エッジコンピューティング環境へのデータ圧縮手法の適用

渡辺 龍[1,a)]　窪田 歩[1]　栗原 淳[2]

概要：Beyond 5G での活用も考慮したモバイルネットワークのユースケースの一つとして，低消費電力かつ計算能力の低い IoT 端末を多数接続する Massive IoT 環境が挙げられる．本環境下で効率的にネットワーク資源を利用するためには，多数の IoT 端末からアップロードされるデータの圧縮・通信量の削減が必要である．本検討では，Multi-access Edge Computing (MEC) 環境を想定し，エッジノードを用いた Massive IoT 環境におけるデータ圧縮について検討する．特に，近年注目が集まりつつある，軽量かつ IoT センシングデータ等を効率的に圧縮できる，重複削除によるストリームデータ圧縮手法 “Generalized Deduplication (GD)” の適用を考慮する．基本的な GD のアルゴリズムは，1 対 1 でのストリーム送受信を前提としている．そのため本報告では，その手法を拡張して，1 対多（エッジノード対 IoT 端末）の MEC 環境に適した，効率的なアルゴリズムを提案する．具体的には，1 対多環境における GD 活用での辞書構築に関する検討と，その効率についての基礎的な評価について報告する．

キーワード：エッジコンピューティング，データ圧縮，IoT

Application of generalized deduplication techniques in edge computing environments

Ryu Watanabe[1,a)]　Ayumu Kubota[1]　Jun Kurihara[2]

Abstract: One of the use cases of mobile networks that can be considered for use in Beyond 5G is a massive IoT environment where many IoT terminals with low power consumption and computing power are connected. In order to efficiently use network resources in this environment, it is necessary to compress and reduce the amount of data uploaded by a large number of IoT terminals. In this study, we consider data compression in a Massive IoT environment using edge nodes, assuming a Multi-access Edge Computing (MEC) environment. In particular, we consider the application of ”Generalized Deduplication (GD)”, a stream data compression method based on duplicate deletion, which has been attracting attention in recent years for its lightweight and efficient compression of IoT sensing data and other data. The basic GD algorithm assumes one-to-one stream transmission and reception. In this report, we propose an extension of the GD algorithm that is suitable for one-to-multi (edge node and IoT terminals) MEC environments and has more efficient performance. Specifically, we examine dictionary construction for GD utilization in a one-to-multi environment and report a basic evaluation of the efficiency of the proposed algorithm.

Keywords: edge computing, data compression, IoT

1. はじめに

通信端末の小型化と偏在化したネットワークによる接続性の向上により，あらゆるモノがネットワークに接続し，インターネット経由で通信する IoT[1], [2] の利用がますます増加している．また，近年では，ネットワーク周辺部に配置されたサーバと周辺端末とで構築される Multi-access Edge Computing の利用も進んでいる．ネットワーク周辺部に設置されるエッジノードと呼ばれるサーバは，中央集約型のサーバに比べ非力であり，リソースの管理等が必要

[1] 株式会社 KDDI 総合研究所
KDDI Research Inc.
[2] 兵庫県立大学 大学院情報科学研究科
Graduate School of Information Science, University of Hyogo
[a)] ry-watanabe@kddi.com

である [3] が，MEC 環境を活用することで，リアルタイム性の向上が見込める．活用検討として，路肩に設置されたセンサーを活用した自動運転支援のために障害物検知の提案・実証や，複数対戦型の AR ゲームの実証実験 [4] なども報告されている．しかしながら，センサー等で利用する場合の多量の配置が必要な IoT 端末は，計算能力が低かったり，消費電力が制限されたりすることが多い．また，用途によっては，気象関連の観測データや画像データなど多種多様な情報を大量に送信する．このため，IoT 端末からアップロードされる大量のデータを圧縮し，通信量を削減することは，MEC 環境の効率的な利用に必要不可欠である．本稿では，MEC を活用する Massive IoT 環境におけるデータ圧縮について検討する．近年，この用途のために，効率的なデータ圧縮を実現する，重複削除によるストリームデータ圧縮手法である“Generalized Deduplication (GD)” は軽量かつ，IoT データの圧縮に適しており注目を浴びている．しかしながら，基本的な GD のアルゴリズムは，1 対 1 でのストリームの送受信を想定しており，先に述べた Massive IoT 環境のようなケースでの利用を想定しておらずそのまま活用することは適切でない．そこで，本 GD の手法を拡張し，1 対多となる通信環境でのデータ重複削除手法について検討する．特に，1 対多通信において，特徴的である，統合的な辞書の構築とその効率に関して基礎的な評価を記す．

