パケットロスの共通度にもとづくマルチキャストとユニキャストの動的切り替えによる送信時間の短縮

五十嵐 蓮[1]　河竹 純一[2]　串田 高幸[1]

概要：一般的に複数の IoT デバイスに対してデータ送信を行う場合，PC と IoT デバイスを直接接続するシリアル通信や TCP，UDP を用いた無線通信を介して行う．本稿では，複数台の IoT デバイスへのファームウェアの送信とロスしたパケットの再送信に UDP を用いる．再送信にはマルチキャストを用いるが，送信時間の減少に反し，重複して同じパケットを受信するクライアントが増加する．課題は，送信時間を最大限に軽減すると，パケットの重複の増加することである．パケット毎に UDP ユニキャストとマルチキャストを動的に切り替え，送信時間と重複率のバランスを調整する必要がある．提案として，UDP ユニキャストとマルチキャストをどの程度で切り替えるかを示す基準値の設定方法を提案する．基準値を設定するために「共通度」と UDP ユニキャストとマルチキャストの遅延を用いる．共通度は，あるパケットが何台のクライアントで共通して失われているかを表している．実験として，サーバから 5 台の ESP32 に対してファームウェアに見立てた 750KB のテキストファイルを送る．各受信デバイスごとの受信状況を収集し，共通度を算出した．150 回の実験の結果，共通度 0，1，2，3，4，5 の割合の最大値は，それぞれ 77.6%, 74.1%, 66.9%, 50.3%, 18.3%，0.4%となった．最小値は，0.1%, 0.4%, 1.7%, 0%, 0%，0%であった．共通度 0，1，2，3，4，5 のパケット数の最頻値は，それぞれ 25 個, 320 個, 225 個, 25 個, 25 個となった．

1. はじめに

背景

IoT デバイスは，シリアル通信をはじめ，TCP/IP による無線通信を用いてデータの送受信を行っている [1]．シリアル通信を用いる事で伝送効率の高い通信が実現可能である．一方，物理配線が必要であり，スペースの圧迫や配線の敷設が必要となり，ユーザビリティの低下が発生する．また，IoT デバイスは，普段人間の手が届かない場所に設置されることもある．そういった点でも物理配線の利便性が低下する [2]．そこで，TCP/IP が提供するトランスポート層プロトコルである Transmission Control Protocol(以下: TCP) や User Datagram Protocol(以下: UDP) を用いる．

TCP や UDP を用いる事で，無線によるデータの送受信が可能になるが，このうち TCP は，基本 1 対 1 の送受信にしか対応していない．さらに TCP は，信頼性が高いものの送信以外の処理，例えば高頻度の ACK の送受信や複雑なフロー制御を行っているためデータ通信におけるオーバーヘッドが多い．特にクライアントの CPU 使用率やエンドシステムの遅延時間は UDP と比較し．TCP の方が高くなる [3]．UDP は，クライアント一台ずつに対して個別でデータの送信を行う．そのため，同じパケットをクライアントの台数分送る必要がある．一方で，マルチキャストは，同じデータを全てのクライアントに対し，1 つだけで各クライアントにデータを送信することができる．したがって，エネルギーと帯域幅を節約し，待ち時間を短縮することが可能である [4]．

また，IoT におけるセキュリティリスクが年々上昇している．なぜなら，利用される IoT デバイスの台数が増加しているためである．2022 年時点では，約 310 億個の IoT デバイスが接続されており，この数は 2025 年までに 750 億個に増加すると推定されている [5]．したがって，IoT デバイスの継続的なセキュリティ対策は必須であり，そのためにファームウェアのアップデートをする必要がある．ファームウェアとは，IoT デバイスに内蔵される制御用のソフトウェアの事である．ファームウェアのサイズは種類によって異なるが，750KB ある [6]．そのため，効率的なデータ転送を行い，迅速にファームウェアを適用する必要がある．

課題

課題は，送信時間を最大限に軽減しつつ，パケットの重

[1] 東京工科大学 コンピュータサイエンス学部コンピュータサイエンス学科
先進情報専攻〒 192–0982 東京都八王子市片倉町 1404-1
[2] 東京工科大学大学院 バイオ・情報メディア研究科コンピュータサイエンス専攻〒 192–0982 東京都八王子市片倉町 1404-1

複率の増加を抑える必要がある事である．送信時間は，最初にパケットを送信してから再送信を含めた処理を終えるまでである．重複率は，パケット総数に対して，重複して受信してしまったパケットの数を表す．

