特別研究報告

題目

リアルタイムオペレーティングシステムを用いたエッジコンピューティング環境における ITS アプリケーションの遅延特性の評価

指導教員
村田 正幸 教授

報告者
八千古嶋　龍

2020 年 2 月 10 日

大阪大学 基礎工学部 情報科学科

2019 年度 特別研究報告

リアルタイムオペレーティングシステムを用いたエッジコンピューティング環境における ITS アプリケーションの遅延特性の評価

八千古嶋　龍

内容梗概

近年、車両や歩行者から局所的に得られる環境情報をデータセンターで収集・解析し、車両群へ安全性に関する情報をフィードバックすることで安全性を高める ITS（Intelligent Transport Systems）アプリケーションへの期待が高まっている。しかし、データセンターを介した情報処理には、遠隔地のデータセンターへ情報を伝達することによる通信遅延の増大や、膨大な車両環境情報がデータセンターに集中して送られることによる処理遅延の増大が懸念されている。そのため、ネットワークの末端にコンピューティングリソースを配置して活用するエッジコンピューティングを導入し、アプリケーションが享受する通信遅延を削減することが期待されている。しかし、エッジサーバーのコンピューティングリソースは限られるため、例えば見通しの悪い交差点での車両衝突の検知などの ITS アプリケーションにおいて、リアルタイム性を確保することが重要である。そこで本報告では、エッジサーバーで RTOS（Real-time Operating System）を動作させ、リアルタイム処理が求められるアプリケーションの応答遅延に対する効果を、実機を用いて明らかにする。まず、車両の衝突検知アルゴリズムを実装し、バックグラウンド負荷を高めた条件下でサーバーの処理遅延の最悪値を測定した。その結果、非 RTOS の Fair スケジューリングの場合に 14 ms であった処理遅延の最悪値が、RTOS のリアルタイムスケジューリングの場合に 0.026 ms となることがわかった。次に、研究室内に LTE 通信システムを構築し、LTE 使用した UDP 通信により車両位置情報を送り衝突検知結果を受信する衝突検知アプリケーションの応答遅延を測定した。その結果、RTOS によりアプリケーションの応答遅延の最悪値が最大で 30%削減されることが明らかとなった。

主な用語

ITS（Intelligent Transport Systems）、先進運転支援システム、エッジコンピューティング、リアルタイム処理、リアルタイムオペレーティングシステム、セルラー網

目 次

図 目 次

表 目 次

1 はじめに

交通事故による死者は世界で年間 100 万人以上に上り、大きな社会問題となっている。安全性の高い ITS を構築するため、車両間、車両とインフラ間、車両と歩行者間、いわゆる V2X（Vehicle to X）で通信を行い、自車周辺環境を的確に把握する技術が考案されている。しかし、不特定の二者間通信による位置情報等の共有はセキュリティ上の懸念から実現が困難である。そこで近年 ICT インフラを介した情報処理、情報伝達が着目されている [1]。

ICT インフラを活用した ITS では、車両自身もしくは車両周辺の情報（以降、車両環境情報）に対する情報処理にデータセンターを用いることが想定されている。これは、コンピューティングリソースが潤沢なデータセンターで車両環境情報の集約・解析を行い、車両群へ安全性に関する情報をデータセンターから車両へフィードバックするものである。具体的なユースケースとして、見通しの悪さや錯覚などの理由で人流の予測が困難となる交差点における衝突の予測や、交通流を意識した信号制御などが考えられている [1, 2]。

データセンターを介した情報処理、情報伝達には、遠隔地のデータセンターへ情報を伝達することによる通信遅延の増大の問題や、膨大な車両環境情報がデータセンターに集中して送られることによる処理遅延の増大の課題がある。そのため、エッジコンピューティングを導入することでアプリケーションが享受する遅延を削減することが期待されている [3]。エッジコンピューティングは、通信網のユーザ側（車両側）の末端にコンピューティングリソースを配置し提供することで、通信遅延を削減するコンピューティング方式である。従って、車両環境情報をデータセンターまで送らず末端に設置されるエッジサーバーに集約することが可能となり、エッジサーバーのコンピューティングリソースを用いて解析し、その結果の車両へのフィードバックを迅速に行うことが可能となる。また、データセンターでは、すべての対象地域の車両環境情報を処理しなければならないが、エッジサーバーでは地理的に近い範囲の車両環境情報のみを集約・解析するだけで良く、情報処理量の低減による処理遅延削減が期待される。その一方で、エッジサーバーのコンピューティングリソースは限られているため、エッジサーバーを利用する他のアプリケーションとのリソース競合への対処が課題となる。すなわち、リソース競合が発生する状況においても、リアルタイム性の求められるアプリケーションの情報処理に要する遅延を抑制し、アプリケーションの応答遅延を削減することが課題となる。