2. 関連研究

センサデータなど，膨大な量のデータの送受信を行うためには，そのデータの圧縮が必須である．LZ アルゴリズム [5], [6] に基づくような通常の圧縮手法では，比較的小さいウインドウサイズを適用して圧縮する．しかしながら，センサデータなど全体に渡ってデータの出現パターンが決まっているような巨大データを圧縮するようなニーズが高まっており，通常の圧縮手法を使うよりも，「重複削除」を使うことに注目が集まっている [7].

基本的な重複削除では，データストレージやデータストリーム全体において，重複するデータチャンク (あるいはファイル) を破棄することでデータの全体量を削減する．このとき，元のデータチャンクを復元するための「辞書」を構築し，削減したデータと保持する．通常の重複削除では，全く同一のデータチャンクに対してのみ破棄が可能である．このため，ログデータなど，データの一部のみ異なる，すなわち類似するデータチャンク同士には，単純に適用することはできない課題がある．Generalized Deduplication (GD) [8], [9], [10] は，この課題を解決するために提案された，重複削除アルゴリズムのデータチャンクの類似性に関する一般化とみなせる．GD では，個々のデータチャンクを，(基底 (base), 偏差 (deviation)) と呼ばれるデータタプルへ，全単射の写像を用いて変換する．そして，基底全体の集合に対してのみ重複削除を実行し，偏差はそのまま保存あるいは送信する．このとき，適用するデータ全体のパターンに応じて写像を適切に選べば，類似性の高いデータチャンクは同一の基底を持つため，高いデータ削減効率が達成できる．

GD の応用例として, In Network Computing による DNS データストリームの削減手法 [11] では，高い削減効率と高速な処理を両立している．Hadi らは，GD を拡張し，一般的な圧縮手法と組み合わせて利用することで，圧縮効率を高める手法を提案している [12]．また，データの保護やプライバシーを考慮し，基底のみを送信する手法や，暗号化後データの重複を削除する手法も提案されている [13], [14]. GD の実装例としてコードサンプル [15] を利用することもできる．一方，これらの手法は，1 対 1 のデータストリームの送受信もしくは，単一のストレージ上の巨大データに適用することが前提とされており，1 対多形式のデータストリームへの適用は今までのところ検討されていない．

3. 提案手法

提案手法の方式を説明するにあたり，まず，適用先として想定している MEC 環境について説明し，また，一般的な GD の辞書作成の手法について記す．

3.1 Multi-access Edge Computing

想定する MEC 環境の概要を図 1 に示す．MEC 環境は，エッジノードと IoT デバイスから構成される．IoT デバイスとしては，センサー等を想定し，5G や Wi-Fi などのワイヤレスで接続されていることを想定している．IoT デバイスからは，センサーデータ等がエッジノードに送信されており，この通信量を削減することが目的である．また，図 2 に示す通り，本 MEC 環境は，同様の環境が存在し，エッジノードに集約された各環境のデータは，最終的にクラウド上のサーバにデータは集約される．MEC 環境での通信量に加え，エッジノード－クラウドサーバ間の通信量，ノード上でのデータ量の削減が期待できる．