まず，マルチキャストを用いることで，送信するパケットの数が減り，送信時間が減少する．図 1 では，単純なトポロジでのマルチキャストの送信処理を示している．ユニキャストの場合は，クライアントの数だけサーバがパケットを送信する．マルチキャストの場合は，サーバが送信した単一のパケットがルータで複製され，同じマルチキャストグループに参加しているクライアントに送信される．このため，伝送効率が上昇し，時間が短縮される．

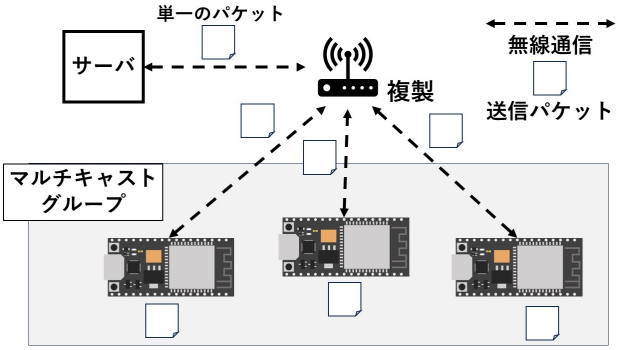

図 1　マルチキャストの送信処理

マルチキャストにより送信時間が減少する一方，一回目の送信の後，再送信を行うことによってクライアントが重複してパケットを受信してしまう．同じマルチキャストグループに属するクライアントは，そのパケットが欲しいかどうかに関らず受信してしまう．既にパケットを受信しているのにも関わらず同じパケットを受信してしまう事は，そのクライアントに重複処理を強いることになる [7]．IoT デバイスが，アプリケーションの処理をしている傍らで，パケットを受信する場合，ネットワーク処理は非同期的に実行される．これがシングルコアのマイクロコントローラの場合に問題となる．非同期処理では，重複処理の追加により増加したタスクに対する実行時間が増えてしまい，アプリケーションの処理に遅延が生じてしまう．こういったネットワーク処理が本来 IoT デバイスで実行しているアプリケーションを阻害してはならない [8]．以上のことから，より短時間でファームウェアの送信を行う上で，送信時間を最大限に軽減に伴い，重複率が増加してしまう事が課題である．

各章の概要

本テクニカルレポートは以下の形式で構成される．第 2 章では，本稿の関連研究について述べる．第 3 章では，本稿で提示した課題を解決するための提案について述べる．第 4 章では，提案手法の実装方法について具体的に述べる．第 5 章では，提案手法に対しての実験内容とその評価，分析内容について述べる．第 6 章では，より提案手法を発展させるための議論について述べる．第 7 章は，本稿のまとめである．

2. 関連研究

信頼性の高いマルチキャストトランスポートプロトコルである RMTP を提案している研究がある [9]. この RMTP では，サーバは，クライアントの集合であるドメインに対してパケットを送信する．送信されたパケットは，ドメインの中の代表である Designated Receiver(DR) によって受信される．そしてそのドメイン内でのパケットの補完は DR が行うことでトポロジ全体で効率的なパケット送信を可能にしている．この研究は再送信の際に，ユニキャストかマルチキャストかを判断する仕組みを提供している．一方，判断するための基準として，”再送信を要求するクライアントが N 台以上”と示されているのみであり，具体的な基準の設定方法には言及されていない．

データセンタにおける信頼性のあるマルチキャスト通信を提案している研究がある [10]．豊富なリンクリソースや修復スキーマである”バックアップオーバレイ”を活用している．これによりデータセンタ内のネットワークに対して，ネットワークトラフィックの節約とアプリケーションスループットの向上をもたらしている．一方で，この研究はデータセンタのみでの活用に限定しており，豊富なリソースがあることを前提としている．また，クライアントが受け取る必要のないパケットはトランスポート層でフィルタリングされる．そのため，比較的リソースに乏しくマルチキャストグループ 1 つでしか構成されていない環境では活用できない．

HLA でのデータ配布管理のためのハイブリッドマルチキャストとユニキャストの割り当て方法を提案している研究がある [11]．シミュレーション中での平均メッセージ配信時間、送信されたメッセージのコピー、受信された無関係なデータなどのパフォーマンス指標の観点から割り当てを行っている．一方で，非常に大規模であり，マルチキャストグループが 1 つである場合では活用できない．