この課題に対し、本報告では RTOS（Real-time Operating System）に着目する。RTOS は、時間的な制約がある処理を実行するためのスケジューリング機能を備えたオペレーティングシステムであり、組み込みシステムなどのハードウェア割込を最優先で実行するシステムに導入される [4]。RTOS をエッジサーバーに導入することで、他のアプリケーションによる処理がなされている場合でも、ITS アプリケーションの情報処理に要する処理遅延が抑制

され、ITS アプリケーションの安定した応答遅延の実現が期待される。そこで本報告では、エッジサーバーで RTOS を動作させ、リアルタイム処理が求められるアプリケーションの処理遅延と応答遅延の削減効果を、実機を用いて明らかにする。なお、エッジサーバーを利用する ITS アプリケーションの検討は文献 [2] で進められているが、そのユースケースの 1 つである車両衝突検知アプリケーションを想定し、車両の衝突検知アルゴリズムを実装した。まず、RTOS が動作する計算機と非 RTOS である一般 OS（以降、GPOS： General-Purpose Operating System）が動作する計算機それぞれで衝突検知アルゴリズムを実行し、実行時間を比較することで RTOS による処理遅延削減の有効性を確認する。次に、OpenAirInterface [5] を用いた LTE 通信システムとエッジサーバーからなる実験環境を構築し、LTE 通信を行う末端端末が車両環境情報を送信する時刻と、衝突検知結果を受信する時刻の差を ITS アプリケーションの応答遅延とし、RTOS による応答遅延の削減効果を実験的に測定する。なお、文献 [6] においても OpenAirInterface による LTE 通信システムを構築し、エッジコンピューティング導入による通信遅延の削減効果を明らかにしている。ただし文献 [6] では、エッジサーバーにおいて他のアプリケーションが実行されていない場合の遅延を示しており、リアルタイム処理が求められるアプリケーションの応答遅延の削減をエッジサーバーで実現するものではない。

本報告の構成は以下の通りである。まず 2 章では、エッジコンピューティングを用いた ITS アプリケーションの動向を述べる。次に 3 章では、エッジサーバーで動作する衝突検知アルゴリズムの実行時間を測定し、RTOS による処理遅延の削減効果を明らかにする。4 章では、LTE 通信システムを用いて衝突検知アプリケーションの応答遅延の削減効果を述べる。最後に 5 章では、本報告のまとめと今後の課題を述べる。

2　エッジコンピューティングを用いたITSアプリケーション

2.1　エッジコンピューティング

エッジコンピューティングは、通信網のユーザ側の末端にコンピューティングリソースを配置し提供することで、通信遅延を削減するコンピューティング方式である。このコンピューティング方式の登場背景にあるのは、インターネットサービスが多様化し登場したリアルタイム性を要するアプリケーションである。具体例としては、実世界センシングやネットワークMRアプリケーションなどが挙げられる。これらのアプリケーションは一般的なアプリケーションとは異なり、遠隔地のデータセンターへ情報を伝送する遅延を許容することはできない。それゆえユーザに地理的に近い位置でのコンピューティングを行う必要がある。そこで考案されたのがエッジコンピューティングである。 エッジコンピューティングではセルラー網のユーザ側の末端、基地局やコアネットワーク付近に小規模なデータセンターを設けサービスを提供することで通信遅延を削減したリアルタイムサービスを実現することが出来る。

2.2　ITSアプリケーションとしての衝突検知

交通事故の大きな割合を占めるのは衝突事故である。衝突事故は主に人対車両、車両対車両、車両対物体の3種類に大別できるが、車両の運転者の視野、認知能力および判断能力には限界があるため、自動車に搭載されるコンピューティングリソースもしくはエッジサーバーのコンピューティングリソースを用いて衝突事故を予防する運転支援が望まれる。