3.2 Generalized Deduplication (GD) の基礎

一般的な GD は重複データの削除を一般化したもので，GD を活用する場合，まずストリームデータのチャンクに，ある変換関数を適用し，根源となる基底 (base) と，その派生となる偏差 (deviation) の二つの異なる値に分割する．以降，送信するストリームに対して，以前送信した値について，基底と同じ部分を重複として排除[*1]する．また，偏差については，データ復元のための反転処理ができるように送信する．

[*1] 実際には，排除と同時に辞書に基づいた置き換えを行う．

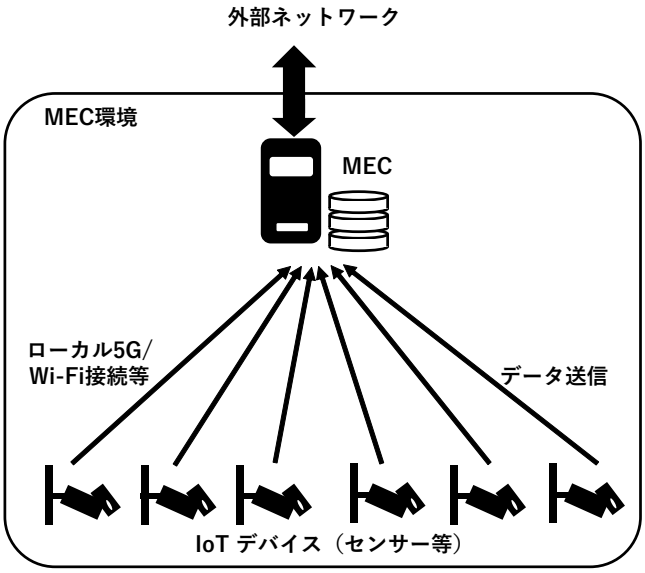

図 1 Multi-access Edge Computing ユースケース
Fig. 1 Use case of MEC environment.

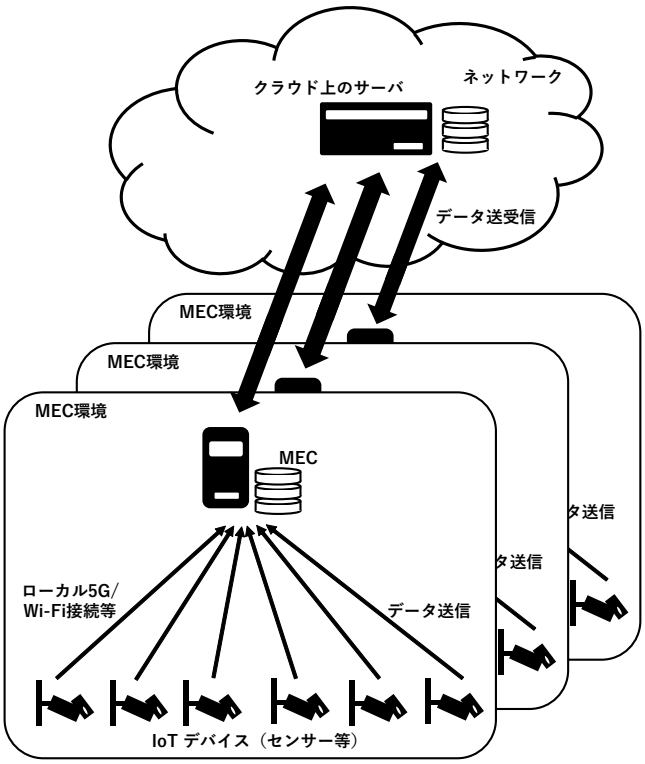

図 2 MEC とクラウドサーバとの連携
Fig. 2 Cooperation of MEC and cloud computing.

3.2.1 GD の基本概念と動作

以下に $(7,4)$ ハミング符号を元にした GD の一例を示す [11].

