Regular Paper

A Lost Data Recovery Scheme for Sensor Data Stream Multicasting

Ei Khaing Win[1,a)] Tomoki Yoshihisa[1,b)] Yoshimasa Ishi[1,c)] Tomoya Kawakami[2,d)]
Yuuichi Teranishi[3,e)] Shinji Shimojo[1,f)]

Received: May 10, 2017, Accepted: November 7, 2017

Abstract: In this paper, we propose a lost data recovery scheme called the synchronized recovery stream merging (SRSM) for sensor data stream multicasting in the unstable networks of the IoT applications. We propose two types of SRSM. The first one is "Latency-aware synchronized recovery stream merging (SRSM-L)" for IoT applications that has restrictions of acceptable latency. The other is "Bandwidth-dependent synchronized recovery stream merging (SRSM-B)" for the cases in which the network bandwidth for the sender is limited. Our proposed schemes reduce the number of the streams managed by the sender by waiting for other recovery streams to synchronize the delivery timing and merging. From our simulation evaluations, we confirmed that our proposed schemes save network bandwidth on the sender in the random and burst failure situations. We confirmed that SRSM-L could reduce the network bandwidth of the sender about 52% in the random failure situation, keeping the acceptable latency. We also confirmed that SRSM-B could keep the specified number of streams constant and the latency overheads small in the burst failure situation.

Keywords: lost data recovery, synchronized recovery stream merging, sensor data stream multicasting, sensor network, latency-aware, bandwidth-dependent

1. Introduction

The internet of Things (IoT) [1], [2], [3] has gained popularity due to the proliferation of small devices and sensors such as camera or temperature sensor. And sensor data stream delivery technology takes an important role in the IoT applications. In the IoT applications, there are many cases where multiple receivers receive sensor data stream generated continuously by a sensor and utilize the sensor data stream for various objectives. For example, a video data stream obtained by a camera can be used for suspect tracking, congestion detection, situation logging, weather analysis, disaster prevention, and so on. In the edge computing environment [4], in which many computers run on the network edges to realize quick-response on IoT applications, a large number of edge computer devices must receive the sensor data stream. For efficient data stream delivery for multiple receivers, we assume to use multicasts. In this paper, we assume that the network has ability of multicast, such as the ad-hoc sensor networks [5], [6], IP multicasts [7], and application layer multicasts like pub/sub messaging [8]. The pub/sub messaging is already widely utilized by many IoT applications [9], [10]. Some existing works implement publish/subscribe messaging on distributed application layer multicast [11], [12].

In the IoT applications, we must assume that the applications are executed on the computer devices that are managed by end users, such as smartphones or Cloudlet [13] in edge computing environments. The applications run on the widely distributed devices on such environments. The devices are connected to the Internet via wired or wireless access network but we cannot assume they are as stable as the network of the datacenters. That is, we must assume that the data losses can occur because of the unstable network links, unexpected power on/off of the sender/receiver devices, electricity black-offs, and some other reasons.

For reliable data usage and data analysis, the lost sensor data by such failures need to be recovered after the network status or receiver device is restored. Typically, the sender waits for a request for recovery and retransmit the lost data to each failure encountered receiver. As the number of recovery request increases, the sender needs to prepare more data streams for receivers with different failure conditions. The requirement of each receiver may differ because the failures may occur asynchronously. As a result, recovery streams are needed to be prepared for each loss-encountered receiver. As we assume IoT Applications, the sender does not have plenty of CPU power nor network ability. That means the network of the sender is limited in bandwidth. Therefore, it is important to reduce the number of streams for recovery to ease the network bandwidth. However, such data loss recovery in multicast was not studied enough so far.

1 Osaka University, Ibaraki, Osaka 567–0047, Japan
2 Nara Institute of Science and technology, Ikoma, Nara 630–0101, Japan
3 National Institute of Information and Communications Technology, Koganei, Tokyo 184–8765, Japan
a) ei.khaing.win@ais.cmc.osaka-u.ac.jp
b) yoshihisa@cmc.osaka-u.ac.jp
c) ishi.yoshimasa@ais.cmc.osaka-u.ac.jp
d) kawakami@is.naist.jp
e) teranisi@cmc.osaka-u.ac.jp
f) shimojo@cmc.osaka-u.ac.jp

To reduce the number of streams and alleviate network bandwidth usage on the sender, existing stream merging schemes (Refs. [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25]) can be applied. Because these existing schemes assume mainly video-on-demand (VoD) applications, they need to keep the jitter of the data stream delivery as small as possible for smooth video playback. However, the jitter is not critical in the IoT applications which analyze the data in order of arrival. Besides, most of the existing schemes assume that there is a large size buffer on the receiver. We cannot expect such a large size buffer on the IoT applications that often deployed as embedded systems.

Hence, in this paper, we propose a novel lost sensor data recovery scheme for sensor data multicasting called synchronized recovery stream merging (SRSM), which is suitable for IoT applications. We propose two types of synchronized recovery stream merging algorithms for different IoT application requirements. The first one is "Latency-aware synchronized recovery stream merging (SRSM-L)" for the IoT applications in which the receivers can tolerate the sender's data delivery latency as long as the delivery latency is in the range of their acceptable latencies, and the other is "Bandwidth-dependent synchronized recovery stream merging (SRSM-B)" for the cases in which the network bandwidth for the sender is limited. Our contribution in this paper is an extensive work of our previous paper [26], which analyzes the basic behavior of SRSM. The previous paper does not include the designs of the algorithms that corresponds to the specific requirements (SRSM-L and SRSM-B) and their evaluations. From our evaluations in this paper, we confirm that our proposed sensor data recovery algorithms can reduce the number of the streams in the different failure scenarios.

The paper is organized as follows. Section 2 describes the assumed system model and recovery stream delivery while Section 3 recalls the related work. In Section 4, the synchronized recovery stream merging, the proposed schemes and the detailed algorithms of the proposed schemes are explained in detail. The paper shows evaluation by simulations in Section 5. Finally, we conclude the paper in Section 6.

2. Assumptions

In this section, we describe our assumed system model and recovery stream delivery.

2.1 Assumed System Model

Figure 1 shows our assumed system model. For sensor data stream delivery, the sender has three components: sensor data obtainer, sensor database and sensor data stream generator. Firstly, sensors sense data from the environment and then forward them to sensor data obtainer. After sensor data obtainer obtains the sensed data, it stores them in sensor database. The task of sensor data stream generator is the generation of new streams for lost sensor data recovery and dropping of some streams after successful merging. In Fig. 1, we assume that the sender delivers some multicast streams like sensor data streams 1 and 2, each for multiple receivers.

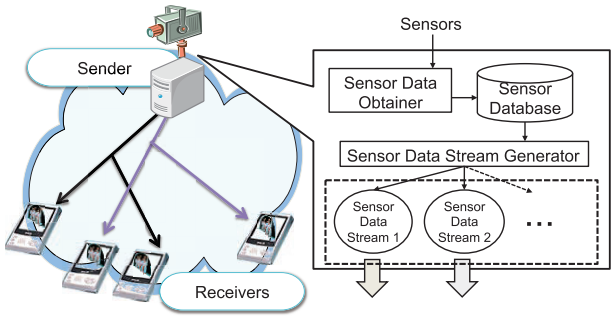

Fig. 1 Our assumed system model.