近年は、自動車に搭載されるカメラやLidarセンサー、そしてECU（Electronic Computing Unit）によるコンピューティングリソースを用いて運転者の認知能力や判断能力を補う先進運転支援システムに関する研究開発が、国内外で広くなされている[7, 8, 9]。しかし、カメラやセンサーによって局所的に得られる情報のみを用いた運転支援技術には、死角外から突然現れる人や車両との衝突を防止できない問題がある。具体的には、見通しの悪い交差点における車両間及び車両と人との衝突や、動静の認知誤り及びコリジョンコース現象などの錯覚による車両間の衝突を防止することが困難となる。そのため車車間で情報共有し運転を支援する研究がなされている[10]。これは車両と車両の間で直接通信を行い、車両の位置、速度情報を共有し、衝突検知を行うものである。しかし、不特定の車両間での情報共有はプライバシーやセキュリティ上の懸念があることや、双方が移動する中で安定した通信を行うことが難しい。そこで、車車間でなく車両とインフラ間で通信を行い、インフラで情報の収集・解析し衝突検知及びフィードバックを行うことが検討されており、セルラー網を介して

データセンターもしくはエッジサーバーに車両環境情報を収集・解析し衝突検知及びフィードバックを行うことが想定されている [11, 2]。

セルラー網を介してインフラで車両情報を収集・解析しフィードバックを行うアプリケーションは C-V2I（Cellular Vehicle to Infrastructure）アプリケーションと呼ばれる。C-V2I アプリケーションの例としては、衝突検知以外にも非常ブレーキ警告や車両状況のリモートモニタリングなどが挙げられる [2]。非常ブレーキ警告アプリケーションでは、前方の死角位置にある車両がブレーキを掛けた場合にドライバーに通知し追突事故を防ぐものである。また、車両状況のリモートモニタリングアプリケーションは、車両に何らかの異常が発生し走行困難となった場合に周囲のドライバーに注意を促すとともに、道路管理者に対応を要請するものである。これらのアプリケーションの中でも、衝突検知は最も応答時間制約に厳しいアプリケーションであり、応答の遅れが車両同士の衝突する致命的な事故に繋がる。厳しい応答時間制約を満たすには衝突の検知処理をリアルタイムに行う必要がある。本報告では、衝突検知アプリケーションを想定し、エッジサーバーに RTOS を導入する有効性を評価する。

3　リアルタイムオペレーティングシステム導入によるエッジサーバー処理遅延の評価

ITS アプリケーションに使用するアルゴリズムを実装し、リソースが競合した状況下における RTOS と GPOS それぞれにおける処理遅延を計測し比較する。この比較結果から RTOS により得られる処理遅延の削減効果を明らかにする。

3.1　リアルタイムオペレーティングシステム

リアルタイムオペレーティングシステム（RTOS）とは、リアルタイム処理に必要な高い応答性能やスケジューラーを併せ持った OS である。非 RTOS のオペレーティングシステムの応答性能が数ミリ秒オーダーであるのに対し、RTOS の応答性能は数マイクロ秒オーダーである。また、RTOS ではリアルタイム性を保証するため、優先度及び時間制約を考慮したスケジューリングが行われる。リアルタイム性とは所定の時間内に処理を完了できること、すなわち、処理完了時間が予測可能であることを意味する。したがって、処理要求が発生した時点で処理の開始時間と終了時間が保証されることがリアルタイム性を保証するための要件となる。

本報告で用いる RTOS は、オープンソースカーネルの Linux にリアルタイム処理拡張を施した Preempt-RT カーネルである。非 RTOS の Linux は、優先度に基づいて CPU 割当量を配分するスケジューラを持つものの、応答時間には数 ms 程度のばらつきがあり、処理完了時間を保証するリアルタイム処理の要件を満たすことが出来ない。このような応答時間のばらつきを低減するため、Preempt-RT カーネルでは既存の Linux カーネルコードを横取り可能なコードに置き換え、割り込みハンドラをスレッド化するなどのアプローチを採っている [4]。これにより、リアルタイムタスクがカーネルスレッドや割り込みハンドラからプロセッサを横取りできるようになり、実行可能状態のリアルタイムタスクはカーネルスレッド、割り込みスレッドの終了を待たずに実行できる。このようにして Preempt-RT では応答時間が小さくより決定的になり、リアルタイム性の要件である処理完了時間の保証を実現している。

3.2　衝突検知アルゴリズム

RTOS による処理遅延削減効果を測定するにあたり、ITS アプリケーションとして衝突検知アプリケーションに着目する。衝突検知アプリケーションでは、セルラー網を介した情報集約、情報更新処理を含むが、本章ではエッジサーバーで実行される衝突検知アルゴリズム

Algorithm 1 Collision detection pseudocode[11]