例 1. 処理単位となるチャンクのサイズを 7 ビットと仮定する．送信するチャンクを $(7,4)$ ハミング符号の復号器で処理することで，4 ビットの情報ビットと，最大で 1 箇所のビット反転誤りの位置が得られる．このとき，4 ビットの情報ビットを基底，一方でビット反転の誤りの位置を偏差とする．ビット反転誤りの位置は，$1,\ldots,7$ で表されるため，誤りなしの場合を含めて 8 通りとなる．従って，偏差は 3 ビットで表される．たとえば，以下の 8 つのチャンクは，全て $(7,4)$ ハミング符号の符号語 $0000000 \in \{0,1\}^7$ に対して最大 1 ビットの誤りを含んだものと見做され，基底 $0000 \in \{0,1\}^4$ にマッピングされる．

$$0000000, 0000001, 0000010, 0000100,$$
$$0001000, 0010000, 0100000, 1000000 \in \{0,1\}^7,$$

同様に，

$$1111111, 1111110, 1111101, 1111011,$$
$$1110111, 1101111, 1011111, 0111111 \in \{0,1\}^7,$$

は基底 $1111 \in \{0,1\}^4$ にマッピングされる．送信時に前から順にチャンクを処理して得られた基底に基づいて「辞書[*2]」を生成する．GD では，この辞書を参照することで，同一の基底を持つチャンクについては，基底を送信せずに辞書のインデックスを送信する．前述の 7 ビットのチャンク群の例で示す．42 ビットのデータ列

$$0000000\ 1111111\ 0100000\ 1111011\ 1000000\ 1011111,$$

は，6 つのチャンクで構成される．まず，全てのチャンクを独立に (基底, 偏差[*3]) にマッピングすると，

$$0000000 \to (0000, 000),\ 1111111 \to (1111, 000),$$
$$0100000 \to (0000, 111),\ 1111011 \to (1111, 100),$$
$$1000000 \to (0000, 101),\ 1011111 \to (1111, 110),$$

が得られる．すなわち，全てのチャンクは 2 種の基底 $0000, 1111$ を持つ．このため，このデータにおいては，基底の辞書を構築し，その中で各基底を識別するために必要なインデックスのビット数は 1 ビットで十分となる．送信時にチャンクを順に処理する場合，辞書にない基底はそのまま送信し，再度出現した際には辞書中のインデックスを送信する．このことにより，上記の 42 ビットのデータ列は，以下のように 30 ビットへ圧縮され，送信される[*4]．

$$0000|000|1111|000|0|111|1|100|0|101|1|110$$

受信側は送信側と同期を取り，受信順にデータを処理することで，送信側と同様に辞書を構築でき，送信されたデータと辞書から，元のチャンクデータを復元することが可能となる．特定の基底が頻出するようなデータに基づいて構築された辞書は，数多くのチャンクを表現することができ，より効率的な圧縮が可能となる． □

[*2] 辞書もしくはレジストリと呼ぶ.
[*3] 誤り位置．0 の場合は誤りなし.
[*4] 実際には，基底データもしくはインデックスデータのいずれかを明示するセパレータビットが必要となる.

このような仕組みのため，辞書として基底部分をどれだけ効率よく集約できるか，同じ基底を持つチャンクが出現頻度がどの程度であるかに，圧縮の効率は左右される．ファイルや，ストレージ上のデータなど，静的なデータを圧縮する場合は，一般的な圧縮手法同様にデータ構造を把握した上で辞書構築を行えるが，IoT データのように，ストリームである場合には，動的に辞書構築を行うこととなる．以降，辞書構築の説明を簡単にするために，エッジノードに向けて IoT デバイスから送信されるデータを，$A, B, C \cdots$ と表現し，A は，基底 a にマッピングされるチャンクデータを表現する．すなわち，GD によるデータ処理で，基底 a と，偏差 dev_n で表現できるデータ列は，全て A と表す．また，辞書に含まれる基底のインデックスデータは $index_*$ と表すこととする．