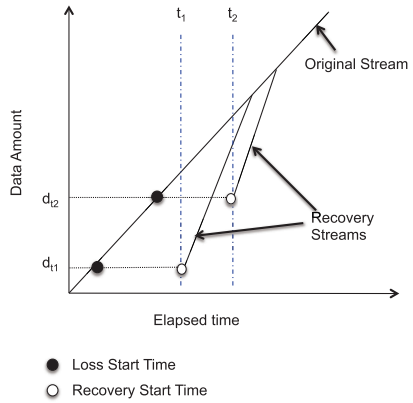

Fig. 2 Recovery stream delivery.

2.2 Recovery Stream Delivery

We define the data stream which delivers the sensor data immediately after they were obtained as the *original stream*. When a receiver encountered a failure and then recovered, which means a receiver encountered a data loss, a new data stream request is issued to the sender so that the loss-encountered receiver can obtain data which are delivered by the original stream during the failure.

We call the newly generated data streams for the lost sensor data recovery as the *recovery streams*. The recovery stream can be a larger bandwidth data stream, which includes more data per unit time than the original stream or a skipped data stream, which drops some data from the original stream. The sender delivers the recovery stream according to the receiver's request. For the recovery stream with skipped data, we define a *skip rate* that means how much data are skipped within a unit time. The amount of data to be skipped within a unit time is equal to (skip rate - 1) $\times$ (original data amount), where original data amount means the amount of data that the sender sends within a unit time in the original stream. The bandwidth is the data amount that the sender sends within a unit time. For example, if the skip rate is 2.0 and bandwidth is 1.0, the sender skips one data within a unit time.

Figure 2 illustrates the basic behavior of the recovery stream delivery. The x-axis represents the elapsed time and y-axis corresponds to the delivered data amount. In Fig. 2, a receiver recovers from failure at time (t_1) and the sender needs to generate a new stream by using the receiver's desired skip rate if there is no existing recovery stream. When another receiver with different failure condition requests for recovery at time (t_2), the sender has to prepare new recovery stream. Recovery stream with larger skip rate or larger bandwidth faster catches up with the original stream.

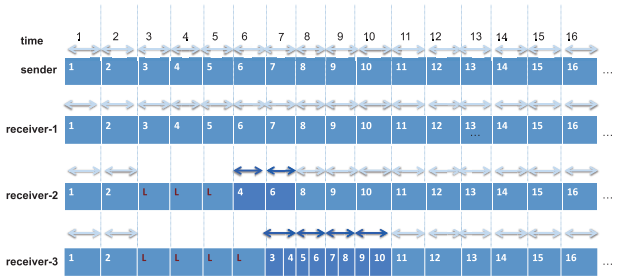

Fig. 3 An example of skipped data delivery.

In **Fig. 3**, we assume that the sender delivers the original data stream with bandwidth = 1.0 situation. Figure 3 shows the situation in which the receiver-2 sets skip rate as 2.0 and bandwidth as 1.0 while the receiver-3 sets skip rate as 1.0 and bandwidth as 2.0 for lost sensor data recovery. As the receiver-2 sets skip rate as 2.0, the sender skips one data before choosing one recovery data. The quantity the sender eliminates the data from the recovery stream is a parameter and depends on the importance level of the data specified by the requested receiver. As the eliminated data increases, the recovery stream can faster catch up with the original stream. Although the recovery stream can faster catch up with the original stream, the number of received data on the receiver decreases. When the recovery streams catch up with the original stream, the sender can deliver the same original stream to the receivers that have already received the lost sensor data. Therefore, the sender does not need to generate the recovery stream and the number of the streams is reduced. In Fig. 3, at time 6, the sender uses total bandwidth 2.0 for the original stream and the recovery stream. At time 7, the sender has to use total bandwidth 4.0 as the number of recovery stream increases. Therefore, the bandwidth of the sender is proportional to the number of recovery streams. If there are multiple requests for recovery with different failure situations, the sender may exhaust in bandwidth usage. Therefore, not only receiver but also sender should be considered in lost data recovery scheme.

The critical problem in such data delivery is how to alleviate the network bandwidth usage of the sender. When more loss-encountered receivers with different failure situations are involved in recovery, the sender's load such as the computational power, memory usage, communication traffic, etc. increases. In other words, the sender's load is proportional to the number of recovery streams.

3. Related Work

To reduce the senders' load such as bandwidth, and I/O overhead, merging schemes have been proposed for VoD senders. There are stream merging schemes called Batching [14], Piggybacking [15], Tapping [19], Patching [20], and Dynamic skyscraper [21].

To reduce the sender's bandwidth, stream delivery is delayed in the Batching. Time interval is specified so that the sender needs to deliver only one stream for all the receivers' same data requests that arrived during that time interval. However, Batching introduces delay in receivers. To make the streams merge faster, the streaming rates for recovery streams are dynamically adjusted in the Piggybacking. In the Tapping or the Patching, the receiver can greedily tap into any streams by using buffer capable of storing at least 10 to 15 minutes. In the Dynamic skyscraper, the sender's bandwidth is reduced by allowing the receivers to receive more than one stream at the same time. The model used in the Tapping and the Dynamic skyscraper is called the receive-two model where receivers can listen to two streams simultaneously. The receiver receives two streams: one for immediate viewing purpose and another for future use. The data for future playback are stored in buffers. The server drops the streams whose data are in the receiver's buffer. The sender informs the receiver processes on the receivers about the streams to listen to and the duration.

In Ref. [24], efficient dyadic stream merging algorithm has been proposed by allowing the receivers to receive up to two streams at any time. Client receive programs are prepared using recursive dyadic interval partitioning. The Dyadic Tree is constructed by assuming the original stream as a root (parent) and the earliest receiver arrived within the specified intervals as the children. Streams for children are merged with parent. Moreover, the streams for later arrival receivers and earliest arrival receiver within the same interval are merged again. In this way, the number of streams are reduced. However, the receivers need to have the buffer to simultaneously accept two streams at any time.

In Ref. [25], five stream merging algorithms for media-on-demand are compared. Using various distributions for receiver request pattern, empirical study results are presented for the dynamic Fibonacci, dyadic and earliest reachable merge target (ERMT). Simulation results are shown for various performance metrics. Most of them use the buffer on the receiver.

Because above existing schemes assume mainly VoD applications, they assume that the length of the data processing interval or the data delivery interval needs to be almost same, in other words, the jitter of the processing data interval or data delivery interval needs to be small so that a person cannot notice for the smooth video playback. By this jitter limitation, the number of the streams that the sender can reduce is limited, though the jitter is not critical in the IoT applications. To obtain the real-time situations of the real world, the arrival order of the data is important because the analysis process has its own context, which means the transitions of the status is decided based on the previous status. For example, in the person tracking by video applications, the movement of a person in the video is calculated by the position of the current video frame and the previous video frame. If the receiver has plenty of buffers to store received data, the order of the data can be adjusted according to the timestamp of the data even when the arrival order of the data is not preserved. In this study, we assume that the receiver does not have such large buffer to store data because IoT applications can run on small devices such as smartphones. Reducing the bandwidth is a more critical issue in the IoT applications.

