1)}} \leftarrow$ Fast-Relay
22: /* Increase RTO_{v_l} if possible*/
23: $RTO_{v_l} \leftarrow max\{RTO_{v_l} \mid budget\_use(l)$
24: $\leq budget\_alloc(l)\}$
25: **if** $RTO_{v_l} < RTO^{min}_{v_l}$ **then**
26: $RTO_{v_l} \leftarrow RTO^{min}_{v_l}$
27: /* Notify $v_{(l+1)}$ of $m_{v_{(l+1)}}$ */
28: Send_Interest($m_{v_{(l+1)}}$)

Collection of Budget Conditions of Links: To recognize the total available budget to allocate, the consumer must obtain the *budget consumption* (estimated recovery delay) of each link as well as the total recovery budget specified by the application. Each downstream node thus estimates the budget consumption based on its uplink conditions and the relay method used. In the stop-and-wait method, the maximum budget consumption is estimated considering unsuccessful loss recovery owing to the long recovery delays. The budget consumption at link l is as follows using a number of recovery interest retransmissions that achieves loss recovery of over 99% at the link (as represented in Eq. (8)):

$$budget\_use(l) = \begin{cases} \sum_{x=1}^{x^{max}_l} \epsilon_l \cdot P^{suc}_l(x) \cdot R_l(x) & \text{(Fast-Relay)}, \\ RTO^{min}_{v_l} \cdot x^{max}_l & \text{(Stop-and-Wait)}, \end{cases} \tag{13}$$

where $RTO^{min}_{v_l}$ is set using Eq.(1). The RTO setting using $RTO^{min}_{v_l}$ can be considered sufficient to avoid duplicate data reception because the stop-and wait method relays recovery data only after receiving a recovery interest from the downstream node.

Each downstream node also estimates the *budget request*, which represents a budget required to execute the stop-and-wait method. The consumer can determine how much of a maximum budget to allocate to the link by referring to the budget request at each link. As the budget request at link l, the estimated maximum budget consumption in the stop-and-wait method is used as follows:

$$budget\_req(l) = RTO^{min}_{v_l} \cdot x^{max}_l. \tag{14}$$

Only when the allocated budget in the link is greater than or equal to the budget request, the stop-and-wait method is executed.

Each downstream node sends latest data including the budget consumption and request at the link to notify the consumer. The budget information sharing in a network telemetry manner does not need to issue additional control messages, which promotes efficient bandwidth usage.

Budget Allocation Policy: Algorithm 1 describes the recovery budget allocation at the consumer. When receiving data with budget information, the consumer reviews and registers the budget consumptions and requests (lines 10 and 11). Furthermore, the total budget consumption and total available budget are computed (lines 13 and 15).

As the proposed method adopts a simple loss detection mechanism for latest data, the downstream nodes detect latest-data losses occurring at links other than their uplinks (as in $v_{(i-1)}$ of Fig 2). Therefore, the further downstream a node is,

the more likely it is to detect latest-data losses. The consumer thus allocates the total available budget preferentially from nodes closest to the consumer (line 17). If the available budget is greater than or equal to the budget request of link l, then the consumer allocates it to link l (line 20). Otherwise, the remaining budget is allocated to link l for an opportunity to increase RTO_{v_l} (line 26).

C. Recovery-Budget-Aware Adaptation

Required Parameters: Algorithm 2 describes the recovery-budget-aware adaptation at v_l. For recovery-budget-aware adaptation, v_l should retain and update three parameters: RTT_l, ϵ_l and ϵ'_l. The parameters are determined using measurement packets and corresponding response data packets (Section III-B). To notify v_l of ϵ'_l, $v_{(l+1)}$ inserts the loss rate value of ϵ'_l into the latest data after updating ϵ'_l.

