A Lost Sensor Data Recovery Scheme for Faster Data Streams Merging

EI KHAING WIN[†1] TOMOKI YOSHIHISA[†1] YOSHIMASA ISHI[†2]
TOMOYA KAWAKAMI[†3] YUUICHI TERANISHI[†1,4] SHINJI SHIMOJO[†1]

Abstract: Due to the recent development of sensing technologies, streaming servers deliver sensor data streams such as video or temperature data streams to many clients. In simple delivery scheme, the servers deliver streams to each client one by one though the data themselves are the same. When a client encounters data loss because of network disconnections or access failures, the server delivers a recovery stream to the client. As the number of the delivering streams increases, the server's management load such as the computational power, memory usage, communication traffic, etc. increases. To reduce the number of the delivering streams, in this paper, we propose a lost sensor data recovery scheme for faster data streams merging. Our proposed scheme reduces the number of the streams managed by the server by merging some of them. To merge streams faster, the server removes some data from the recovery streams. From our evaluations, we confirmed that our proposed scheme can faster reduce the number of the streams.

Keywords: sensor data stream delivery, IoT (Internet Of Things), streaming and distribution, sensor network

1. Introduction

Due to the recent proliferation of sensors such as cameras or temperature sensors, sensor data stream delivery technology takes an important role in various applications. For example, some smartphones request a live camera stream to the server and the clients check the current trip destination situations. In this case, the servers deliver video data streams to many clients. In simple delivery scheme, the servers deliver streams to each client one by one though the data themselves are the same. Therefore, when a client encounters data loss, the server delivers a new recovery stream to the client. Examples follow:

- A client requests a live camera stream to show the current situation of the area covered by the camera. When the client loses its network connection because electromagnetic waves can not reach to the client, the client loses some data in the stream. In this case, the client requests the lost data for the recovery to the server.
- Some smartphones receive a real time temperature data stream to check the climate of their travel destinations. In cases that one of them moves to underground, it loses some of the temperature data since the client cannot catch electromagnetic waves in underground. So, the client requests the recovery stream. In this case, the server delivers different streams to each client one by one though the data themselves are the same.

In the above cases, the server delivers a new stream to the client for the recovery. As the number of the delivering streams increases, the server's management load such as the computational power, memory usage, communication traffic, etc. increases. The servers' load can be relieved by reducing the number of streams. Therefore, some methods to reduce the number of the streams have been proposed in ([1]-[3]). Most of them adopt the stream merge technique, one of the major techniques to reduce the number of streams. In the stream merge technique, some streams that include the same data are merged to one stream. The clients receive and buffer the merged stream and use the data included in the merged stream when needed. Therefore, the key issue for lost data recovery in stream delivery is how to merge some streams faster. Suppose the case when the server delivers a same stream to some clients and one of them encounters data loss. The client requests a recovery stream to the server and the server delivers the recovery stream to the client. Although the recovery stream lags behind the original stream for the other clients that do not encounter data loss, the data included in the recovery stream and the original stream are the same. So, by merging the recovery streams to the original stream, the number of the streams can be reduced.

Hence, in this paper, we propose a lost sensor data recovery scheme for faster data streams merging. In our proposed scheme, the server prepares a recovery stream starting with the next sensor data from the last data the client received before the network disconnection. The number of recovery streams is large when many clients request recovery streams. Instead of continuously delivering the recovery streams, the server eliminates some sensor data from recovery streams so that the recovery streams quickly catch up with the original stream. In this way, our proposed scheme reduces the number of streams using faster data streams merging. The number of eliminated data is a parameter correlated to the client's desired catch-up time with the original stream. From our evaluations, we confirm that our proposed sensor data recovery scheme can faster reduce the number of the streams.

The paper is organized as follows. Section II describes related works. In Section III, our assumed system model is shown firstly. Then, simple and proposed schemes are explained in detail. The paper shows evaluation and performance comparison in Section IV. Finally, we conclude the paper in Section V.