Require: $\vec{x_0}, \vec{v}, \mathrm{B}$

1: $\mathrm{C} \leftarrow \emptyset$
2: $\vec{x}(t) \leftarrow \vec{x_0} + \vec{v}(t)$
3: **for all** $b \in \mathrm{B}$ **do**
4: $\quad \vec{x^b}(t) \leftarrow \vec{x_0^b} + \vec{v^b} \cdot t$
5: $\quad D(t) \coloneqq |\vec{d}(t)|^2 \leftarrow (\vec{v} - \vec{v^b})(\vec{v} - \vec{v^b})t^2 + 2(\vec{x_0} - \vec{x_0^b}) \cdot (\vec{v} - \vec{v^b})t + (\vec{x_0} - \vec{x_0^b}) \cdot (\vec{x_0} - \vec{x_0^b})$
6: $\quad t^* \coloneqq t : \frac{\mathrm{d}}{\mathrm{d}t} D(t) = 0 \leftarrow \frac{-(x_0 - \vec{x_0^b}) \cdot (\vec{v} - \vec{v^b})}{|\vec{v} - \vec{v^b}|^2}$
7: $\quad$ **if** $t^* < 0$ **or** $t^* > t2c_t$ **then**
8: $\quad\quad$ **continue**
9: $\quad$ **end if**
10: $\quad d^* \leftarrow \sqrt{D(t^*)}$
11: $\quad$ **if** $d^* \leq s2c_t$ **then**
12: $\quad\quad \mathrm{C} \leftarrow \mathrm{C} \cup \{b\}$
13: $\quad$ **end if**
14: **end for**
15: **return** C

のみによる処理遅延を測定する。セルラー網を介した情報集約、情報更新処理を含む衝突検知アプリケーションの応答遅延は 4 章で詳しく述べる。

本報告では、文献 [11] で提案されている衝突検知アルゴリズムを用いる。その擬似コードを、Algorithm 1 に示す。なお、このアルゴリズムは交差点における衝突検知に限らず、任意の対物間における衝突を検知するものである。

このアルゴリズムに対する入力は、注目する物体の情報 (以降 *ego*) と衝突する可能性のある物体情報の集合 B である。B の要素全てに対し *ego* との衝突判定を行い、衝突すると判定された車両は衝突する物体のリスト C に加えられる。2 物体間の衝突判定（line 4-11）において、最初に求めるのは最接近時刻 t^* である（line 4-6）。t^* は時刻 t における距離の式 $D(t)$ に極値を与える t である。このアルゴリズムでは誤検知を減らすため、差し迫った衝突にのみ警告を発するよう設計されている（line 7-8）。これを反映して、最接近時刻 t^* が閾値よりも大きい場合にはその時点でその車両との衝突可能性判定を終了する。また、t^* が負の場合には以降 2 車間の距離が離れていくので、同様に判定を終える。最接近時刻が差し迫っている場合には、続いて最接近距離 d^* を算出する（line 9）。さらにこの最接近距離が危険な距離つまり、閾値 $s2c_t$ 以下の場合にはこの物体を衝突物体リスト C に加える（line 10-12）。