4. Proposed Method

In this section, we present the synchronized recovery stream merging (SRSM) firstly. Then, two stream data recovery schemes called "Latency-aware synchronized recovery stream merging (SRSM-L)" and "Bandwidth-dependent synchronized recovery

stream merging (SRSM-B)" are explained in detail.

4.1 Synchronized Recovery Stream Merging (SRSM)

In our proposed SRSM scheme, we assume that the receiver specifies an acceptable latency. The acceptable latency is specified according to the IoT application demands. Example scenarios are:

- For navigation application, the status of congestion (by analyzing street camera video) older than 30 sec has no meaning. In this case, the receiver will specify the acceptable latency as 30 sec.
- For congestion logging application, the receivers may wait lost data for a half day. In this case, the receiver's acceptable latency is equal to half day.

The acceptable latency may depend on the the receivers. Different receivers may have different latency tolerance and different specification of data importance on the same data.

In our proposed schemes, stream merging occurs when the recovery stream catches up with the original stream and it is also performed among recovery streams. Recovery streams are merged when they reach the same data delivery point. The following notations are used as parameters of the receiver-i for the recovery.

r_i : the acceptable skip rate of the receiver-i.
b_i : the acceptable bandwidth of the receiver-i
d_i : the latest data timestamp that the receiver-i already received
l_i : the acceptable latency of the receiver-i
t_i : the time when the receiver-i recovered from failure

In SRSM, the sender eliminates some lost sensor data to merge recovery streams faster. In this way, the recovery streams quickly catch up with the original stream. If there is no existing recovery stream when the receiver-i recovered from the failure, the proposed scheme creates new recovery stream (s) with the following parameters.

R_s : the skip rate of the recovery stream s
B_s : the bandwidth of the recovery stream s
A_s : the data count already delivered by the recovery stream s
D_s : the data that s starts delivery
T_s : the start time of s
G_s : the set of members of the multicast group in s

Then, it adds the receiver-i in G_s. Any receivers that can be merged to the group are added as group members without creating new recovery stream. Whether the receiver-i can be added to the group or not is determined according to the recovery parameters, the acceptable latency, the expected catchup time, and the related latency.

If the data already received by the receiver-i is more than that of the existing recovery stream (s), the expected catch up time of stream (s) to the receiver-i ($C_{s,i}$) and the related latency ($L_{i,s}$) can be calculated as follows:

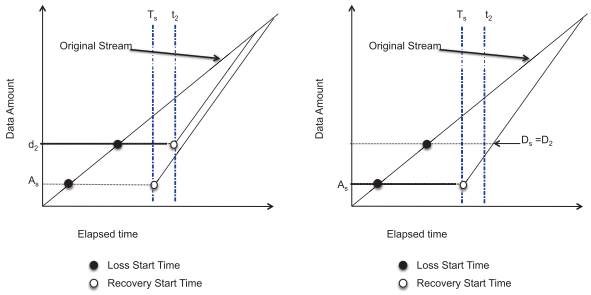

Fig. 4 Case I. Before merging vs After merging.

$$C_{s,i} = \frac{(D_i - D_s)}{R_s \times B_s} + t_i \qquad \text{if } d_i > A_s$$

$$L_{i,s} = C_{s,i} - (d_i + 1) \qquad \text{if } d_i > A_s$$

where $D_i = d_i + r_i \times b_i$ and $D_s = A_s + R_s \times B_s$.

The related latency ($L_{i,s}$) is the latency of the receiver-i related to the existing recovery stream (s). In other words, it is the waiting time of the receiver-i to get recovery data when its received data is more than that of the existing recovery stream (s).

When the data already received by the existing recovery stream (s) is more than that of the receiver-i, the calculations of the expected catch up time for the receiver-i and related latency of existing recovery stream ($L_{s,i}$) are as follows:

$$C_{i,s} = \frac{(D_s - D_i)}{r_i \times b_i} + t_i \qquad \text{if } A_s > d_i$$

$$L_{s,i} = C_{i,s} - D_s \qquad \text{if } A_s > d_i$$

To continue the existing recovery stream or create new recovery stream by dropping the existing one, one of the following conditions must be held. $L_{\min:s}$ is the minimum acceptable latency among the receivers of the existing recovery stream (s).

$$L_{i,s} \le l_i \qquad \text{when } d_i > A_s$$

$$L_{s,i} \le L_{\min:s} \qquad \text{when } A_s > d_i$$

If there is one or more recovery streams, the proposed scheme merges the recovery streams for the following three cases.

4.1.1 Case-I: A Merge-able Stream Exists But Not Reached to d_i

In the first case, the proposed scheme adds the receiver-i recovered from the failure to the existing recovery stream (s) if the following conditions are satisfied.

$$scontains(R_s, r_i, B_s, b_i) \tag{1}$$

$$D_s < D_i \tag{2}$$

$$C_{s,i} - (d_i + 1) \le l_i \tag{3}$$

The function *scontains*() checks whether the skip rate and bandwidth usage of the existing recovery stream s is acceptable for the receiver-i. The second statement (2) means the existing recovery stream has not received the data d_i already received by the receiver-i. The final condition (3) checks whether the related latency satisfies the acceptable latency of the receiver-i.

Figure 4 shows the situation in which there is one existing recovery stream starting at (T_s). At the recovery request time (t_2) of the receiver-2, the conditions for Case I is fulfilled. Therefore, the receiver-2 is added to the existing recovery stream without creating new recovery stream.

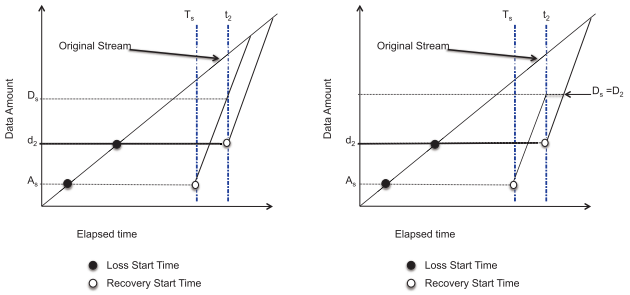

Fig. 5 Case II. Before merging vs. After merging.

4.1.2 Case II. A Merge-able Stream Exists and Already Delivered d_i

Whether the new recovery stream should be generated is checked using the following conditions.

$$scontains(R_s, r_i, B_s, b_i) \tag{4}$$

$$D_s > D_i \tag{5}$$

$$C_{i,s} - D_s \leq L_{min:s} \tag{6}$$

The first condition (4) checks whether the skip rate and bandwidth usage of the receiver-i is acceptable for the existing recovery stream (s). The second condition (5) means that the existing recovery stream has already delivered more data than that required by the receiver-i. The final condition (6) checks whether the related latency satisfies the acceptable latencies of all receivers in the existing recovery stream so that all receivers of the existing recovery stream can wait until new recovery stream delivers D_s.

Figure 5 shows the second situation in which the recovery requested by the receiver-2 has less recovery start data count than that of the existing recovery stream. When the receiver-2 starts its recovery, there is already one existing recovery stream and recovery data count status is ($D_s > D_2$). We assume that the final conditional statement for latency is also satisfied. Therefore, the receivers in the existing recovery stream are added as the receivers of new recovery stream before eliminating the existing recovery stream. Finally, there is only one recovery stream.