†1 Osaka University
†2 PIAX inc Osaka
†3 Nara Institute of Science and Technology
†4 National Institute of Information and Communications Technology

2. Related Works

There are some methods to reduce the server's loads for sensor data stream delivery. The servers' loads for sensor data stream delivery are based on the number of I/O operations. So, some methods reduce the servers' loads by reducing the I/O operations. In [1], batch stream processing system finding I/O and computation redundancies for optimizations was proposed. Its effectiveness how I/O operations are reduced was evaluated by query processing system Comet. However, the system focuses only on the computational processing with I/O operations. The system does not adopt the stream merge technique adopted in our proposed scheme.

In [2], two stream buffers handling methods were proposed for graphics processing units stream-based computing platform. In the first method, stream buffer addresses are stored in statically allocated memory called pinned memory. Therefore, graphic processing units (GPU) needs to access those addresses to perform stream-based computing. In the second method, buffer addresses are stored on shared memory of (GPU). In this way, GPU avoids overhead for pinned memory access.

In [3], optimization algorithm was proposed for distributed stream delivery. Instead of manually tuning the batch size to reduce main memory consumption and degree of parallelism for each node, optimal values are calculated automatically in the system. As optimal batch size and degree of parallelism is used for distributed stream delivery, throughput is achieved with optimization time that is less than one second.

There are some methods to recover data loss for sensor data stream delivery. To reduce the servers' load such as bandwidth, and I/O overhead, merging schemes [4, 5, 6, 7] have been proposed for video-on-demand servers. These schemes are called patching, dynamic skyscraper, and piggybacking. However, these schemes do not consider the clients' desired catch-up time with the earlier stream or real-time sensor data stream and there is no such scheme.

The main difference between the conventional methods and our proposed method is stream merging focusing on recovery streams. This leads faster reduction of the number of streams.

3. Proposed Method

In this section, we firstly describe our assumed system model. Then, lost sensor data delivery in simple scheme and proposed scheme is also explained in detail.

3.1 Assumed System Model

Fig. 1 shows our assumed system model. For sensor data stream delivery, server 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 loss sensor data recovery and dropping of some streams after successful merging. In Fig. 1, we assume the data rate of stream delivery is 30 frames per second and the server delivers some streams like sensor data streams 1 and 2. For simplicity, the latency for delivering data from the server to the clients is ignored in our assumed model.

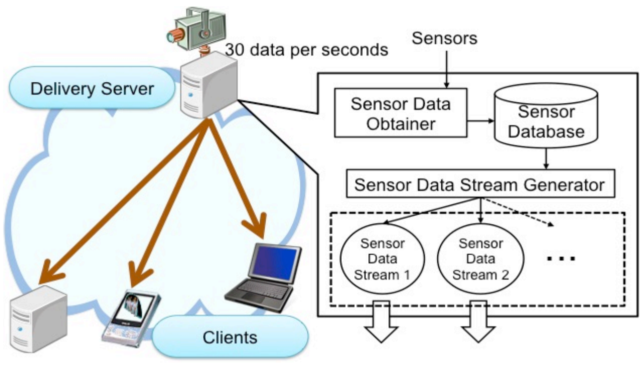

Fig. 1. Our assumed system model

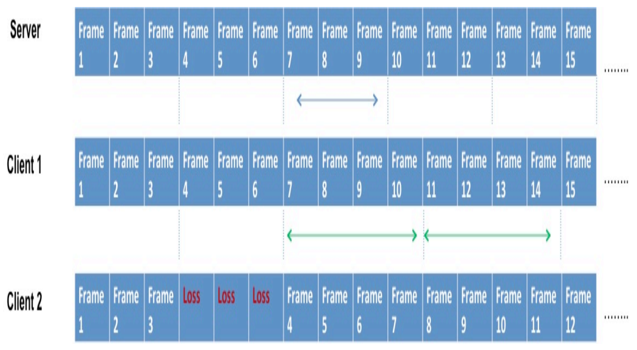

Fig. 2. An example for the simple scheme

Table 1. The number of the streams in the simple scheme

	1s	2s	3s	4s	5s
Server	30 [fps]	30 [fps]	30 [fps]	30 [fps]	30 [fps]
Client1	30 [fps]	30 [fps]	30 [fps]	30 [fps]	30 [fps]
Client2	(loss)	40 [fps]	40 [fps]	40 [fps]	30 [fps]
Streams	1	2	2	2	1

3.2 Simple Scheme

In this section, we explain a simple scheme for sensor data stream delivery. In the simple scheme, lost sensor data retransmission is performed sequentially from the beginning point of data loss with higher frame rate.