3. 提案

本稿では，ファームウェアの送信を短時間で実行することが目的である．本提案の流れとして，何台で特定のパケットが共通して失われているかを表す「共通度」を求める．その後，本稿で提案する不等式を用いて，マルチキャストとユニキャストを動的に切り替えるための基準値を算出する．それを基に，基準値を超えた共通度を持つパケットをマルチキャストで送り，基準値以下をユニキャストで送信する．

図 2 にマルチキャストとユニキャストを判断するべき状

況を説明する．サーバは，パケットを送信し，ACK を受信する役割を担う．ルータはパケットの中継とパケットの複製を行う．クライアントである ESP32 はパケットの受信を行う．パケット 1，2，3，4 を送信しており，それぞれのクライアントがいくつかパケットをロスしている．この図からどのクライアントもパケット 4 を損失しているということが分かる．この場合，送信はマルチキャストを用いる方が効率的なのは明らかである．しかしながら，パケット 2 に関しては，2 台のクライアントのみで損失しており，これはどちらを用いて送信するべきかは定かではない．したがって，マルチキャストとユニキャストを切り替えるための基準値を設定する必要がある．

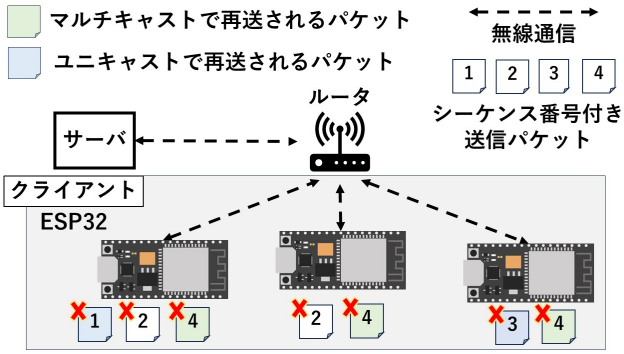

図 2　マルチキャストとユニキャスト送信を判断すべき状況

提案方式

マルチキャストとユニキャストを判断する基準値を設定するために，本稿では「共通度」を提案する．「共通度」は，何台のクライアントでパケットロスしているかを示す．図 3 は共通度を表している．

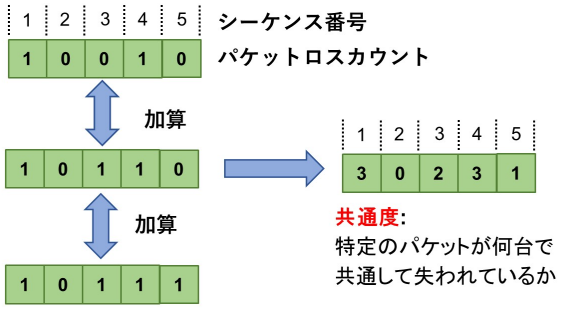

図 3　何台でパケットロスが共通してしているかを表す「共通度」

この共通度は，各クライアントから返される ACK を用いて計算される．例として三台のクライアントからそれぞれ ACK を受け取る．ACK の内容として，パケットの受信状況を表す配列が返される．この配列のインデックスは，パケットのシーケンス番号を表し，各要素は，「パケットロスカウント」を示す．パケットロスカウントは，パケットが失われたかを表しており，シーケンス番号 1 のパケットロスカウントが 1 の場合，一番目に送信したパケットがロスしたということを表している．各 ACK に含まれる配列の要素を足し合わせることによって，共通度を算出する．これを参照する事で例えば，シーケンス番号 1 のパケットは 3 台のクライアントで共通して失われているということが分かる．多くのクライアントで共通して失われているパケット，つまり共通度が高いパケットをマルチキャストで送ることでサーバが再送信するパケットの数が減り，送信時間が早くなる．しかしながら，マルチキャストで送る場合，課題で説明した通り，重複率が増加する．送信時間を優先しつつ，重複率の増加を考慮したマルチキャストの送信割合を決定する必要がある．

本提案では，ユニキャストとマルチキャストの遅延と共通度を考慮し，切り替えるための基準値を算出する．以下に計算式 (1) を提示する．マルチキャスト，ユニキャストそれぞれの遅延は，RTT を半分とすることで予測を行う．

$$D_m \geq D_u \cdot C_i \tag{1}$$

D_m : マルチキャストの遅延

D_u : ユニキャストの遅延

C_i : 共通度 i

この式が成り立つ C_i の値をユニキャストで送る最大の共通度とする．提案を適用した場合，図 4 に示す．例えば C_i の最大値が 2 場合，共通度 3 以降のパケットをマルチキャストで送信し，共通度 1 と 2 のパケットをユニキャストで送信する．