3.2.2 辞書生成

送信側が，A, A, B, B なるチャンクデータを送信する場合 (図 3)，辞書は次のように生成される．

(1) 送信側は，チャンク A を送信する際に，基底 a と偏差 dev_n に変換し，基底 a を $index_1$ として，辞書に登録し，$a|dev_n$ を送信する (図 4-(1))．

(2) 受信側は，初めて受信した a を，$index_1$ として辞書に登録し，受信したデータを $index_1|dev_n$ として，保存する (図 4-(2))．

(3) 送信側は，次のチャンク A を送信する．A は，最初のステップ同様に，基底 a と 偏差 dev_n に変換される．

(4) 送信側は，辞書を検索し，a が，$index_1$ であることから，a を，$index_1$ に置換し，$index_1|dev_n$ を送信する (図 5-(1)．

(5) 受信側は，$index_1|dev_n$ を受け取ると，そのまま $index_1|dev_n$ として保存する (図 5-(2))．

(6) 続いて，送信側は，チャンク B を送信する．この時，B は，基底 b と，偏差 dev_n に変換され，基底 b を，$index_2$ として，辞書に登録し，$b|dev_n$ を送信する (図 6-(1))．

(7) 受信側は，初めて受信した b を，$index_2$ として辞書に登録し，受信したデータを，$index_2|dev_n$ として，保存する (図 6-(2))．

次のチャンク B が送信される場合，(3) - (5) と同様の手順で処理される．このようにして，送信側と受信側とで，$index_1$: a, $index_2$: b という辞書が共有される[*5] ．保存されたデータから，$index_*|dev_n$ に対して，辞書を用いて，$*|dev_n$ とし，復号処理を行うことで，もとのチャンクを復元される．

[*5] 出現する基底の数が，インデックスのビット数で表現可能な最大数を超える可能性は存在する．その場合は，登場頻度の低いインデックスを上書きするなどの処理を行う [11]．この処理については，送受信側 (本稿の例では，IoT 端末，エッジノード，クラウド) において合意されていると仮定する．このとき，受信データを正しく参照していれば，辞書の完全な同期が可能である．したがって，簡単化のため，この処理を含めた詳細なアルゴリズムの説明は割愛する．

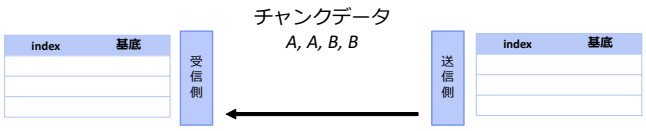

図 3 辞書作成 (初期設定)
Fig. 3 Dictionary creation (initial setting).

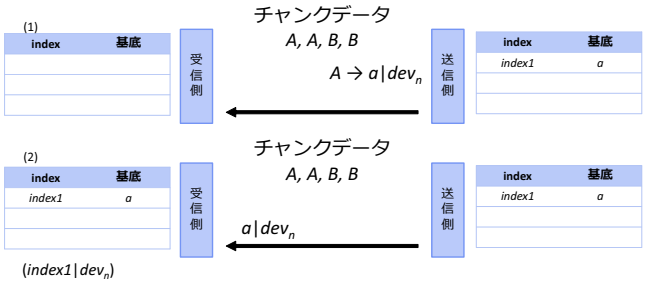

図 4 辞書作成 (index1 登録)
Fig. 4 Dictionary creation (index1 entry).

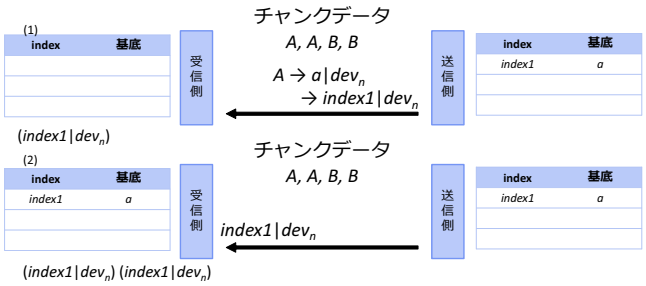

図 5 辞書作成 (index1 利用)
Fig. 5 Dictionary creation (index1 usage).