Adjustment of Relay Method and RTO value: If a received SMI has budget allocation information for the l^{th} link (line 11), then v_l determines the relay mode of $v_{(l+1)}$ and RTO_{v_l} based on the allocated budget. If the allocated budget is greater than or equal to the budget request, the relay method $m_{v_{(l+1)}}$ is set to the stop-and-wait method (line 18). Then, RTO_{v_l} is set to $RTO_{v_l}^{min}$ (line 19). To require the upstream node $v_{(l+1)}$ to execute the stop-and-wait method, v_l sends the SMI including the information (line 28). If the allocated budget is insufficient for $v_{(l+1)}$ to execute the stop-and-wait method, the relay method $m_{v_{(l+1)}}$ is set to the fast-relay method. Then, RTO_{v_l} is increased based on the allocated budget if possible (lines 23 and 24). Duplicate data receptions in the fast-relay method can be suppressed as much as possible by increasing the RTO value such that the budget consumption falls within the allocated budget.

VI. Experimental Evaluation

A. Implementation

We implemented the recovery-budget-aware in-network retransmission mechanism using the open-source software *Cefore* [25], which is compatible with the CCNx protocol 1.0, as specified by the Internet Research Task Force [9]. Cefore provides actual running codes for ICN-based communication equipped with SMIs (and RGIs) and in-network caches. For recovery-budget-aware adaptation, we modified *cefnetd*, a basic forwarding daemon, to handle interest and data encoded in IPv4/v6 packets. Our implementation enabled nodes to insert budget information into SMIs and data using the hop-by-hop Type-Length-Value (TLV) header [9]. Cefore provides two sample programs named *cefputstream* and *cefgetstream*. *cefputstream* is a publisher application that sends real-time stream data sequentially, while *cefgetstream* is a consumer application for obtaining data with specified names. For recovery budget allocation at the consumer, *cefgetstream* was enhanced to support the ReBARC algorithm.

B. Experimental Setting

We experimentally investigated in-network retransmission in an emulated testbed using Mininet [26] and the modified Cefore. The main questions were about how the proposed method could enhance the success of loss recovery compared with existing methods, satisfy latency requirements efficiently, and improve video quality.

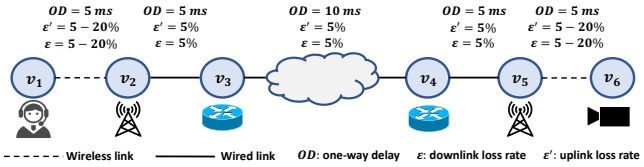


Fig. 6. Network topology used in the experiment.

Network Topology and Condition Setting: We used a typical chain topology (Fig. 6). The network comprised of six nodes: (1) one consumer, (2) one publisher (which sends real-time data at 20 Mbps), (3) two wireless access points (APs), and (4) two edge nodes.

The access links were assumed to be wireless links where the interest and data packet losses followed the official 3GPP scenario (Table 6 in [27]), which considers a pedestrian at 3 km/h. Considering severe loss conditions, mean loss rates of 5%, 10%, and 20% were used. The one-way propagation delay of each access link was set to 5 ms. Given that most of loss rates are 1%–10% [28], we set the loss rate of the wired core link between the two edge nodes to 5%. The one-way propagation delay of the core link was set to 10 ms. Similarly, regarding the two wired links between (1) the AP at the consumer and the edge node and (2) the AP at the publisher and the edge node, the loss rates were set to 5%. The one-way propagation delay was set to 5 ms.

Performance Metrics: The measured performance metrics were (1) the ratio of successful loss recovery at the consumer, (2) the ratio of duplicate data reception (the total number of duplicate data received by a node divided by the total number of latest data sent by the publisher), and (3) the video playback quality (specifically the structural similarity index measure [SSIM]) [29]. We used the ratio of duplicate data reception averaged by the number of links. The SSIM provided a similarity score of 0–1 to assess the video frame displayed at the consumer end. The lowest SSIM score of 0 implied that the video frame was dropped, whereas the highest score of 1 indicated that the displayed video frame was identical to the original one.

Schemes for Comparison: The performance of ReBARC was compared with those of two ICN-based schemes: consumer-based retransmission scheme [15] with a TCP-like RTO (called baseline scheme) and an in-network PLR scheme [19] where intermediate nodes spontaneously retransmit lost latest data (called existing scheme). For the existing scheme to achieve a high successful loss recovery ratio, we assumed that the nodes retransmitted lost latest data immediately and provided explicit loss notification. Both schemes do not perform timely loss detection at the consumer end for latest data. Therefore, for a fair comparison, the two schemes were set to detect latest-data loss in the same manner as our proposed scheme.