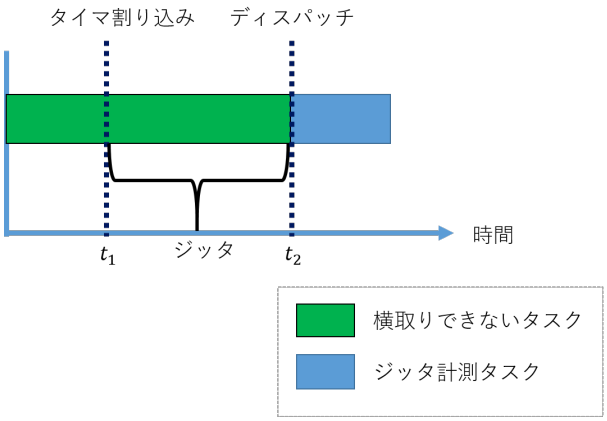

図 1: Cyclitest によるジッタ計測

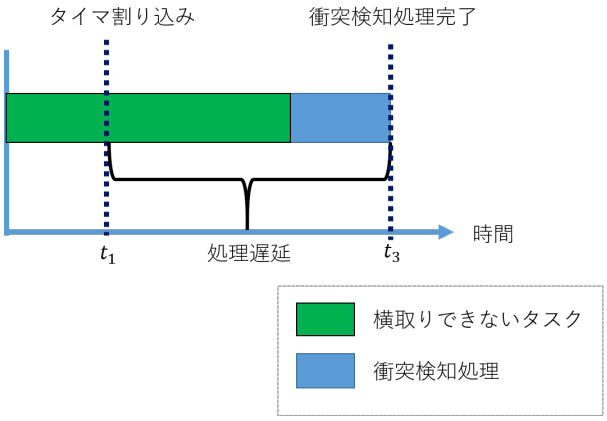

図 2: 衝突検知アルゴリズムの処理遅延の計測

3.3 処理遅延の測定方法

衝突検知アルゴリズムの処理遅延の計測するにあたり、リアルタイム性のテストツール Cyclictest[12] を拡張する。

Cyclictest はスレッドのウェイクアップからディスパッチまでの時間すなわちジッタを計測するツールである。ジッタはカーネルタスクや割り込みハンドラなど CPU を横取りできない処理の完了を待つために生じる。Cyclictest では図 1 のように、ジッタ計測スレッドが指定した時刻 t_1 にタイマ割り込みが発生させ、ディスパッチされるとすぐさま時刻 t_2 を取得する。この時間差 $t_2 - t_1$ が求めるジッタとなる。

本報告では、Cyclictest を拡張し、タイマ割り込みを処理要求の発生とし、タイマ割り込みの発生から衝突検知アルゴリズムの実行処理終了までの時間を計測している。具体的には、図 2 のように計測タスクの処理の始めに衝突検知処理を行い、完了した時点で時刻 t_3 を取得し処理遅延 $t_3 - t_1$ を計測対象とするように拡張した。

表 1: エッジコンピューティング環境の諸元

使用 OS	Ubuntu 18.04 LTS
使用カーネル	GPOS: linux-5.2.17-general
	RTOS: linux-5.2.17-rt9（Preempt-RT）
CPU	Intel Xeon E-2174G（4 コア、3.8 GHz）

表 2: 車両台数毎の平均処理遅延（単位 : ms）

車両数 \ ポリシー	GPOS_Fair	GPOS_Real	RTOS_Real
100	0.103	0.018	0.007
1,000	0.142	0.060	0.046
10,000	0.656	0.546	0.482
100,000	15.491	7.889	7.786

なお、この計測を 1) GPOS で標準スケジューリングポリシー、2) GPOS のリアルタイムスケジューリングポリシー、3) RTOS のリアルタイムスケジューリングポリシー、それぞれを用いた場合で行う。ここで標準スケジューリングポリシーとは Linux の Fair クラスのスケジューリングポリシー、sched_other であり、リアルタイムスケジューリングポリシーとは Real-time クラスのスケジューリングポリシー sched_fifo である。また、本報告でリアルタイムスケジューリングポリシーを用いる際にはリアルタイム優先度 80 を付与している。1) と 2) の結果を比較することでリアルタイムスケジューリングポリシーの処理完了時間短縮効果を、2) と 3) の結果を比較することで RTOS による応答時間の削減効果を、それぞれ評価する。評価指標には、処理遅延の平均値と最悪値を用いる。

また実際の運用においては他のアプリケーションを同一のエッジサーバーで運用することが想定され、リソースが競合した状況下においても、衝突検知処理にかかる遅延を抑え時間制約を満たすことが求められる。この状況を再現するために計測の際にはバックグラウンド負荷を掛ける。バックグラウンド負荷をかける手段として stress コマンドを用いた。バックグラウンド負荷の内容は CPU 負荷、メモリ入出力負荷、メモリ割当・解放負荷、ディスク入出力負荷の計 4 種類である [13]。

3.4 評価結果

測定に用いたエッジコンピューティング環境の諸元を表 1 に示す。なお、エッジコンピューティング環境は仮想化環境を用いて構築することも想定されているが、仮想化環境で生じる

表 3: 車両台数毎の最悪時処理遅延（単位 : ms）

車両数 \ ポリシー	GPOS_Fair	GPOS_Real	RTOS_Real
100	14.056	1.169	0.026
1,000	14.056	1.284	0.159
10,000	18.491	1.752	1.303
100,000	86.453	72.158	18.056

表 4: 車両台数毎の処理遅延の 99 パーセンタイル値（単位 : ms）

車両数 \ ポリシー	GPOS_Fair	GPOS_Real	RTOS_Real
100	0.892	0.349	0.010
1,000	0.905	0.408	0.054
10,000	2.298	0.947	0.549
100,000	36.668	8.578	8.277