Fig. 2 shows sensor data stream delivery by assuming that the server delivers data with 3 frames per second rate. Client2 encounters network disconnection at the second delivery time. It starts receiving the recovery stream at the third delivery time. In this simple scheme, the server delivers new recovery stream with higher frame rate for Client2 to reduce retransmission times until it catches up with the original stream.

Based on our assumed system model, the number of the data stream in simple scheme is shown in Table 1. Suppose the case

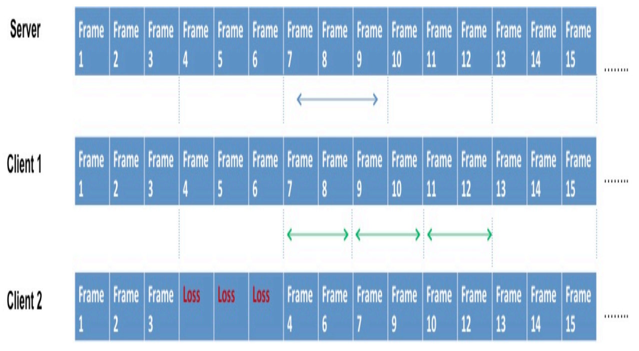
Server
Client 1
Client 2
Frame 1 Frame 2 Frame 3 Frame 4 Frame 5 Frame 6 Frame 7 Frame 8 Frame 9 Frame 10 Frame 11 Frame 12 Frame 13 Frame 14 Frame 15
Frame 1 Frame 2 Frame 3 Loss Loss Loss Frame 4 Frame 6 Frame 7 Frame 9 Frame 10 Frame 12 Frame 13 Frame 14 Frame 15

Fig. 3. The idea of our proposed scheme

Table 2. The number of the streams in our proposed scheme

	1s	2s	3s	4s	5s
Server	30 [fps]	30 [fps]	30 [fps]	30 [fps]	30 [fps]
Client1	30 [fps]	30 [fps]	30 [fps]	30 [fps]	30 [fps]
Client2	(loss)	20 [fps]	20 [fps]	30 [fps]	30 [fps]
Streams	1	2	2	1	1

that Client2 encounters network disconnection for the first 30 frames. The recovery stream includes data for missing 30 frames. The server sends the recovery stream with faster frame rate (40 frames per second) so that the recovery stream approaches to the original stream by 10 frames per second. That rate (40 frames per second) continues as the server delivers the recovery stream.

As a result, there are two streams for the server to deliver the original stream to Client1 and the recovery stream to Client2. The total number of the streams delivered by the server is (1+2+2+2+1=8). When there are so many clients requesting different lost data, then it will result large number of data streams for server. Thus, the server's load increases as the number of the streams increases.

3.3 Proposed Scheme

In our proposed scheme, the server delivers the recovery streams according to the timestamps at which the clients received the last sensor data. In addition to redundant recovery streams avoidance, the server eliminates some lost sensor data to merge recovery streams faster. In this way, recovery streams quickly catch up with the original stream. Here, the original streams mean the sensor data streams that the clients without encountering data losses receive. When the recovery streams catch up with the original stream, the server can deliver the same original stream to the clients that were receiving the lost sensor data. Therefore, the server does not need to generate the recovery stream and the number of the streams is reduced.