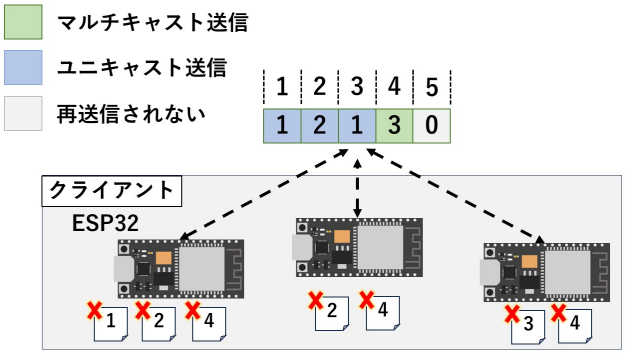

図 4　提案を適用した場合の送信処理

ユースケース・シナリオ

本提案は，複数台のクライアントに対して同じデータを送信するケースにおいて有効である．IoT のユースケースの例として図 5 のスマートホームがある．

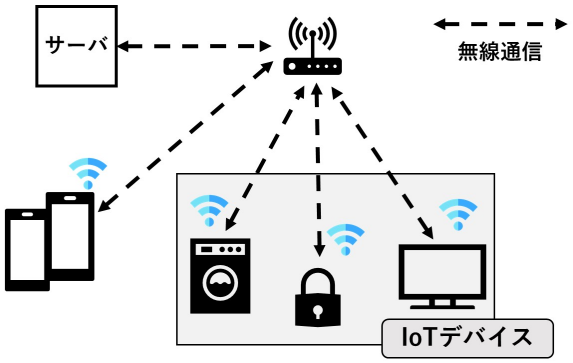

図 5　IoT を活用したスマートホーム

スマートホームで用いられる IoT デバイスは，Wi-Fi に代表される無線通信を用いて家庭のインターネット環境に接続している [12]．これらのデバイスにはマイクロコントローラが搭載されており，定期的なファームウェアのアップデートが必要である．一台ずつに対してアップデートを行うよりも本提案を適用することで迅速かつ信頼性のあるアップデートが可能とある．

4.　実装

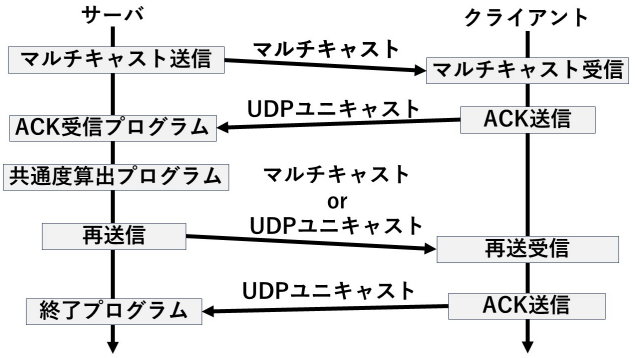

図 6　プログラム構成

図 6 で，プログラム構成を示す．サーバの OS として Ubuntu20.04 を用いた．サーバのソフトウェアは Python で実装し，クライアントは Arduino 言語 (.io) で実装を行った．IoT デバイスとして，Espressif Systems 社の ESP32 を使用した．

マルチキャスト送信 / 受信

本稿では一回目の送信時に，750 個のパケットを全てマルチキャストで送信する．クライアントは各パケットを受け取ったら，ACK としてそのパケットのシーケンス番号とパケットロスカウントを記録する．送信パケットは図 7 のフォーマットで送信される．

最大シーケンス番号は，パケットの総数を表す．シーケンス番号は，そのパケットが何番目のパケットであるかを示し，データ部は，任意のデータとなる．例として，

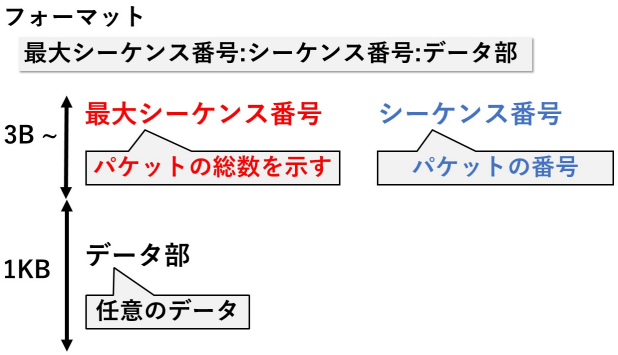

図 7　送信パケットのフォーマット

100:1:printf(“%s”, str) という形式となる．

クライアント，受け取ったパケットの総数と最大シーケンス番号を比較し，イコールであるならば，すべてのパケットを正常に受信したことを伝える ACK を出す．そうでなければ，タイムアウト後にパケットの受信状況を示す ACK を返す．