4.1.3 Case III. A Coincidental Merge-able Stream Exists

When the receiver and the existing recovery stream (s) have the same recovery data to be delivered, and skip rate and bandwidth usage of s are acceptable for the receiver-i, the receiver is added to the group of the existing recovery stream. The following conditional statements check whether the coincidental merge-able stream exists.

$$scontains(R_s, r_i, B_s, b_i) \tag{7}$$

$$D_s = D_i \tag{8}$$

4.2 Latency-aware Synchronized Recovery Stream Merging (SRSM-L)

We propose the "Latency-aware synchronized recovery stream merging (SRSM-L)" algorithm on the basis of SRSM. We assume the latency-aware environment for SRSM-L in which the receivers specify the acceptable latency to merge the streams. In SRSM-L, the sender does not create a new stream as long as the existing recovery stream can satisfy the receiver. Once the sender receives the request, the sender decides whether to generate a new recovery stream or add the requested receiver as a receiver of one of the existing recovery streams.

Algorithm 1 Algorithm for SRSM-L

```
 1: procedure SUBSCRIBE
 2:     Input: t, i, r_i, b_i, d_i, l_i
 3:     Initialize empty stream array z
 4:     D_i ← d_i + r_i × b_i
 5:     for all s in streams do
 6:         D_s ← A_s + R_s × B_s
 7:         if (D_s = D_i) and (scontains(R_s, r_i, B_s, b_i)) then
 8:             l_i ← l_i − (t − (d_i + 1))
 9:             w_i ← (t − (d_i + 1))
10:             s.addReceiver(i)
11:             L_s ← L_min:s
12:             return
13:         end if
14:         if (D_s < D_i) and (L_i,s ≤ l_i) and (scontains(R_s, r_i, B_s, b_i)) then
15:             l_i ← l_i − (C_s,i − (d_i + 1))
16:             w_i ← (C_s,i − (d_i + 1))
17:             s.addReceiver(i)
18:             L_s ← L_min:s
19:             return
20:         end if
21:         if (R_s! = 1.0) and (D_s > D_i) and (L_s,i ≤ L_s) and
    (scontains(R_s, r_i, B_s, b_i)) then
22:             z.addStream(s)
23:         end if
24:     end for
25:     Create new stream ns
26:     l_i ← l_i − (t − (d_i + 1))
27:     w_i ← (t − (d_i + 1))
28:     ns.addReceiver(i)
29:     for all s in z do
30:         for all u in s.receivers do
31:             w_u ← (C_i,s − D_s)
32:             l_u ← l_u − (C_i,s − D_s)
33:             ns.addReceiver(u)
34:         end for
35:         z.deleteStream(s)
36:     end for
37:     L_ns ← L_min:ns
38:     streams.addStream(ns)
39: end procedure
```

The algorithm for SRSM-L is shown in Algorithm 1 where *streams* represent the streams to be delivered at the current time t by considering the acceptable latency of each receiver-i.

Firstly, the proposed scheme checks whether the receiver-i has the same recovery start data count with that of the recovery stream. Conditional statements from line number (7) to line number (13) checks this situation. If the condition is true, stream merging is performed.

Although the receiver and the recovery stream have different recovery start data count, stream merging may occur depending on the acceptable latency of the receiver-i or the recovery stream. The proposed scheme always delivers the recovery stream with the smaller data amount. If the existing recovery stream has delivered smaller recovery start data than that required by the recov-

ery requested receiver-i, the proposed scheme continues it without generating new recovery stream for the recovery requested receiver-i. The acceptable latency of the receiver-i with larger data amount is considered because it has to wait until the existing recovery stream (s) delivers its recovery start data count. Therefore, the related latency ($L_{i,s}$) must be less than or equal to the acceptable latency (l_i) so that the receiver-i can be added to the existing recovery stream (s). The acceptable latency (l_i) of recovery requested receiver-i is updated for time (t) so that the sender considers the remaining latency of the receiver-i in merging with other streams or other receivers at the same time (t). Therefore, the acceptable latency of the recovery stream (L_s) is modified with the minimum acceptable latency among the receivers of the recovery stream ($L_{min:s}$) whenever a receiver is added to the existing recovery stream. Moreover, the latency (w_i) of receiver-i is needed to be updated to find its maximum latency throughout its processing time. The conditional statements from line number (14) to line number (20) are grouping receivers for the existing recovery stream. The summary of this case can be described like this:

$G_s = G_s \cup \{i\}$ when $L_{i,s} \leq l_i$ and $d_i > A_s$

In case of the existing recovery stream with more recovery start data count ($D_s > D_i$), the proposed scheme generates new recovery stream for the recovery requested receiver and eliminate the existing recovery stream. The acceptable latency of the receivers from the existing recovery stream needs to be considered. In checking the acceptable latency, the related latency ($L_{s,i}$) must be less than or equal to the acceptable latency (L_s) so that new recovery stream will be generated by dropping the existing recovery stream. Conditional statements starting from line number (21) to line number (23) describe the above situation. The summary of this case is

$G_{ns} = G_s \cup \{i\}$ when $L_{s,i} \leq L_s$ and $A_s > d_i$

delete G_s

where G_{ns} is the group of receivers in new recovery stream.

The receiver-i and receivers of the existing recovery stream are added to new recovery stream (ns) before dropping the existing recovery stream. Then, the existing recovery stream is removed and the acceptable latency of new recovery stream (L_{ns}) is assigned with the minimum acceptable latency among its receivers ($L_{\min:ns}$). The conditional statements described from line number (25) to line number (38) correspond to new stream generation and latency assignment of new stream and its receivers.

4.3 Bandwidth-dependent Synchronized Recovery Stream Merging (SRSM-B)

We also propose the "Bandwidth-dependent synchronized recovery stream merging (SRSM-B)" algorithm on the basis of SRSM. In the real scenario, the available bandwidth of the sender may dynamically change time by time. Instead of the receiver requirements, SRSM-B prioritizes the sender. In SRSM-B, the sender considers the receivers' latencies as much as possible in merging streams. Firstly, it performs streams merging by considering the receivers' acceptable latencies. Then, it checks the required bandwidth and the sender's available bandwidth. In case of the larger bandwidth requirement, it agains merges some streams to adapt to the sender's available bandwidth. As a result, some or all portions of the receivers may exceed the acceptable latencies. However, SRSM-B merges the streams with the least latency difference to make latency exceeded value as small as possible. The sender needs to have the following parameter values for bandwidth-dependent recovery stream delivery.