Fig. 3 shows the situation in which the proposed scheme uses lower sensor data delivery rate by eliminating some sensor data. If the server delivers the lost sensor data with the data rate of 20 frames per second that is the reduced rate of 45 frames, the

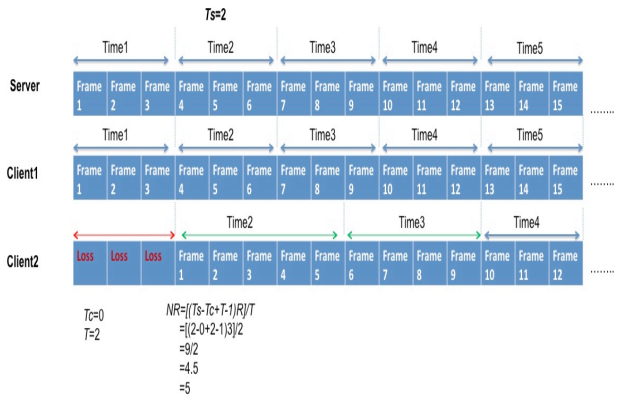
Ts=2
Time1 Time2 Time3 Time4 Time5
Server
Client1
Client2
Frame 1 Frame 2 Frame 3 Frame 4 Frame 5 Frame 6 Frame 7 Frame 8 Frame 9 Frame 10 Frame 11 Frame 12 Frame 13 Frame 14 Frame 15
Loss Loss Loss Frame 1 Frame 2 Frame 3 Frame 4 Frame 5 Frame 6 Frame 7 Frame 8 Frame 9 Frame 10 Frame 11 Frame 12
Tc=0
T=2
NR=[(Ts-Tc+T-1)R]/T
=[(2-0+2-1)3]/2
=9/2
=4.5
=5

Fig. 4. Parameter values for the simple scheme

recovery stream approaches to the original stream by 15 frames per second. So, the recovery stream catches up with the original stream after 2 [s] recovery duration. It is shown in Table 2.

The quantity the server eliminates the data from the recovery stream is a parameter and depends on the importance of the data. As the eliminated data increases, the recovery stream can faster catch up with the original stream. But, the quantity of the lost sensor data that the clients obtained decreases.

3.4 Calculating New Delivery Rate for Recovery Stream

The idea of our proposed scheme is shown in Fig. 3. According to the client's preferred time to catch up with the original stream, server calculates the sensor data delivery rate, *NR* for recovery streams. The delivery rate will vary according to client's desired catch-up time.

$$NR = \frac{(Ts - Tc + T - 1)R}{T} + L$$

Where

NR: new data delivery rate for recovery stream
Ts: the timestamp of the last data that the server delivers
Tc: the timestamp of the last data that each client receives
R: normal data delivery rate
T: catch-up time desired by the client from the current time
L: latency for data delivery

The situation of *NR* calculation is shown in Fig. 4. Suppose the case that the server delivers the frames with data rate of 3 [fps]. The current server timestamp is 2 (*Ts*=2). The client encounters network disconnection at the beginning of the original stream. Therefore, there are no received frames (*Tc*=0). The client wants to catch up with the original stream after 2 [s] recovery duration (*T*=2). By using the *NR* formula, new frame rate is 4.5 [fps]. However, sensor data delivery rate (5 [fps]) is used for the recovery stream. Stream merging point begins at the Time4 that is 2 [s] from the client's recovery time. As delivery rate value is increased from 4.5 to 5, at the last time before merging point, sensor data delivery rate is reduced to (4 [fps]).

To decide whether delivery rate should be reduced or not, it is necessary to check whether the last frame number for the loss

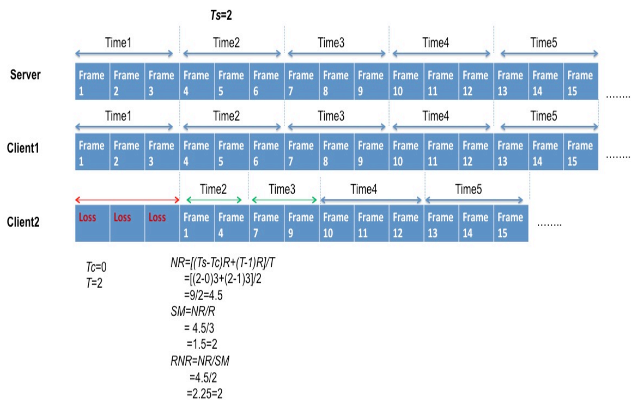

Fig. 5. Parameter values for a higher frame rate

encountered clients exceeds the server's last frame number or not.