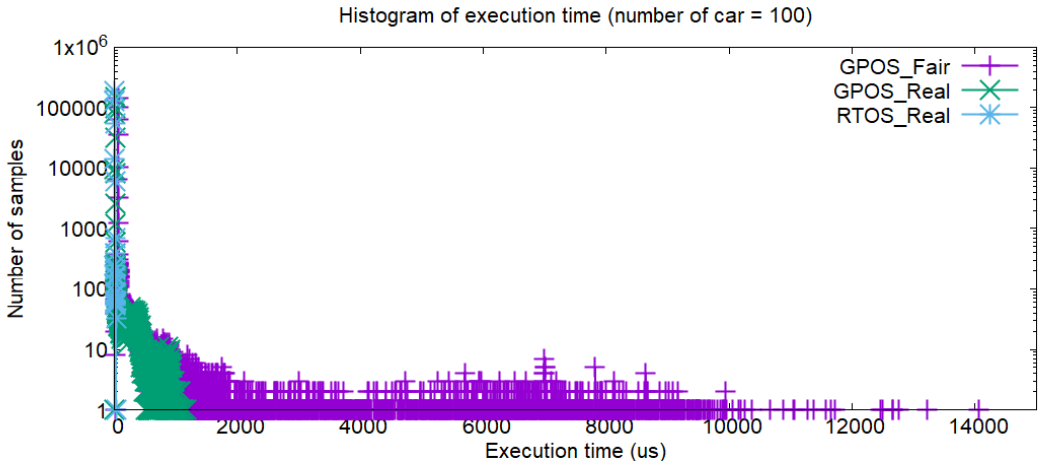

図 3: 処理遅延の度数分布：車両台数 100 台

要因を排除するため本報告では仮想化環境は用いないこととした。また、測定にあたっては CPU 周波数を動的に変更する機能やハイパースレッディング機能を無効化している。

なお計測する際パラメータの車両台数を 100、1,000、10,000、100,000 と変更して計測を行った。車両台数のパラメータを n は生成される車両環境情報の数を表しており、計測の際はその内 1 つの車両と残り $n-1$ 台との衝突検知処理を行う。それぞれの車両台数に対して 4 つの CPU でそれぞれ 100,000 回ずつ反復計測し計 400,000 のサンプルデータを得る。反復は 1 回の計測が終了して 2 ms 後に行われる。各車両台数における処理遅延の度数分布を図