Algorithm 2 Algorithm for SRSM-B

```
1:  procedure MERGE
2:      Performs SUBSCRIBE procedure of SRSM-L for all recovery requested
    receivers at time t
3:      if (n_t ≤ a_t) then
4:          break
5:      else
6:          s_1 ← stream_{t(ori)}
7:          s_2 ← stream_{t(ori)}
8:          s_t.deleteStream(s_2)
9:          q = n_t − 1
10:         m = q/(a_t − 1)
11:         for (i = 1 to q) do
12:             s_3 ← stream_{t(min)}
13:             for all s in mergestream do
14:                 if (s_count < m) then
15:                     add all receivers of s_3 to s
16:                     if (R_s > R_{s_3}) then
17:                         R_s = R_{s_3}
18:                     end if
19:                     if (B_s > B_{s_3}) then
20:                         B_s = B_{s_3}
21:                     end if
22:                     for all u in s_3.receivers do
23:                         w_u ← (C_{s,s_3} − D_{s_3})
24:                         l_u ← l_u − (C_{s,s_3} − D_{s_3})
25:                     end for
26:                     s_t.deleteStream(s_3)
27:                     increment s_count by 1
28:                     return
29:                 end if
30:             end for
31:             mergestream.addStream(s_3)
32:             s_{3count} = 1
33:             s_t.deleteStream(s_3)
34:             increment i by 1
35:         end for
36:         mergestream.addStream(s_1)
37:     end if
38: end procedure
```

a_t : the number of streams allowed for available bandwidth

s_t : the streams to be delivered at time t

n_t : the number of streams to be delivered at time t

Here, (s_t, n_t) can be determined after performing SUBSCRIBE procedure of SRSM-L for all recovery requested receivers at time t.

The algorithm SRSM-B is shown in Algorithm 2. Depending on the time-variant number of streams (a_t) allowed by the sender and the required number of streams (n_t), the proposed scheme de-

cides whether stream merging should be performed or not. If the bandwidth requirement for the recovery stream delivery is within the limit of the sender's available bandwidth, no further stream merging is necessary. This situation is described in line number (3) and (4). Here, we assume that the sender's allowed number of streams is always greater than 1.

For situation in which the bandwidth requirement is beyond the sender's limit, more than one stream from the (s_t) are merged into the new stream. Here, the new stream is denoted as the merged stream (*mergestream*). Stream merging is performed among the streams except the original stream ($stream_{t(ori)}$). For merging other streams except the original stream, both the required number of streams and allowed number of streams are reduced by one because we assume that the original stream is a merged stream. Then, the number of streams to be merged into a merged stream (m) is decided by dividing the reduced number of streams (q) by the reduced allowed number of stream ($a_t - 1$). The conditional statements are described from line number (7) to line number (10).

Other streams to be merged are chosen according to the data count already received. In general, different streams with minimum data count difference are merged so that the latency of receivers is not much larger than their acceptable latencies. We refer the stream with minimum data count already received at time t as $stream_{t(\min)}$. To achieve the safe delivery, the skip rate and bandwidth of the merged stream are set using the minimum skip rate and minimum bandwidth among all the streams to be merged. Then, the scheme modifies the latency of receivers from the streams to be merged into a merged stream. The line numbers from (11) to (35) shows merging multiple streams. In Algorithm 2, the initialization of a merged stream is described from line number (31) to (34). Finally, the original stream is added as a merged stream at line number (36).

5. Evaluations by Simulations

We conducted the simulation evaluations for our proposed schemes. We evaluated the number of streams, the average number of streams, the number of receivers that exceed acceptable latency and average of maximum latency for two different failure/recovery patterns.

5.1 Simulation Setup

We evaluated the method for two failure/recovery simulation scenarios: 1) Poisson scenario and 2) Gaussian scenario. We selected these scenarios because they are typical failure situations. Poisson scenario corresponds to a random failure situation and Gaussian scenario corresponds to a burst failure situation. For example, the errors of smartphones (e.g., starting/stopping an application) in a wide area may follow the Poisson scenario and the errors when the receivers in a train that goes into tunnel may follow the Gaussian scenario.

The common simulation setup is shown in **Table 1**, where p is a receiver in the simulation. We used a notion of virtual 'unit time', which can correspond to a second, a minute, or an hour, in the simulations. The evaluation results can be interpreted to the various situations by translating the unit time. The acceptable

Table 1 Common simulation setup.

Parameters	Value
Simulation time	1,000 (unit time)
Number of receivers	100
r_p	2.0
b_p	1.0
l_p	200 (unit time)
a_t	2
Number of trials	1,000

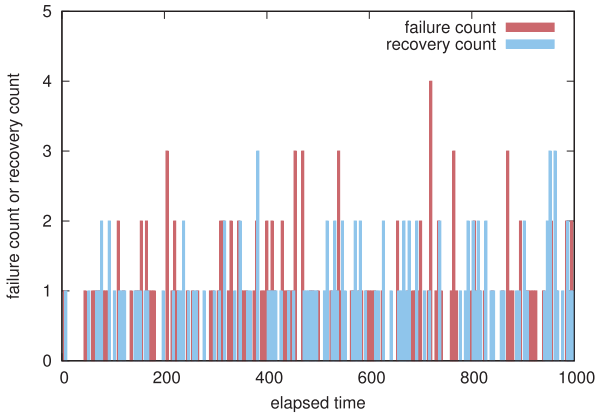

Fig. 6 Loss and recovery model of the second trial (Poisson).

bandwidth ($b_p = 2$) is selected because we assume IoT applications, in which plenty of network bandwidth is not available. The simulation time (= 1,000 unit time) is decided as an enough duration for the stable simulation. The number of receivers (= 100) is an enough number that we can reproduce the congested situation. $r_p = 2$ and $l_p = 200$ are selected as typical parameters to see the basic behavior. Same as the unit time, both r_p and l_p can be translated for various situations. If we want to translate $l_p = 200$ as a minute, which is a typical interval to analyze sensor data, then the unit time need to be interpreted as 1/3 second. To see the basic performance, we assume all receivers have the same requirements for the recovery.

In the Poisson scenario, a receiver encounters failure multiple times within the simulation time because the number of receivers is limited in the simulation. The average number of failure occurred within a unit time is set as 8. **Figure 6** shows the failure/recovery pattern of the second trial for the Poisson scenario. The x-axis represents the elapsed time and y-axis represents the number of failure or recovery.

In the Gaussian scenario, a receiver can encounter a failure at most one time within the simulation time. In all trials, we assume half of the receivers encounter failure around 500 unit time following the Gaussian distribution. The standard deviation of the Gaussian distribution is set as 50. The failure interval also follows the Gaussian distribution. The average failure interval (fi) is set from $fi = 100$ to 200 by the increment of 10. The standard deviation of the average failure interval is set as 10. **Figure 7** describes the assumed loss and recovery model of the second trial for the Gaussian scenario.

As a comparison method, the simple Piggybacking scheme is used. In Piggybacking, a slow stream is generated when no slow stream exists within a time window W. Otherwise, a fast stream is generated. We assume that the slow stream has $R_s = 1.0$

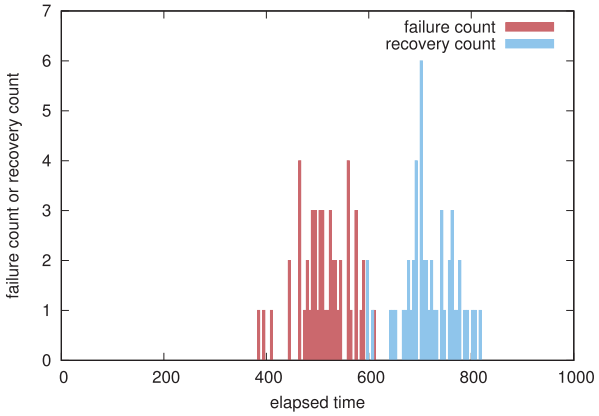

Fig. 7 Loss and recovery model of the second trial (Gaussian).