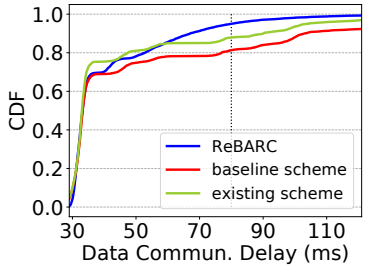
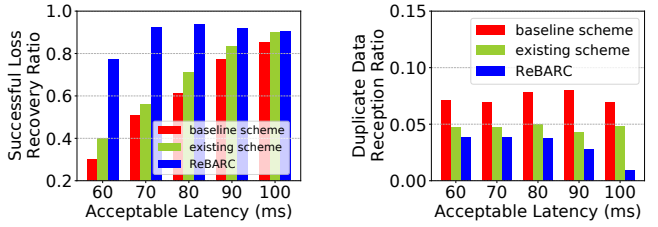


Fig. 7. Performance of end-to-end data communication delay. In this experiment, the acceptable latency was set to 80 ms.



(a) **Successful loss recovery ratio** (b) **Duplicate data reception ratio**

Fig. 8. Performance results according to acceptable latency requirements.

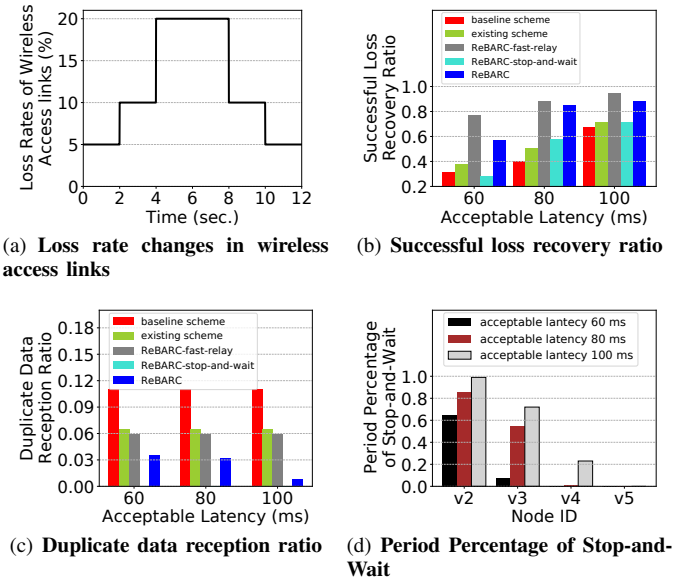
C. Loss Recovery Performance: Preliminary Test

We first investigated the communication delay performance of ReBARC with a relatively severe latency requirement, where the acceptable latency was set to 80 ms. In this experiment, we used the topology shown in Fig. 6, with the loss rates at the wireless access links being statically set to 10%.

Fig. 7 presents the ReBARC performance result of end-to-end data communication delay, compared to those corresponding to the baseline and existing schemes. The baseline scheme failed to achieve a high ratio of successful loss recovery because of the lack of a fast in-network recovery mechanism. Similarly, the existing scheme could not achieve a higher successful loss recovery ratio, because the loss recovery partly relies on consumer-based retransmission, prolonging the recovery delays. Therefore, both the schemes could not satisfy the acceptable latency requirement well. In contrast, ReBARC kept almost the entire data transmission within the acceptable latency of 80 ms, achieving a high successful loss recovery ratio of more than 90%.

D. Impact of Differences in Latency Requirements

In this evaluation, the loss rates at the wireless access links were statically set to 5% to investigate acceptable latency effects. Figs. 8(a) and 8(b) present the performance assessment results with respect to the application latency requirement. The successful recovery ratios of both the baseline and existing schemes were lower than that of ReBARC, especially with a low acceptable latency. The results indicate that ReBARC works effectively for loss recovery, even under severe end-to-end loss. In case of the acceptable latency $\geq$ 90 ms, the successful recovery ratios of ReBARC slightly decreased,



(a) **Loss rate changes in wireless access links** (b) **Successful loss recovery ratio**

(c) **Duplicate data reception ratio** (d) **Period Percentage of Stop-and-Wait**

Fig. 9. Performance at varying loss rates of wireless access links.