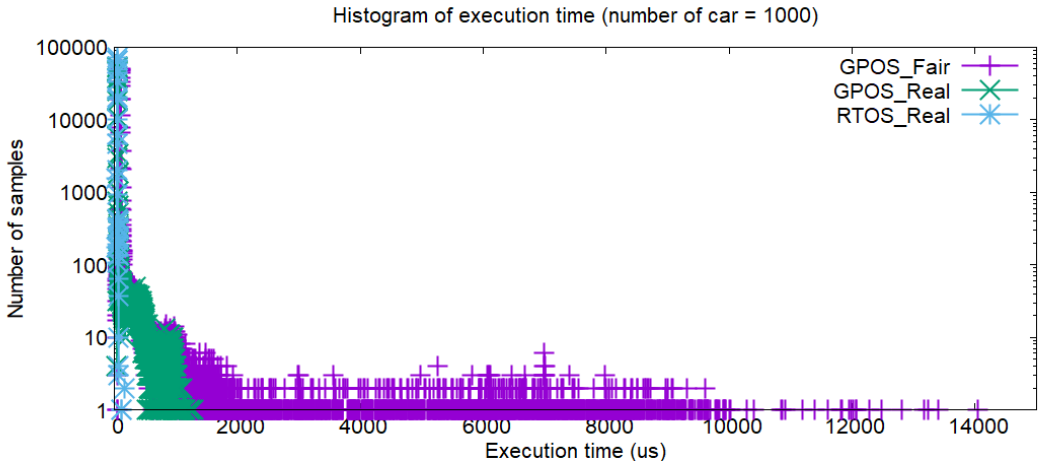

図 4: 処理遅延の度数分布：車両台数 1,000 台

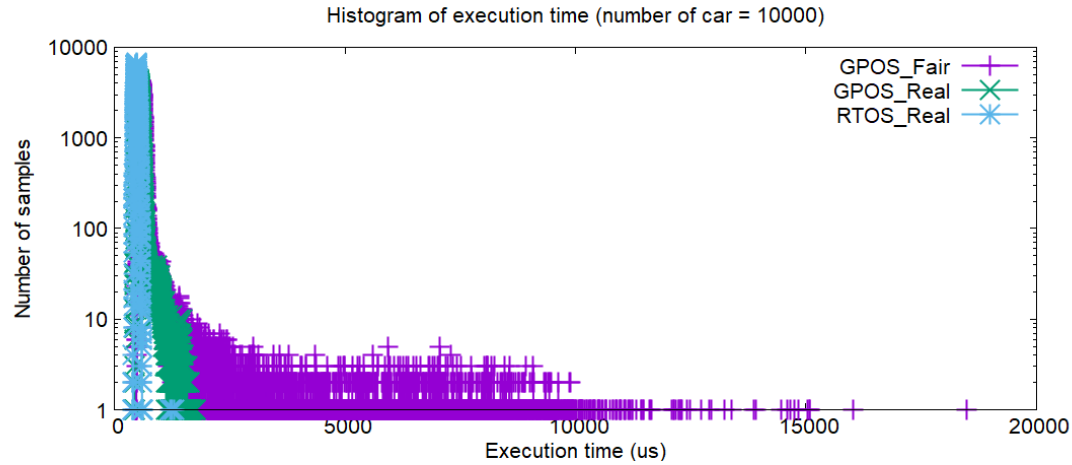

図 5: 処理遅延の度数分布：車両台数 10,000 台

3,4,5,6 に示す。グラフ中の凡例 GPOS_Fair は GPOS の標準スケジューリングポリシー、凡例 GPOS_Real は GPOS のリアルタイムスケジューリングポリシー、凡例 RTOS_Real は RTOS のリアルタイムスケジューリングポリシー、それぞれを用いた場合の処理遅延を表している。また、各車両台数に対する処理遅延の平均一覧を 2、最悪値一覧を表 3 に示す。

車両台数が 100 台の場合の処理遅延に注目する。GPOS の通常スケジューリングポリシーとリアルタイムスケジューリングポリシーを比較すると、平均処理遅延は 0.103 ms から 0.018 ms に減少し、最悪時の処理遅延は 14.056 ms から 1.169 ms に大幅に減少している。また、GPOS のリアルタイムスケジューリングポリシーと RTOS のリアルタイムスケジューリングポリシーを用いた場合と比較し、平均処理遅延は 0.018ms から 0.007ms まで減少し、最悪時の処理遅延は 1.169 ms から 0.026 ms に減少している。最悪時のこの結果から、バックグラ

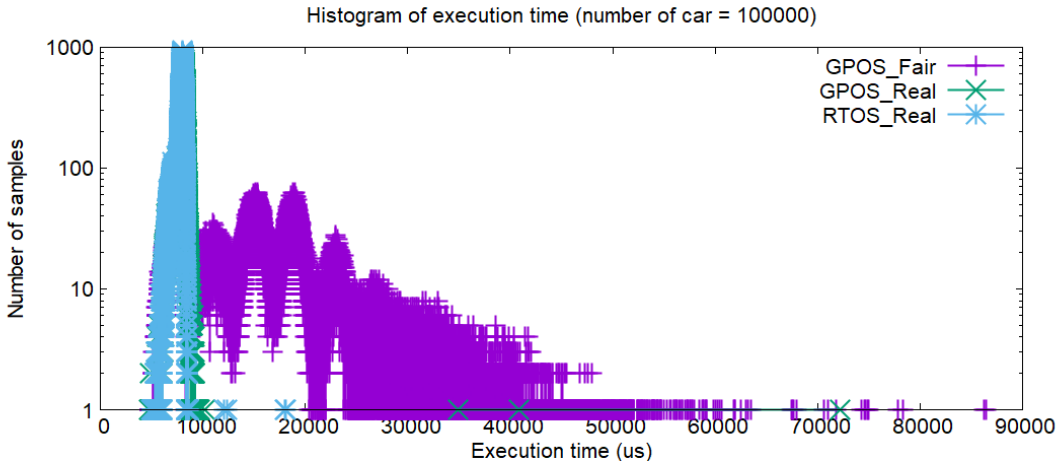

図 6: 処理遅延の度数分布：車両台数 100,000 台

ウンド負荷のある状況下すなわちリソースの競合が発生している状況においては、RTOS のリアルタイムスケジューリングポリシーを用いることにより処理遅延が小さく決定的になり厳しい時間制約に対応できることが明らかとなった。