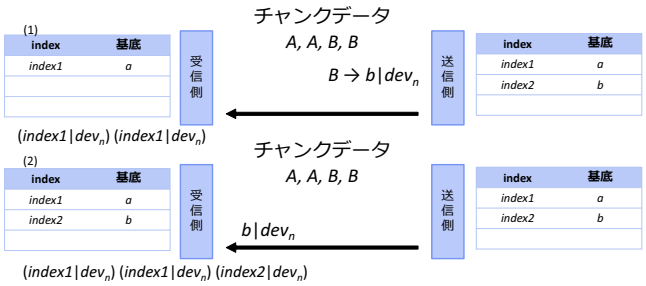

図 6 辞書作成 (index2 登録)
Fig. 6 Dictionary creation (index2 entry).

3.3 提案手法での辞書構築

想定する MEC 環境でのデータ送信に適用するために，前節で示した辞書作成手法を拡張する．送信側に変更はなく，受信側の辞書を次の様に拡張する．

- 受信側は，相対する送信側との辞書（個別辞書）の他に，統合化された辞書（統一辞書）を作成する．
- 統一辞書は，インデックスと基底との関係を保持する．

この統一辞書がメインの辞書となる[*6].

- 個別辞書は，インデックスと基底の関係を保持せず，個別のインデックスと統合化された辞書でのインデックスとの関係を保持する[*7].

また，受信側での辞書作成の際に，前節で説明した手順に加えて次の処理を追加する．

(1) 新たな基底を受け取った際に，統一辞書に同じ基底を示すインデックスがあるか否か確認する．

(2) ある場合は，送信側に対応する個別辞書に個別辞書でのインデックスと統一辞書のインデックスとの関係を記す．

(3) ない場合は，統合辞書にその基底のインデックスを登録し，対応する個別辞書に統一辞書でのインデックスとその個別辞書でのインデックスとの関係を記す．

辞書がどのように作成されるかを，送信側が 2 つある場合 (図 7) を例に説明する．2 つ送信側をそれぞれを，α, β とする．送信側 α からは，チャンクデータ A, A, B, B が，送信側 β からは，チャンクデータ C, C が送信される．説明のため，β からの送信されたデータの受信は，α から送信されたデータの到着後としているが，到着時間の前後は実際の動作に影響はない．

(1) 送信側 α から，チャンクデータ A, A, B, B が基底と偏差（あるいは，インデックスと偏差）に変換されて送信され，手順に従って，統一辞書に，$index(u)_1 : a$, $index(u)_2 : b$ が登録される．同時に，α との個別辞書に，$index_1 : index(u)_1, index_2 : index(u)_2$ が登録される．

(2) その後，送信側 β から，チャンク C が基底と偏差に変換されて送信されると，受信側は，手順に従い，統一辞書を確認し，インデックスがないことから，統一辞書に，$index(u)_3 : c$ を，登録し，β との個別辞書に，$index_1 : index(u)_3$ を記録する．

統一辞書を基準として，各送信側からの到着順に，統一辞書に新たなインデックスが作成され，個別辞書も同時に作成される．

3.3.1 比較評価

提案手法の評価として，既存手法をそのまま 1 対多の環境で利用した場合との比較を表 1 に示す．統一辞書と作成のための処理が必要となるが，その分，個別辞書のサイズを小さくすることができ，基底のサイズを k ビット，インデックスのサイズを t ビット $(k > t)$ とし，IoT デバイスの数を n とした場合，辞書サイズとして，最大で，$k(n-1) - t(n+1)$ ビットの縮小が期待できる．提案手法は辞書にのみ処理を施すため送信されるデータ及びエッジ

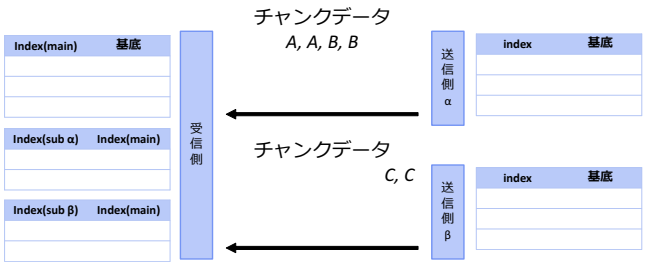

図 7 辞書作成 (初期設定)
Fig. 7 Dictionary creation (initial setting).