共通度算出プログラム

もしパケットロスをしたクライアントが一台でも存在した場合，共通度算出に移行する．共通度を算出した後，提案の不等式によってマルチキャストとユニキャストの切り替える基準値を決める．

再送信 / 再送受信

サーバは，共通度算出プログラムで求めた基準値を基に，パケットを再送信する．最初と同様に全ての再送パケットを送る．

ACK 送信 / 終了プログラム

再送パケットを受信したら，クライアントは，ACK を返答する．サーバは，ACK を受け取り，クライアントの受信を確認でき次第，プログラムを終了する．

パケットサイズ

本稿では，パケットサイズを 1KB とする．イーサネットの Maximum Transmission Unit(MTU) は 1500 バイトである．送信するデータグラムの全長を L バイトとし，MTU を M バイト，ネットワークパケットの損失率を Pb とすると，データグラムが完全に受信される確率は，$P_{bk} = (1 - P_b)^{(\frac{L}{M})}$ で表される [13]．データグラムの全長が MTU を超える場合，IP 層で自動的にフラグメンテーションが行われる．フラグメンテーションが行われると，帯域の圧迫やパケットの再構成により遅延が発生し，パケットロスの確率が高まる [14]．また，ネットワークを有効に活用するためには，送信するパケットのヘッダよりもペイロードに使用容量を割くべきである．したがって，データの信頼性

とネットワークの利用効率を考慮し，データグラムのサイズを 1KB とする．

5. 実験

送信側である PC から受信側である ESP32 にファイルを送信を行った．ファイルサイズは，750KB とする．実験は 150 回行い，共通度がどの程度出現するかを実験した．

実験環境

図 8 に実験環境を示す，5 台のサーバから 5 台のクライアントである ESP32 へ 750KB のファイルを送信する．ファイルを一つ 1KB のパケットに分割し，マルチキャストを用いて各クライアントに同時に送信を行う．送信時にはタイマーベースのフロー制御を行う．パケットを一定数を送るたびに sleep を挟み，クライアントのバッファを十分に確保してから送信を行うことで，パケットロスを減少させている．サーバがファイルを送信しきった時点で，クライアントはサーバに対して ACK を送信し，自身の受信状況をサーバに通知する，