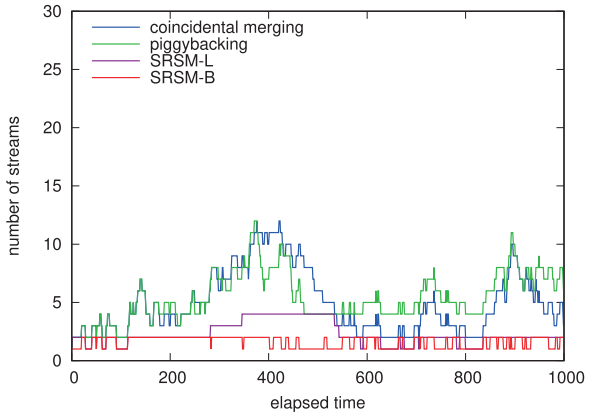

Fig. 8 Number of streams in the Second Trial (Poisson).

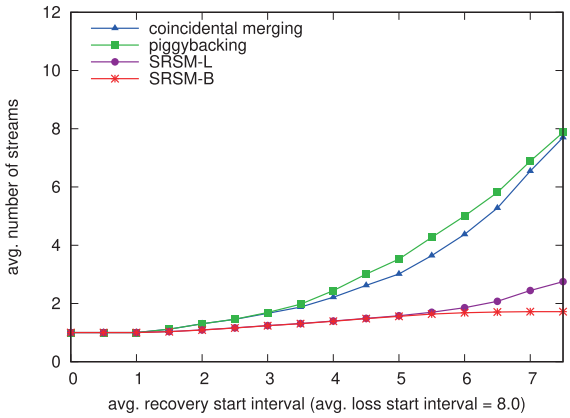

Fig. 9 Average number of the streams (Poisson).

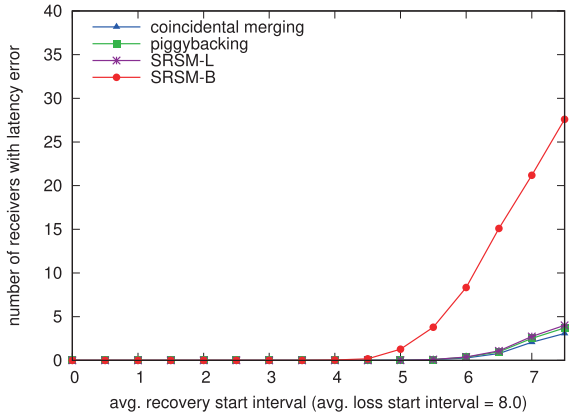

Fig. 10 Number of the receivers that exceed acceptable latency (Poisson).

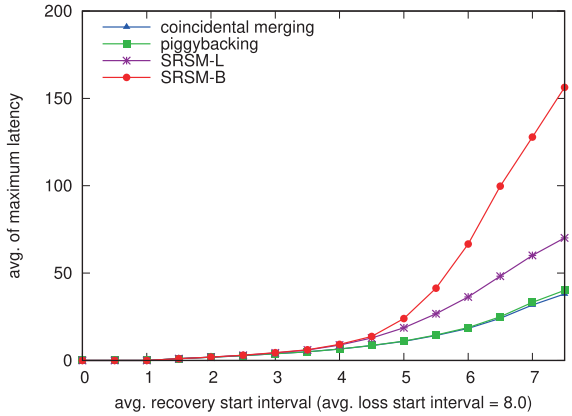

Fig. 11 Average of the maximum latency (Poisson).

and $B_s = 1.0$ while the fast stream delivers with $R_s = 2.0$ and $B_s = 1.0$. The window size W is set as 40. Another comparison method is Coincidental merging in which the receiver joins the existing stream when it has exactly same parameter values of the existing stream.

5.2 Simulation Results

5.2.1 Poisson Scenario

Figures 8–11 shows the simulation results for the Poisson scenario. We denote λ as the average number of failures within a unit time and μ as the average number of recoveries within a unit time. For the Poisson scenario, we set $(\lambda = \frac{1}{8})$ and $(\mu = \frac{1}{h})$ to define the average number of failure and recovery respectively. The value of (h) is set from 0.5 to 7.5 with the incrementing value 0.5. The comparison of the number of streams resulted from the second trial is shown in Fig. 8. The parameters for the second trial are $(\lambda = \frac{1}{8})$ and $(\mu = \frac{1}{7.5})$. Figure 9 shows the average stream count by changing the average interval between recoveries, which corresponds to $\frac{1}{\mu}$. Figure 10 shows the number of the receivers that exceeds the acceptable latency and Fig. 11 shows the average of the maximum latency for all simulation trial.

According to these results in Poisson scenario, SRSM-L showed good performance than Piggybacking and Coincidental merging, when the average interval between recoveries is large, which means the number of receivers in simultaneous failure is large, in terms of the number of streams. The performance of Piggybacking is worse around the average interval value between 4 and 8. That is because more stream generation occurs in Piggybacking and the random failure gives disadvantage for it. The average number of streams (i.e., network bandwidth used by the sender) in SRSM-L is reduced about 52% compared with Piggybacking and Coincidental merging when the average interval between recoveries is 7.5. On the other hand, as expected, the number of streams of SRSM-B did not exceed 2. However, as Fig. 10 shows, the number of receivers which exceeded 200 unit time increases. As the Fig. 11 shows, the average latency increases to around 250 when the loss start interval = 7.5.

As a summary, in the Poisson scenario, we can say SRSM-L can keep good performance even when there are frequent failures and recoveries. SRSM-L showed balanced results which keep

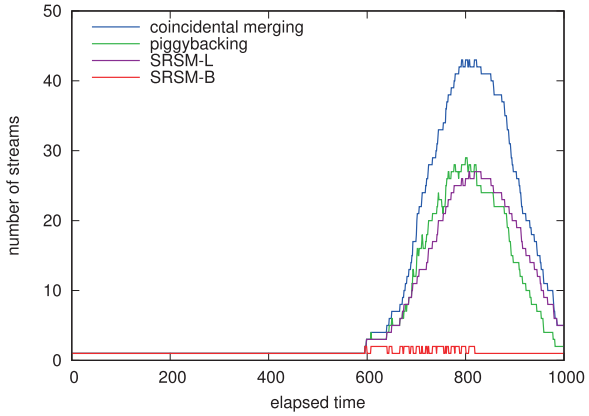

Fig. 12 Number of the streams in Second Trial (Gaussian).

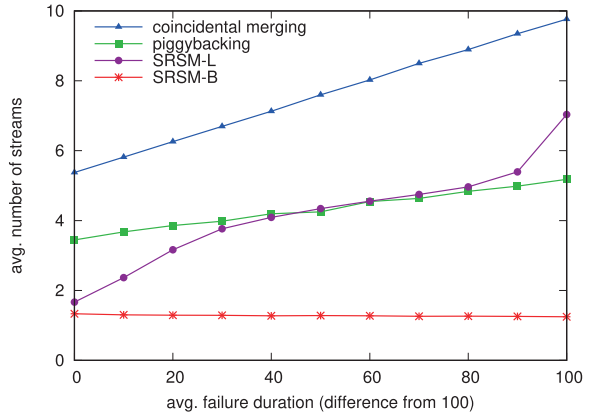

Fig. 13 Average number of the streams (Gaussian).