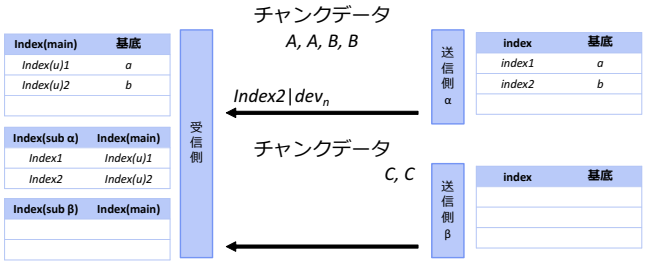

図 8 辞書作成 (α 側登録)
Fig. 8 Dictionary creation (α entry).

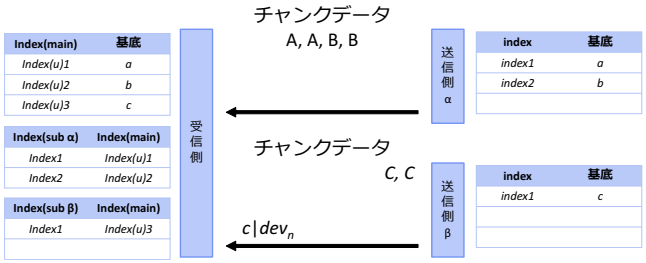

図 9 辞書作成 (β 側登録)
Fig. 9 Dictionary creation (β entry).

ノード上にストアされるデータサイズには影響しない．すなわち，データの圧縮サイズ自体に大きな変化はない．しかしながら，統合辞書と個別辞書とを活用することで，基底を一つの辞書にまとめることができるため，管理上のメリットがあると考える．

3.3.2 考察

本稿で説明した提案手法では，GD 適用のための辞書作成に着目して拡張している．しかしながら，実際の利用にあたっては，データプライバシー，データ保護などセキュリティへの配慮が必要である．例えば，送信側は基底のみを送付し，偏差については別個にまとめておき，復号時に参照するといった，データ分離を行う手法や，暗号化データの重複を削除する手法などの適用が考えられる．各手法の適用については，MEC のユースケースを想定し利用シーンごとに的確な手法を検討することが必要である．

4. おわりに

提案手法は，GD のを MEC 環境のような 1 対多の通信環境に適する際に，辞書サイズの低減の他，管理上メリッ

*6 説明の図中では，(main) と表現する．
*7 個別辞書は本来の辞書とは異なりインデックスとインデックスの関係を保持するため補助的な役割を果たす辞書である．説明の図中では，(sub) と表現する．

表 1 比較評価
Table 1 Evaluation.

項目	提案手法	既存手法	備考
統一辞書	要 (1)	不要	既存手法では個別に圧縮を行うため不要．提案手法のみ一つ必要．
個別辞書	小サイズ	通常サイズ	個別辞書は，どちらの手法を利用しても送信端末の数だけ必要であるが，既存手法の場合は，インデックスと基底の組が登録されるのに対し，インデックスとインデックスの組が登録されるため，サイズをその分小さくできる．
データサイズ	GD による圧縮	GD による圧縮	エッジノード上で保持されるデータサイズとしては，双方ともに同様の圧縮を受けるので，基本的には同サイズとなる．

トが期待できる．本検討では，辞書作成にフォーカスしたためデータ漏洩や，データプライバシーの保護等への配慮がない．今後は，安全に利用できるように，セキュリティ機能の追加に関する検討進める予定である．

謝辞 本研究は JSPS 科研費 JP22K11994, JP21H03442, JP20K23329 の助成を受けたものです．

参考文献

[1] IEEE: Towards a definition of the Internet of Things (IoT), https://iot.ieee.org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Issue1_14MAY15.pdf (2015).