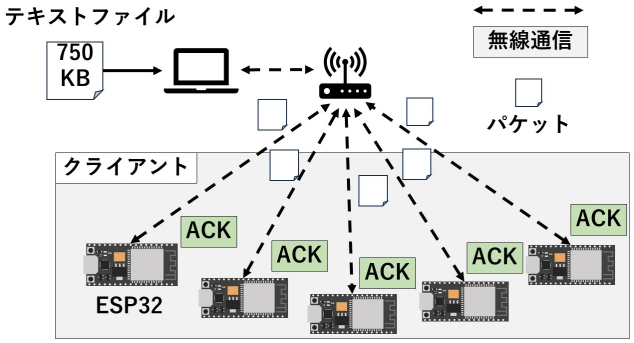

図 8 実験環境

実験結果と分析

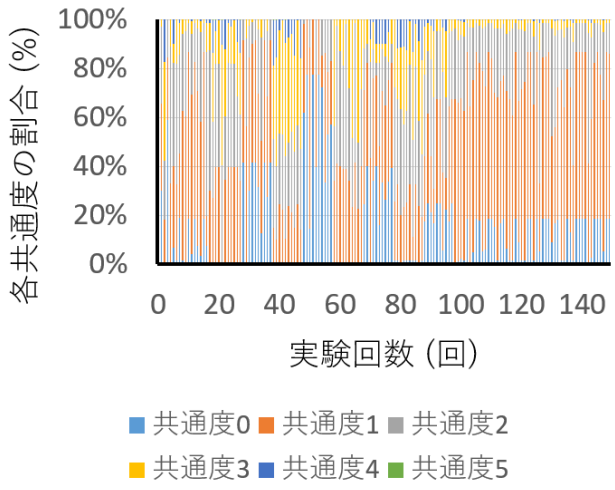

図 9 各共通度の割合

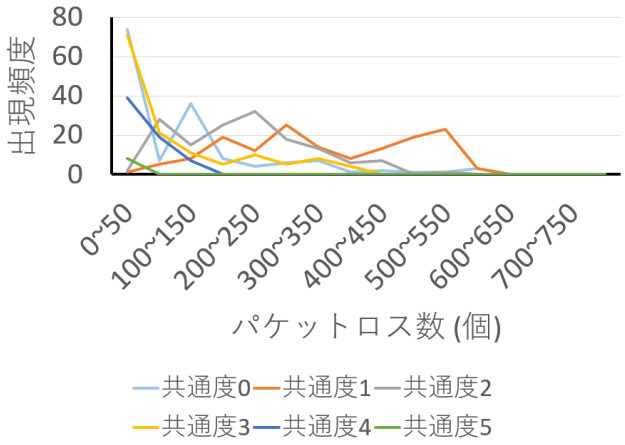

図 10 各共通度におけるパケットロス数

実験では，共通度を抽出する．共通度を算出するために 150 回計測を行った．本実験では受信デバイスが 5 台であるため，共通度は最大で 5 である．パケットの総数は 750 個である．図 9 より，共通度 0，1，2，3，4，5 の割合の最大値は，それぞれ 77.6%, 74.1%, 66.9%, 50.3%, 18.3%, 0.4%となった．最小値は，0.1%, 0.4%, 1.7%, 0%, 0%，0%であった．実験結果は，測定の度に共通度の割合が大きく変動する結果となった．図 10 は，パケットロスの数を度数折れ線にしたものである．共通度 0，1，2，3，4，5 のパケット数の最頻値は，それぞれ 25 個, 320 個, 225 個, 25 個, 25 個となった．共通度 5 は，上流のリンクで失われている可能性がある．共通度 1 と 2 にパケットロスが偏っているため，特定の機器だけパケットロスを起こしていることが分かる．本実験では，ホモジーニアスな構成だが機器ごとでパケットロスの特徴が異なるため，重ねて実験が必要である．設定される基準値に応じて，送信時間と重複率に大きく影響が出るため，動的な基準値の決定が必要である．

6. 議論

送信時には，タイマーベースのフロー制御を行っている．しかしながら，タイマーベースの制御では，タイムアウトの時間まで待機し続ける無駄が発生してしまう [15]．そのため，ACK ベースのフロー制御に改良する必要がある．ACK ベースの制御とは，パケットを送信する度に ACK という肯定応答を受け取り，相手が正常に受信したことを把握することによって安定した送信を実現することである．ACK は TCP で一般的に用いられているが，本稿ではこの ACK の受信プロセスを送信時のオーバヘッドと捉えているため，ACK の使用は最小限にしなくてはならない．したがって，バッチ ACK を用いる．複数のパケット群を一単位とし，それに対して ACK を返答する．サーバは ACK を受け取り次第，次のパケット群を送信する．これにより，サーバとクライアント間の確認応答のオーバーヘッドを最小限にする．提案手法に関して，本稿では遅延を用い

て最適な送信処理を切り替えるための基準値を決定するというものだった．今後は，より柔軟に判断を行うため，より多くのパラメータを含む基準値を定める必要がある．例えば，送信時間に関して，以下の式が成り立つ．

$$\mathbb{T} = T_b + D_m * P * m + D_u * P * (1 - m) * Ci \quad (2)$$

変数	意味
T	送信時間
T_b	再送直前までの経過時間
D_m	マルチキャスト遅延
D_u	ユニキャスト遅延
P	全体のパケットロス数
m	マルチキャストの送信割合
C_i	共通度

ユニキャストとマルチキャストの RTT や共通度，パケットロス率の情報を総合的に考慮することにより，送信時間が予測できる．より柔軟な基準値の決定が行えるよう，多くのパラメータを含む式を提案する必要がある．