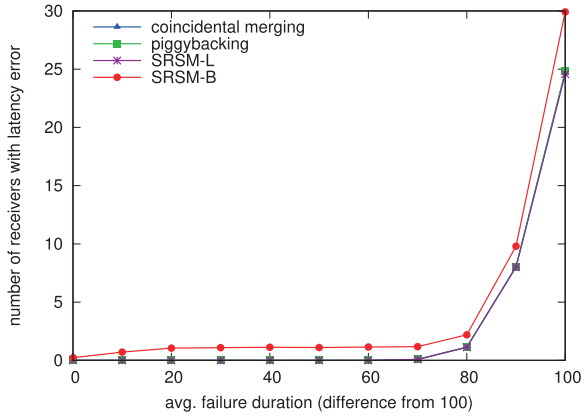

Fig. 14 Number of the receivers that exceed acceptable latency (Gaussian).

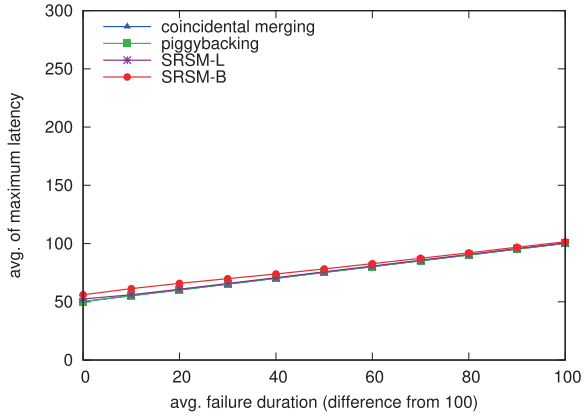

Fig. 15 Average of the maximum latency (Gaussian).

the bandwidth usage and the delivery latency small. About the SRSM-B, we confirmed that the scheme could keep the number of streams constant, but increased delivery latency. Therefore, SRSM-B is suitable only when the application does not need to limit the latency in the Poisson scenario.

5.2.2 Gaussian Scenario

Figures 12–15 shows the simulation results for the Gaussian scenario. For the Gaussian scenario, the simulation is performed using different average failure interval (fi) where (fi = 100 to 200) (unit time) with standard deviation 10.

Figure 12 shows the result of stream count for the second trial. The parameters for the second trial are mean failure start time (500) with standard deviation (10), and average failure interval (200) with standard deviation (10). Depending on the average length of the failure interval, the recovery start time for the receivers is slightly different. As a result, more recovery stream generation occurs nearly at the same time. The comparison of all schemes in terms of the average number of stream is illustrated in Fig. 13. Figure 14 shows the number of the receivers that exceeds the acceptable latency and Fig. 15 shows the average of the maximum latency for all simulation trial.

The Coincidental merging showed worst performance in terms of the number of streams. The Piggybacking showed better performance than Coincidental merging. It showed similar performance with SRSM-L because in the Gaussian scenario, simultaneous errors can be accommodated to the merging stream. The number of streams increases in SRSM-L as the average length of the failure interval increases. That is because the number of long failure receivers, in which the failure interval exceeds 200, increases as shown in Fig. 14. The average number of streams in SRSM-L is nearly equal to that of Piggybacking when the difference is from 40 to 80. As expected, SRSM-B could keep the number of streams no more than 2 during the simulation time. The average number of the receivers that exceed the acceptable latency and the average max latency becomes larger when the fi becomes larger in the all schemes. As showin in the Fig. 14 and Fig. 15, SRSM-B showed larger latency than other schemes. However, the increases in the latency was small.

As a summary, in the Gaussian scenario, we can say SRSM-B could keep good performance. SRSM-B clould keep the max bandwidth usage constant and the delivery latency overhead small. SRSM-L, could not keep the number of streams small, when the difference is large, that means the failure duration is longer than the threshold l_p.

6. Conclusion

We proposed efficient lost sensor data delivery schemes for sensor data multicasting. They are “Latency-aware synchronized recovery stream merging (SRSM-L)” and “Bandwidth-dependent synchronized recovery stream merging (SRSM-B),” which are suitable for the IoT applications. We evaluated the performance in two simulation scenarios that corresponds to the random failures and the burst failures. Simulation results are shown in terms of the number of streams, the average number of streams, the

number of receivers that exceeds the acceptable latency and the average of the maximum latency.

According to the evaluation results, we found that the performance of SRSM-L can save about 52% network bandwidth on the sender in the frequent random failure situation, comparing with existing schemes, keeping the acceptable latency. We also found that in the burst failure situation, SRSM-B shows best performance even when the failure duration is long. It could keep the maximum number of the streams constant and the latency overheads small.

In the future work, we will do more detailed evaluations to confirm whether our scheme is applicable for more applications or situations. Then we will implement the proposal method on an actual pub/sub multicasting platform for reliable IoT applications. In addition, further improvement of delivery load reduction mechanism, the recovery stream generation scheme using distributed caches on the network, dynamically applying the appropriate scheme (SRSM-L or SRSM-B) according to the observations and feedback, also form part of our future work.

Acknowledgments This research was supported by Grants-in-Aids for Scientific Research (B) numbered 15H02702.