[2] Atzori, L., Iera, A. and Morabito, G.: The Internet of Things: A survey, *Computer Networks*, Vol. 54, No. 15, pp. 2787–2805 (online), DOI: https://doi.org/10.1016/j.comnet.2010.05.010 (2010).

[3] 渡辺 龍, 窪田 歩, 栗原 淳: エッジコンピューティングにおけるリソース認可のパターンについて，信学技報, Vol. 120, No. 414, 電子情報通信学会，pp. 85–90 (2021).

[4] Deutsche Telekom: Deutsche Telekom, Niantic Inc., MobiledgeX and Samsung Showcase World's First Mobile Edge Mixed Reality Multi-Gamer Experience, https://www.telekom.com/en/media/media-information/archive/worlds-first-mobile-edge-mixed-reality-multi-gamer-experience-564004 (2019).

[5] Ziv, J. and Lempel, A.: A Universal Algorithm for Sequential Data Compression, *IEEE Trans. Inf. Theory*, Vol. 23, No. 3, pp. 337–343 (1977).

[6] Ziv, J. and Lempel, A.: Compression of individual sequences via variable-rate coding, *IEEE Trans. Inf. Theory*, Vol. 24, No. 5, pp. 530–536 (1978).

[7] Xia, W., Jiang, H., Feng, D., Douglis, F., Shilane, P., Hua, Y., Fu, M., Zhang, Y. and Zhou, Y.: A Comprehensive Study of the Past, Present, and Future of Data Deduplication, *Proceedings of the IEEE*, Vol. 104, No. 9, pp. 1681–1710 (2016).

[8] Vestergaard, R., Zhang, Q. and Lucani, D. E.: Generalized Deduplication: Bounds, Convergence, and Asymptotic Properties, *2019 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6 (online), DOI: 10.1109/GLOBECOM38437.2019.9014012 (2019).

[9] Vestergaard, R., Lucani, D. E. and Zhang, Q.: Generalized Deduplication: Lossless Compression for Large Amounts of Small IoT Data, *European Wireless 2019; 25th European Wireless Conference*, pp. 1–5 (2019).

[10] Vestergaard, R., Zhang, Q. and Lucani, D. E.: Lossless compression of time series data with generalized deduplication, *Proc. IEEE GLOBECOM 2019*, pp. 1–6 (2019).

[11] Vaucher, S., Yazdani, N., Felber, P., Lucani, D. E. and Schiavoni, V.: ZipLine: In-Network Compression at Line Speed, arXiv:2101.05323 (2021).

[12] Sehat, H., Kloborg, A. L., Mørup, C., Pagnin, E. and Lucani, D. E.: Bonsai: A Generalized Look at Dual Deduplication, arXiv:2202.13925v2 (2022).

[13] Bellare, M., Keelveedhi, S. and Ristenpart, T.: Message-Locked Encryption and Secure Deduplication, *Advances in Cryptology – EUROCRYPT 2013* (Johansson, T. and Nguyen, P. Q., eds.), Berlin, Heidelberg, Springer Berlin Heidelberg, pp. 296–312 (2013).

[14] Zhou, Y., Yu, Z., Gu, L. and Feng, D.: An efficient encrypted deduplication scheme with security-enhanced proof of ownership in edge computing, *BenchCouncil Transactions on Benchmarks, Standards and Evaluations*, Vol. 2, No. 2, p. 100062 (online), DOI: https://doi.org/10.1016/j.tbench.2022.100062 (2022).

[15] Kurihara, J.: rust-gd: A Rust Implementation of Generalized Deduplication, https://github.com/junkurihara/rust-gd (2022).