7. おわりに

課題は，減少する送信時間に反して重複率が増加する事である．本稿での提案は再送時に送信時間を最大限削減しつつ，重複率を抑制することができるマルチキャストとユニキャストの送信割合を決定する事である．そのため，UDP ユニキャストとマルチキャストを動的に切り替えるための基準値を求める方法を提示した．基準値の設定には，共通度とマルチキャスト，ユニキャストそれぞれの遅延を使用する計算式を用いる．基礎実験では，実際のデータ送信においてどの程度，共通度の割合が出現するかを計測した．実験の結果，共通度 0，1，2，3，4，5 の割合の最大値は，それぞれ 77.6%, 74.1%, 66.9%, 50.3%, 18.3%, 0.4%となった．最小値は，0.1%, 0.4%, 1.7%, 0%, 0%，0%であった．共通度 0，1，2，3，4，5 のパケット数の最頻値は，それぞれ 25 個, 320 個, 225 個, 25 個, 25 個となった．

参考文献

[1] Wukkadada, B., Wankhede, K., Nambiar, R. and Nair, A.: Comparison with HTTP and MQTT in Internet of Things (IoT), *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*, IEEE, pp. 249–253 (2018).

[2] Sharma, N., Mangla, M., Mohanty, S. N., Gupta, D., Tiwari, P., Shorfuzzaman, M. and Rawashdeh, M.: A smart ontology-based IoT framework for remote patient monitoring, *Biomedical Signal Processing and Control*, Vol. 68, p. 102717 (2021).

[3] Gu, Y. and Grossman, R. L.: Optimizing UDP-based protocol implementations, *Proceedings of the Third International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet 2005)*, pp. 3–4 (2005).

[4] Gupta, S. K. and Cherukuri, S.: An adaptive protocol for efficient and secure multicasting in IEEE 802.11 based wireless LANs, *2003 IEEE Wireless Communications and Networking, 2003. WCNC 2003.*, Vol. 3, IEEE, pp. 2021–2026 (2003).

[5] Schiller, E., Aidoo, A., Fuhrer, J., Stahl, J., Ziörjen, M. and Stiller, B.: Landscape of IoT security, *Computer Science Review*, Vol. 44, p. 100467 (2022).

[6] Börsig, M., Nitzsche, S., Eisele, M., Gröll, R., Becker, J. and Baumgart, I.: Fuzzing framework for ESP32 microcontrollers, *2020 IEEE International Workshop on Information Forensics and Security (WIFS)*, IEEE, pp. 1–6 (2020).

[7] Ahn, J. H., Kim, J. H. and Kim, T. G.: Design and implementation of data distribution management in IEEE 1516 HLA/RTI, *Proceedings of the 2007 Summer Computer Simulation Conference*, Citeseer, pp. 386–391 (2007).

[8] Neamtiu, I., Hicks, M., Stoyle, G. and Oriol, M.: Practical dynamic software updating for C, *ACM SIGPLAN Notices*, Vol. 41, No. 6, pp. 72–83 (2006).

[9] Paul, S., Sabnani, K. K., Lin, J.-H. and Bhattacharyya, S.: Reliable multicast transport protocol (RMTP), *IEEE journal on selected areas in communications*, Vol. 15, No. 3, pp. 407–421 (1997).

[10] Li, D., Xu, M., Liu, Y., Xie, X., Cui, Y., Wang, J. and Chen, G.: Reliable multicast in data center networks, *IEEE Transactions on Computers*, Vol. 63, No. 8, pp. 2011–2024 (2013).

[11] Wang, J. and Zheng, T.: A hybrid multicast–unicast assignment approach for data distribution management in HLA, *Simulation Modelling Practice and Theory*, Vol. 40, pp. 39–63 (2014).

[12] Wang, Y. and Yan, S.: A compact in-band full duplexing antenna with quasi-isotropic radiation pattern for IoT-based smart home and intravehicle wireless communication applications, *IEEE Internet of Things Journal*, Vol. 9, No. 17, pp. 16689–16700 (2022).

[13] Guo, L. and Chengtong, L.: Research and Achievement of a Way to Improve the Data Transmission Reliability of UDP, *2012 International Conference on Computer Science and Service System*, IEEE, pp. 627–630 (2012).

[14] Korhonen, J. and Wang, Y.: Effect of packet size on loss rate and delay in wireless links, *IEEE Wireless Communications and Networking Conference, 2005*, Vol. 3, IEEE, pp. 1608–1613 (2005).

[15] Hei, X., Chen, J., Lu, H., Xie, G. and Meng, H.: A UDP-based way to improve data transmission reliability, *2017 29th Chinese Control And Decision Conference (CCDC)*, IEEE, pp. 2612–2617 (2017).