References

[1] Hodges, S., Taylor, S., Villar, N., Scott, J., Bial, D. and Fischer, P.T.: Prototyping connected devices for the internet of things, *J. Computer*, Vol.46, No.2, pp.26–34, IEEE (2013).
[2] Gubbi, J., Buyya, R., Marusic, S. and Palaniswami, M.: Internet of Things (IoT): A vision, architectural elements, and future directions, *J. Future Generation Computer Systems*, Vol.29, No.7, pp.1645–1660, Elsevier (2013).
[3] Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M. and Ayyash, M.: Internet of things: A survey on enabling technologies, protocols, and applications, *J. IEEE Communications Surveys and Tutorials*, Vol.17, No.4, pp.2347–2376, IEEE (2015).
[4] Patel, M., et al.: Mobile-Edge Computing - Introductory Technical White Paper, *ETSI MEC white paper*, V1 18-09-14, 36 pages (2014).
[5] Porambage, P., Braeken, A., Schmitt, C., Gurtov, A., Ylianttila, M. and Stiller, B.: Group key establishment for enabling secure multicast communication in wireless sensor networks deployed for IoT applications, *IEEE Access*, No.3, pp.1503–1511 (2015).
[6] Jiang, D., Xu, Z. and Lv, Z.: A multicast delivery approach with minimum energy consumption for wireless multi-hop networks, *Telecommunication Systems*, Vol.62, No.4, pp.771–782 (2016).
[7] Quinn, B. and Almeroth, K.: IP multicast applications: Challenges and solutions, RFC 3171 (2001).
[8] Eugster, P.T., Felber, P.A., Guerraoui, R. and Kermarrec, A.: The many faces of publish/subscribe, *J. ACM Computing Surveys* (*CSUR*), Vol.35, No.2, pp.114–131, ACM (2003).
[9] MQTT Version 3.1.1, https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.pdf
[10] Advanced Message Queuing Protocol, http://www.amqp.org
[11] Banno, R., Takeuchi, S., Takemoto, M., Kawano, T., Kambayashi, T. and Matsuo, M.: Designing overlay networks for handling exhaust data in a distributed topic-based Pub/Sub architecture, *J. Inf. Process.*, Vol.23, No.2, pp.105–116, IPSJ (2015).
[12] Teranishi, Y., Banno, R. and Akiyama., T.: Scalable and locality-aware distributed topic-based pub/sub messaging for IoT, *Proc. IEEE GLOBECOM 2015* (2015).
[13] Verbelen, V., Simoens, P., Turck, F.D. and Dhoedt, B.: Cloudlets: bringing the cloud to the mobile user, *Proc. 3rd ACM Workshop on Mobile Cloud Computing and Services*, pp.29–36 (2012).
[14] Dan, A., Shahabuddin, P., Sitaram, D. and Towsley, D.: Channel allocation under batching and VCR control in video-on-demand systems, *J. Parallel and Distributed Computing*, Vol.30, No.2, pp.168–179, Elsevier (1995).
[15] Golubchik, L., Lui, J. and Muntz, R.: *Reducing I/O Demand in Video-on-Demand Storage senders*, Vol.23, No.1, Ottawa, Ontario, Canada, pp.25–36, ACM (1995).
[16] Aggarwal, C.C. et al.: On optimal piggyback merging policies for Video-on-Demand systems, *Proc. ACM Measurement and Modeling of Computer Systems*, pp.200–209 (1996).
[17] Golubchik, L., Lui, J.C. and Muntz, R.R.: On optimal batching policies for video-on-demand storage servers, *Journal of Multimedia Systems*, pp.140–155 (1996).
[18] Aggarwal, C.C., Wolf, J.J. and Yu, P.S.: Adaptive piggybacking: A novel technique for data sharing in video-on-demand storage servers, *Proc. 3rd IEEE Intl. Conf. Multimedia Computing and Systems*, pp.253–258 (1996).
[19] Carter, S.W. and Long, D.D.: Improving Video-onDemand sender Efficiency Through Stream Tapping, *Proc. 6th ACM Intl. Conf. Computer Communications and Networks*, pp.200–207, IEEE (1997).
[20] Hua, K.A., Cai, Y. and Sheu, S.: Patching: A multicast technique for true video-on-demand services, *Proc. 6th ACM Intl. Conf. Multimedia*, Bristol, United Kingdom, pp.191–200, ACM (1998).
[21] Eager, D.L. and Vernon, M.K.: Dynamic Skyscraper Broadcasts for Video-on-Demand, *Intl. Workshop on Multimedia Information Systems*, pp.18–32 (1998).
[22] Lau, S.W. et al.: Merging video streams in a multimedia storage server: complexity and heuristics, *ACM Multimedia Systems Journal*, Vol.6, No.1, pp.29–42 (1998).
[23] Lau, S.W. et al.: Improving bandwidth efficiency of video-on-demand servers, *Journal of Computer Networks*, Vol.31, No.1, pp.111–123, Elsevier (1999).
[24] Coffman, E.G., Jelenkovic, Jr., P. and Momcilovic, P.: Provably Efficient Stream Merging, *Proc. Web Caching Workshop*, pp.63–74, CiteSeer (2001).
[25] Bar-Noy, A., Goshi, J., Ladner, R.E. and Tam, K.: Comparison of stream merging algorithms for media-on-demand, *Multimedia Systems*, pp.411–423, IEEE (2004).
[26] Teranishi, Y., Win, E.K, Yoshihisa, T. and Shimojo, S.: A Sensor Data Stream Recovery Scheme for Event-Driven IoT Applications, *Proc. IEEE GLOBECOM 2017* (2017).

Ei Khaing Win received her B.C.Sc. degree from University of Computer Studies (Mandalay), Myanmar, in 2004 and her M.C.Sc. degree from Computer University (Mandalay), Myanmar, in 2010, respectively. From 2007 to 2009, she was appointed as a demonstrator of Computer University (Hinthada), Myanmar. From 2009 to 2015, she became a tutor of University of Technology (Yatanarpon Cyber City), Myanmar. Since October 2015, she has been an assistant lecturer of University of Technology (Yatanarpon Cyber City), Myanmar. Her research interests include security and stream data processing. She is a member of IPSJ.

Tomoki Yoshihisa received his Bachelor's, Master's, and Doctor's degrees from Osaka University, Osaka, Japan, in 2002, 2003, 2005, respectively. Since 2005 to 2007, he was a research associate at Kyoto University. In January 2008, he joined the Cybermedia Center, Osaka University as an assistant professor and in March 2009, he became an associate professor. From April 2008 to August 2008, he was a visiting researcher at University of California, Irvine. His research interests include video-on-demand, broadcasting systems, and webcasts. He is a member of IPSJ, IEICE, and IEEE.

Yoshimasa Ishi received his B.E. degree from Kyoto Institute of Technology, Japan, in 2004 and his M.I. degree from Osaka University, Japan, in 2006, respectively. From 2006 to 2008, and from 2012 to 2015, he was a specially appointed researcher of Cybermedia Center, Osaka University. From 2008 to 2012, he was a specially appointed researcher of Graduate School of Information Science and Technology, Osaka University. Since January 2017, he has been a specially appointed researcher of Institute for Datability Science, Osaka University. His research interests include technologies for distributed network systems and its development. He is a member of IPSJ.

Tomoya Kawakami received his B.E. degree from Kinki University in 2005 and his M.I. and Ph.D. degrees from Osaka University in 2007 and 2013, respectively. From 2007 to March 2013 and from July 2014 to March 2015, he was a specially appointed researcher at Osaka University. From April 2013 to June 2014, he was a Ph.D. researcher at Kobe University. Since April 2015, he has been an assistant professor at Nara Institute of Science and Technology. His research interests include distributed computing, rule-based systems, and stream data processing. He is a member of IPSJ and IEEE.

Yuuichi Teranishi received his M.E. and Ph.D. degrees from Osaka University, Japan, in 1995 and 2004, respectively. From 1995 to 2004, he was engaged Nippon Telegraph and Telephone Corporation (NTT). From 2005 to 2007, he was a Lecturer of Cybermedia Center, Osaka University. From 2007 to 2011, he was an associate professor of Graduate School of Information Science and Technology, Osaka University. Since August 2011, He has been a research manager and project manager of National Institute of Information and Communications Technology (NICT). He received IPSJ Best Paper Award in 2011. His research interests include technologies for distributed network systems and applications. He is a member of IPSJ and IEEE.

Shinji Shimojo received his M.E. and Ph.D. degrees from Osaka University in 1983 and 1986, respectively. He was an assistant professor with the Department of Information and Computer Sciences, Faculty of Engineering Science at Osaka University from 1986, and an associate professor with Computation Center from 1991 to 1998. During this period, he also worked for a year as a visiting researcher at the University of California, Irvine. He has been a professor with the Cybermedia Center (then the Computation Center) at Osaka University since 1998, and from 2005 to 2008, and since 2016, he had/has been the director of the Center. He is an executive researcher at National Institute of Information and Communications Technology and a director of Network Testbed Research and Development Promotion Center. His current research work is focusing on a wide variety of multimedia applications, peer-to-peer communication networks, ubiquitous network systems, and Grid technologies. He was awarded the Osaka Science Prize in 2005. He is a member of IEEE, and IEICE and IPSJ fellow.