3.5 Reduced Frame Rate and Recovery Ratio

Firstly, the amount of eliminated (skipped) frames per one frame fetching *SM* is calculated using new data delivery rate for recovery stream. Then, reduced frame rate *RNR* can be calculated using *NR* and *SM* values. The formulas for calculating *SM* and *RNR* are as follows:

$$SM = \frac{NR}{R}$$

Where

SM: eliminated frames per one frame fetching

$$RNR = \frac{NR}{SM}$$

Where

RNR: reduced frame rate

Instead of higher frame rate, *NR* for recovery streams delivery, reduced frame rate is used in our proposed scheme. Stream delivery using reduced frame rate is shown in Fig. 5. For simplicity, the latency for stream delivery from server to client (*L*) is eliminated in calculation. In comparison with Fig. 4, the server delivers lost data stream using 2 [fps] for Time2 and Time3 using skip amount value *SM*=2. Moreover, recovery ratio and distortion ratio can be calculated as follows:

$$RR = \frac{RNR}{NR}$$

$$DR = \frac{(NR - RNR)}{NR}$$

Where

RR: recovery ratio

DR: distortion ratio

An example of higher frame rate is shown in Fig. 5. In the figure, the recovery percentage will be 44% as the ratio of *RNR*=2 and *NR*=4.5 is 0.44:1. Distortion percentage will be 56% because the ratio of *NR*-*RNR*=2.5 and *NR*=4.5 gives 0.56:1.

The server can deliver the recovery streams using three different delivery rates: the faster rate (FR) which is higher than the original stream delivery rate, the same rate (SR) with the original delivery rate, and the variable rate which is calculated depending on the client's desired catch-up time and the amount of lost data that the client missed.

3.6 Merits and Demerits

Our proposed method can faster reduce the number of the streams for lost sensor data stream delivery. One of the main merits of the method is the server's load reduction. To make the stream for lost data catch up with the original stream, the server eliminates some sensor data from missing data.

One of the main demerits of the method is that the clients lack some data. But, the number of the eliminated data is a parameter for our proposed method and this is not a large problem. Clients may have different criteria for determining what is important and interesting. Depending on the clients' preferences, the same data may have different importance levels. For some clients, not all sensor data may be useful or important. For example, some clients focus on less communication traffic while others prefers to better data quality.

4. Evaluation and Performance Comparison

In this section, we explain the simulation evaluations for our proposed scheme. Total simulation time is 100 [msec] and client arrival rate is 2.

For simple scheme, the recovery streams have the delivery rate of 60. For proposed scheme, the recovery streams are delivered using three different lost sensor data delivery rates: the faster rate (FR=60 [fps]), the same rate (SR=30 [fps]) and the variable rate (VR) which is the reduced frame rate obtained using catch-up time (1 [s]).

4.1 Simulation for the Cases with Constant Rate Disconnection

First, we assume that some clients who arrive later than the original stream delivery time encounter data loss. In the simulation, 25%, 50%, 75%, or 100% of all late arrival clients become data loss encountered clients. For each case, a client arriving earlier encounters data loss earlier.

The amount of lost sensor data is proportional to the duration for the loss of the network connection. When the loss duration is 1 [s], the lost sensor data amount is 30 frames. If loss duration is 2 [s], then the lost sensor data amount is 60 frames. The loss duration is defined as the subtracted value from client arrival time to original stream delivery time. For example, the client who arrives after 2 [s] from the beginning time of the original stream will have 2 [s] loss duration. All data loss encountered clients disconnect the system at a time after starting recovery. We use the term disconnection time to refer to this time. The same value with loss duration time is assigned for disconnection time. For example, the client who arrives after 2 s from the beginning time of the original stream will disconnect the system at 2 [s] after starting the recovery stream delivery.