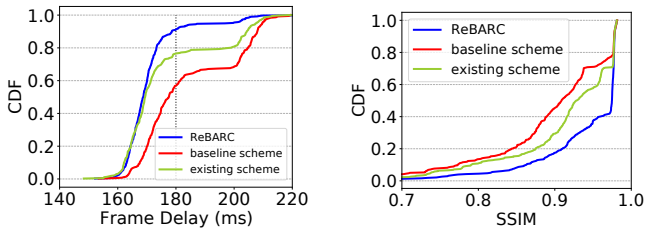
compared to the case where that was 80 ms. This was because the ReBARC adaptation was more likely to adopt the stop-and-wait method when having more recovery budget, which caused unsuccessful loss recovery owing to the long recovery delays.

Regarding the duplicate data reception ratio, both the baseline and existing schemes performed worse than ReBARC. This was because of the variation in the places of nodes holding the lost data and ineffective RTO settings at the consumer, as reported in [15]. ReBARC avoids this issue by adopting hop-by-hop adaptation based on the allocated recovery budget. Therefore, ReBARC efficiently suppressed duplicate data reception compared with the other schemes. These results indicate that effective, efficient in-network retransmission was performed by ReBARC.

E. Impact of Loss Conditions

We investigated the adaptability of ReBARC by varying the loss rates of the wireless access links (Fig. 9(a)) considering a situation where wireless links become severe and unstable. As shown in Figs. 9(b) and 9(c), ReBARC outperformed the baseline and existing schemes in terms of both the successful loss recovery ratio and duplicate data reception ratio.

We also evaluated the performance of two ReBARC variants: one where all nodes always adopt the fast-relay method (called ReBARC-fast-relay) and one where all nodes always adopt the stop-and-wait method (called ReBARC-stop-and-wait). As discussed in Section IV, ReBARC-fast-relay suppressed recovery delays, thus enhancing the successful loss recovery ratio, but increased the duplicate data reception ratio. ReBARC-stop-and-wait substantially suppressed the duplicate data reception ratio but achieved a lower successful loss recovery ratio. Fig. 9(d) presents the period percentage of the stop-and-wait method of each node adjusted by ReBARC,



(a) **Frame delay** (i.e., playback latency) (ms)

(b) **SSIM**

Fig. 10. Video playback quality. In this experiment, the acceptable latency was set to 180 ms.

which affected ReBARC performance. Nodes v_2 and v_3 denote the consumer-end AP and edge node, respectively, and v_4 and v_5 denote the publisher-end edge node and AP, respectively. Given an insufficient acceptable latency ($\leq$ 80 ms), the period percentages at v_2 and v_3 decreased due to the less available budget, resulting in more duplicate data receptions at the links. Instead, ReBARC achieved a considerably higher successful loss recovery ratio compared with ReBARC-stop-and-wait. ReBARC adaptation bridged the gap between the fast-relay and stop-and-wait methods, achieving well-balanced performance according to the acceptable latency and network conditions. Thus, these results showed that ReBARC could ensure stable teleoperation [32].

F. Video Quality

We evaluated ReBARC in terms of video playback quality. We used the video application GStreamer [30] at both the publisher and consumer ends. The publisher-end application encoded a prerecorded source video (4K resolution) using an x264 encoder at a bitrate of 20 Mbps and sent it over the real-time transport protocol [31] via *cefputstream*. The source video contained a sufficiently moving scene. For low-latency settings, we only used I-frames and P-frames (i.e., without B-frames, which would have required additional latency).

In this experimental environment, the total processing time of video encoding/decoding was $\simeq$ 60 ms at most. The acceptable latency was set to 180 ms considering the processing time. The network conditions in the Section VI-D evaluation were used. The video play time was approximately 10 s.