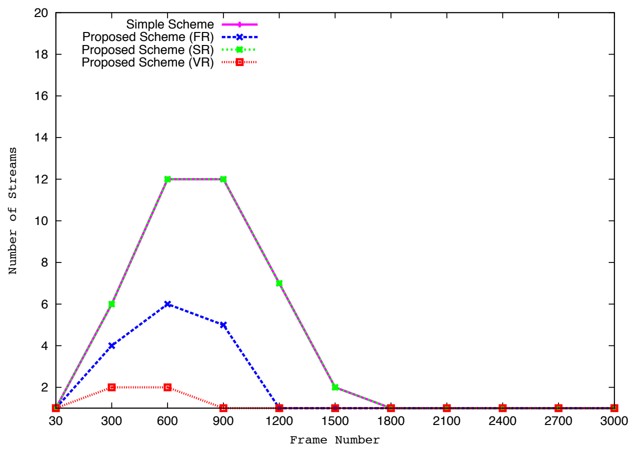

Fig. 6. The number of the streams and the frame number

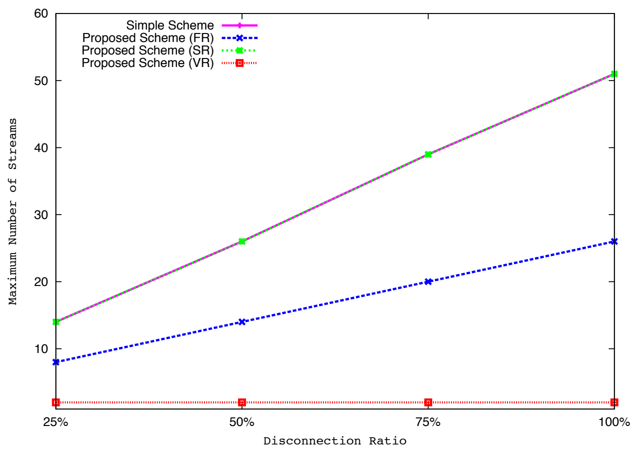

Fig. 7. The maximum number of the streams and the disconnection ratio

4.1.1 Number of Streams

Simple scheme and proposed scheme are compared in terms of the number of streams and the maximum number of streams. Fig. 6 shows the number of streams required by the server. To show the easily understandable evaluation result, we show the result only for the case that 25% of all later arrival clients encounters data losses. In the figure, the horizontal axis is the frame number. This corresponds to the time and the duration for one frame is 1/30 [s]. The vertical axis is the number of the streams that the server delivers for each time. From the figure, we can see that the number of the stream increases in the early stage of the simulation and decreases after that. This is because 25% of earlier arrival clients encounter data losses. However, the number of the streams becomes less after those clients catch up with the original stream or disconnect the system.

4.1.2 Maximum Number of Streams

The maximum number of the streams for different percentages 25%, 50%, 75%, or 100% is shown in Fig. 7. The horizontal axis is the disconnection ratio and the vertical axis is the maximum number of the streams. If there are more data loss encountered clients, the maximum number of the streams

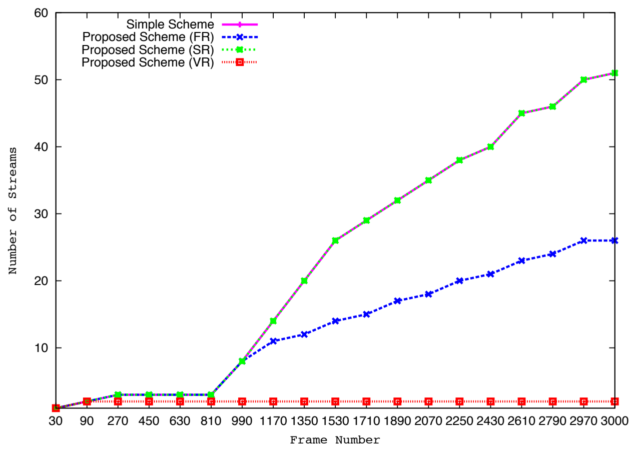

Fig. 8. The number of the streams and the frame number (disconnection time is 2 [s])

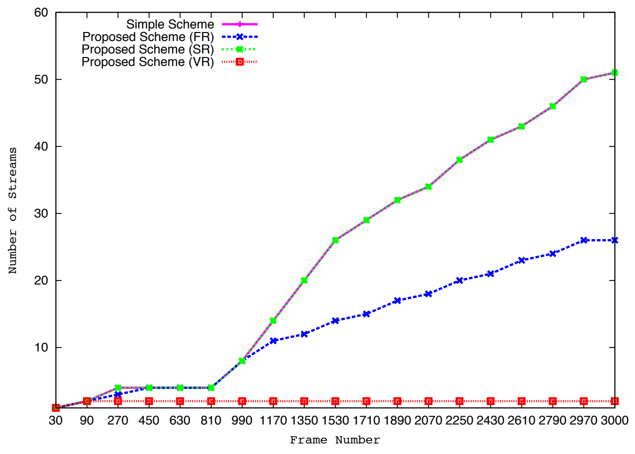

Fig. 9. The number of the streams and the frame number (disconnection time is 3 [s])

becomes larger.

Simulation results show that the number of the streams in our proposed scheme is less than that of the simple scheme. Our proposed scheme reduces the maximum number of the streams by providing faster stream merging.

4.2 Simulation for the Cases with Constant Time Disconnection

Next, we assume all clients who arrive after 1 [s] from the start time of the original stream delivery encounter data loss.

The loss duration is the subtracted value from client arrival time to the original stream delivery time. If the client arrives after 2 [s] from the beginning time of the original stream, then it will have 2 [s] loss duration.

In this simulation, some of all data loss encountered clients disconnects the system and assume 25%, 50%, 75%, or 100% of all data loss encountered clients disconnect the system. Different from the previous simulation, the disconnection time is constant in this simulation. For example, 25%, 50%, 75%, or 100% of all data loss encountered clients disconnect the system at 2 [s] after starting delivering the recovery stream.

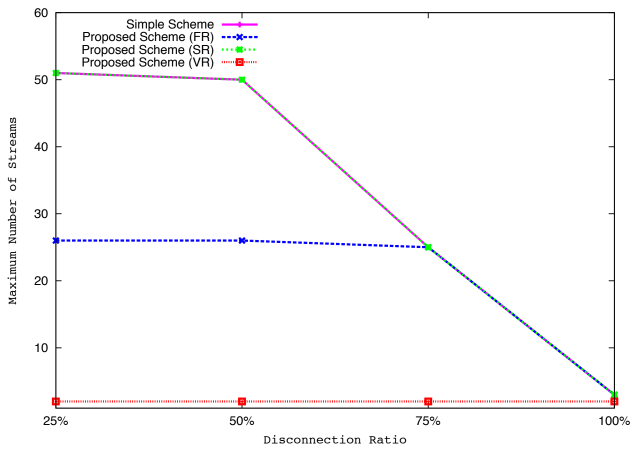

Fig. 10. The maximum number of the streams and the disconnection ratio (disconnection time 2 [s])

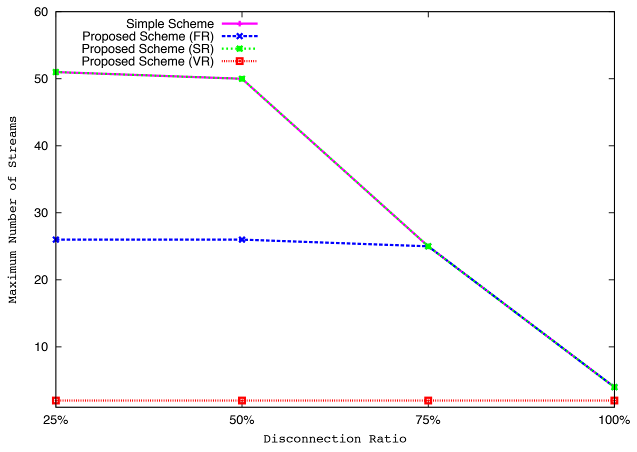

Fig. 11. The maximum number of the streams and the disconnection ratio (disconnection time 3 [s])