Figs. 10(a) and 10(b) show the overall frame delays and SSIM performance, respectively. ReBARC outperformed both the baseline and existing schemes owing to the in-network retransmission mechanism per link; its frame delays were lower than those of the baseline and existing schemes, resulting in higher SSIM scores for the frames. In ReBARC, approximately 90% of the frame delays were suppressed within the acceptable latency and the SSIM scores for most of the frames were maintained at above $\simeq$ 0.8. These results highlighted that even in such a severe loss condition, ReBARC can contribute to achieving stable real-time video transmission with higher video quality.

VII. Discussion

Application of FEC: If an allocated recovery budget is insufficient due to a long propagation delay at a link, then only the retransmissions at that link cannot achieve successful loss recovery. In this case, FEC can be useful [16]. Our proposed in-network retransmission mechanism can effectively incorporate FEC. Using recovery interests, a downstream node requests redundant data (replica or coded data) from the upstream node if the allocated budget is less than the estimated budget consumption. Such redundant data traffic can be effectively eliminated, as redundant data transmissions are applied only at links where successful loss recovery cannot be achieved by in-network retransmissions alone.

Rate Control: If network congestion occurs at a bottleneck link, then the publisher should decrease the data transmission rate by lowering the video quality to avoid congestion collapse. Several effective rate control methods using explicit congestion notification in ICN have been proposed [33], [34]. ReBARC has a potential to sophisticate such rate control mechanisms, as in-network retransmission can effectively address data losses caused by network congestion for successful loss recovery. In our future work, we will combine in-network retransmission with such a rate control mechanism.

VIII. Conclusions

Sophisticated in-network retransmission for fast loss recovery should be designed to promote various ICN-enabled use cases. We propose ReBARC as an effective fully distributed loss recovery mechanism for high-quality, low-latency streaming applications. It leverages the architectural benefits of ICN, such as in-network caching and consumer-driven hop-by-hop communication. ReBARC enables nodes to recognize the maximum allowable delay and perform loss recovery considering the end-to-end latency requirement without issuing and exchanging additional control messages with each other. It also prevents duplicate data reception. We implement ReBARC using an open-source software with the CCNx 1.0 protocol and conduct experiments. Compared with existing methods, ReBARC helps maintain higher video quality by achieving higher successful loss recovery while suppressing duplicate data reception.

Acknowledgement

This work was partly supported by JST Moonshot R&D Goal 1, Grant Number JPMJMS2216.

References

[1] A. Baltaci, et al., "A survey of wireless networks for future aerial communications (FACOM)," *IEEE Communications Surveys and Tutorials*, vol. 23, no. 4, Aug. 2021, pp. 2833–2884.
[2] P. Nadrag, et al., "Remote control of a real robot taking into account transmission delays," *Proc. IFAC*, vol. 43, no. 10, 2010, pp. 59–64.
[3] J. Iyengar and M. Thomson, "QUIC: A UDP-based multiplexed and secure transport," IETF, RFC 9000, May 2021.
[4] H. She, et al., "Analytical evaluation of retransmission schemes in wireless sensor networks," *Proc IEEE VTC*, Jun. 2009.
[5] M. Irland and G. Pujolle, "Comparison of two packet-retransmission techniques (corresp.)," *IEEE Trans. Inf. Theory*, vol. 26, no. 1, Jan. 1980, pp. 92–97.

[6] S. Heimlicher, et al., "End-to-end vs.hop-by-hop transport under intermittent connectivity," *Proc. of ICST Autonomics,* 2007, Oct. 2007.

[7] H. Asaeda, et al., "A Survey of Information-Centric Networking: The Quest for Innovation," *IEICE Transaction on Communications*, vol. E107-B, No. 1, Jan. 2024, pp. 139–153.

[8] L. Zhang, et al., "Named data networking," *ACM Comput. Commun. Rev*., vol. 44, no. 3, Jul. 2014, pp. 66–73.

[9] M. Mosko, I. Solis, and C. Wood, "Content-centric networking (CCNx) messages in TLV format," IRTF, RFC 8609, Jul. 2019.