4.2.1 Number of Streams

The number of the streams for this case is shown in Fig. 8. The horizontal axis is the frame number and the vertical axis is the number of the streams. From the figure, we can see that the number of the streams increases as the time proceeds since the number of the clients increases. However, the increasing rate differs based on the delivery rate of simple scheme and proposed scheme. The simple scheme gives the largest number of the streams as the server delivers lost data continuously. The simple scheme and SR may have the same number of the streams although SR is lower than delivery rate of the simple scheme. This is because the server delivers some lost data but not for all in SR. FR gives the number of the streams lower than that of SR and simple scheme because the server not only delivers lost data with faster frame rate but also eliminates some data. VR gives the smallest number of the streams. This is because VR focuses on the client's desired catch-up time. If the catch-up time is short, data delivery rate will be high. If the catch-up time is long, data delivery rate will be low. In this simulation, FR has higher number of the streams because VR uses short catch-up time.

For the case when the disconnection ratio is 25% and disconnection time 3 [s], the number of the streams required by the server is shown in Fig. 9. In the simple scheme, the server delivers more streams than that of Fig. 8 since the clients' disconnection time is longer. Other features are the same reason as that for Fig. 8.

4.2.2 Maximum Number of Streams

The maximum number of the streams is shown in Fig. 11. In comparison with Fig. 10, the maximum number of streams in simple scheme and proposed scheme with SR and FR is a little bit larger than VR because the number of streams and the maximum number of the streams depends on the disconnection time and disconnection ratio.

5. Conclusion

We proposed a lost sensor data delivery scheme to reduce the server's load. In our proposed system, to faster merge the streams, the server eliminates some lost sensor data so that different recovery streams can be merged into one quickly. When the recovery stream catches up with the original stream, the number of streams becomes less. We evaluated the performance in two simulation situations in terms of the number of streams and the maximum number of streams. According to the evaluation results, our proposed scheme faster reduces the number of streams for the server.

In the future, we are planning to adopt P2P streaming technique to recovery streams delivery. And also, we will develop an actual system and evaluate our proposed scheme using this.

Acknowledgments This work is supported in part by Scientific Research (B)(15H02702) and Challenging Exploratory Research (26540045) of the Ministry of Education, Culture, Sports, Science and Technology (MEXT), Japan.

Reference

[1] B. He, M. Yang, Z. Guo, R. Chen, B. Su, W. Lin, and L. Zhou, "Comet: Batched Stream Processing for Data Intensive Distributed Computing," in Proc. of the ACM Symposium on Cloud Computing, pp. 63-74, 2010.

[2] S. Yamagiwa, M. Arai and K. Wada, "Efficient Handling of Stream Buffers in GPU Stream-based Computing Platform," in Proc. of the IEEE Pacific Rim Conference, pp. 286-291, 2011.

[3] M.J. Sax, M. Castellanos, Q. Chen, M. Hsu, "Performance optimization for distributed intra-node-parallel streaming systems," in International Conference on Data Engineering Workshops, pp. 62-69, 2013.

[4] K. A. Hua, Y. Cai, and S. Sheu, "Patching: A multicast technique for true video-on-demand services," in Proc. of the ACM International Conference on Multimedia, pp. 191-200, 1998.

[5] S. W. Carter and D. D. E. Long, "Improving Video-onDemand Server Efficiency Through Stream Tapping," in Proc. of the International Conference on Computer Communications and Networks, pp. 1095-2055, 1997.

[6] D. L. Eager and M. K. Vernon, "Dynamic Skyscraper Broadcasts for Video-on-Demand," in International Workshop on Advances in Multimedia Information Systems, pp. 18-32, 1998.

[7] L. Golubchik, J. C. S. Lui, and R. Muntz, "Reducing I/O Demand in Video-on-Demand Storage Servers," in Proc. of the ACM SIGMETRICS Joint International Conference on Measurement and modeling of computer systems, pp. 25-36, 1995.