[10] G. Carofiglio, et al., "Leveraging icn in-network control for loss detection and recovery in wireless mobile networks," *Proc. ACM ICN*, Sep. 2016, pp. 50–59.

[11] M. Nikzad, et al., "An accurate retransmission timeout estimator for content-centric networking based on the Jacobson algorithm," *Digital Communications and Networks*, vol. 8, no. 6, Dec. 2022, pp. 1085–1093.

[12] K. Schneider, et al., "A practical congestion control scheme for named data networking," *Proc. ACM ICN*, Sep. 2016, pp. 21–30.

[13] K. Ueda, et al., "Revisiting loss detection in NDN: detecting spurious timeout using probe interest," *Proc. IEEE Globecom*, Dec. 2022.

[14] W. M. H. Azamuddin, et al., "The emerging of named data networking: architecture, application, and technology," *IEEE Access*, vol. 11, Feb. 2023, pp. 23620–23633.

[15] A. Abu, et al., "Interest packets retransmission in lossy CCN networks and its impact on network performance," *Proc. ACM ICN*, Sep. 2014, pp. 167–176.

[16] K. Matsuzono, H. Asaeda, and T. Turletti, "Low Latency Low Loss Streaming Using In-Network Coding and Caching," *Proc. IEEE INFOCOM*, May 2017, pp. 1–9.

[17] Z. Wang, et al., "R^2T: A rapid and reliable hop-by-hop transport mechanism for information-centric networking," *IEEE Access*, vol. 6, 2018, pp. 15311–15325.

[18] S. Oueslati, J. Roberts, and N. Sbihi, "Flow-aware traffic control for a content-centric network," Proc. *IEEE INFOCOM*, May 2012, pp. 2417–2425.

[19] Y. Chen, et al., "PLR: An In-Network Proactive Loss Recovery Scheme for Named Data Networking," *Proc. IEEE ICCCN*, Jul. 2023.

[20] M. Mosko and H. Asaeda, "CCNx Content Object Chunking," IRTF, Internet-Draft (work in progress), Oct. 2024.

[21] K. Matsuzono, D. Nguyen, and H. Asaeda, "Content Request Handling for Application-Oriented Transport Control," *IEEE Commn. Mag*., vol. 57, no. 6, Jun. 2019, pp. 14–19.

[22] M. Sargent, et al., "Computing TCP's retransmission timer," IETF, RFC 6298, Jun. 2011.

[23] S. Mastorakis, et al., "Information-centric networking (ICN) ping protocol specification," IRTF, RFC 9508, Mar. 2024.

[24] D. Han et al., "RPT: re-architecting loss protection for content-aware networks," *Proc. USENIX NSDI*, Apr. 2012.

[25] "Cefore," available at: https://github.com/cefore, accessed Mar. 5. 2025.

[26] "Mininet," available at: http://mininet.org/, accessed Mar. 5, 2025.

[27] ETSI, "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Multimedia Broadcast/Multicast Service (MBMS); Selection and characterization of application layer Forward Error Correction (FEC)," 3GPP TR 26.947 version 17.0.0 Release 17, Apr. 2022.

[28] M. Rudow, et al., "Tambour: Efficient loss recovery for videoconferencing via streaming codes," *Proc. 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Apr. 2023.

[29] Z. Wang, et al., "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, Apr. 2004, pp. 600–612.

[30] "GStreamer," available at: https://gstreamer.freedesktop.org/, accessed Mar. 10, 2024.

[31] H. Schulzrinne, et al., "RTP: A Transport Protocol for Real-Time Applications," IETF, RFC 3550, Jul. 2003.

[32] A. Baltaci, et al., "Analyzing real-time video delivery over cellular networks for remote piloting aerial vehicles," *Proc. ACM Internet Measurement Conference (IMC)*, Oct. 2022, pp. 98–112.

[33] D. Nguyen, J. Jin, and A. Tagami, "Cache-friendly streaming bitrate adaptation by congestion feedback in ICN," *Proc. ACM ICN*, Sep. 2016, pp. 71–76.

[34] K. Schneider, et al., "A practical congestion control scheme for named data networking," *Proc. ACM ICN*, Sep. 2016, pp. 21–30.