また車両台数と平均値の関係に注目する。GPOS の通常スケジューリングポリシーとリアルタイムスケジューリングポリシーの平均値の差は車両台数が増加するとともに増加している。例として車両台数 100,000 台に注目すると平均処理遅延は 15.491 ms から 7.889 ms まで減少しており、約半分となっている。また、GPOS のリアルタイムスケジューリングポリシーと RTOS のリアルタイムスケジューリングポリシーの平均値の差は車両台数の増加ともに増えていく傾向がみられた。平均処理遅延は 7.889 ms から 7.886 ms まで減少しているが、処理に要する CPU 時間が増加したことにより RTOS の高い応答性能による遅延低減効果が相対的に小さなものとなっている。

次に車両台数と最悪値の関係に注目する。GPOS の通常スケジューリングポリシーとリアルタイムスケジューリングポリシーの最悪時処理遅延の差は車両台数の増加ともに増大していく傾向がみられる。しかし、100,000 台にまで増加すると、GPOS のリアルタイムスケジューリングポリシー、RTOS のリアルタイムスケジューリングポリシーの計測の際間欠的に大きな遅延が発生しその傾向が乱されている（図 6)。これはリアルタイムタスクが長時間 CPU を占有していることが起因すると考えられる。また、。また、GPOS のリアルタイムスケジューリングポリシーと RTOS のリアルタイムスケジューリングポリシーの最悪時の差は車両台数の増加とともに小さくなっていくが、100,000 で発生する間欠的な遅延によりこの傾向が乱されている。

最悪値付近の遅延の詳細な傾向を見るために、99 パーセンタイル値を表 4 を示す。99 パーセンタイル値に関しては、GPOS の通常スケジューリングポリシーとリアルタイムスケジュー

リングポリシーの処理遅延の差は車両台数の増加ともに増大していく傾向がみられた。また、GPOS のリアルタイムスケジューリングポリシーと RTOS のリアルタイムスケジューリングポリシーの最悪時の差は車両台数の増加しようともほぼ一定であることが読み取れる。

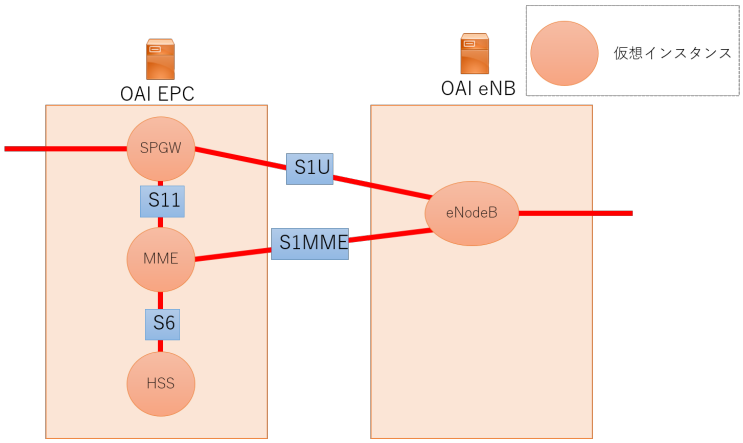

図 7: LTE 通信システムの構成

4　ITS アプリケーションの応答遅延の評価

本章では、エッジコンピューティングを利用する ITS アプリケーションを実装し、実機を用いて ITS アプリケーションの応答遅延を計測する。エッジサーバーに GPOS、RTOS を用いた場合それぞれで応答遅延を計測及びそれらを比較することで RTOS の有効性を評価する。

4.1　衝突検知アプリケーションの仕様

車両環境情報をエッジサーバーが収集・解析し、衝突に関する情報を車両へフィードバックするアプリケーションを実装した。UE（User Equipment）・エッジサーバー間の通信プロトコルには UDP を用いている。実装の詳細を UE 側とエッジサーバ側に分けて説明する。

UE 側のアプリケーションでは実行開始時にエッジサーバーに送信する車両環境情報数、送信間隔を指定することができる。開始時に指定された台数分の車両環境情報を生成し、指定された送信間隔で送信を繰り返す。同時に、エッジサーバーからの応答メッセージを受信し、処理要求毎の応答遅延を記録する。

一方、エッジサーバー側のアプリケーションでは、実行開始時に指定台数分の車両環境情報データベースを作成する。以降、UE から車両環境情報を受信する度、保持している車両環境情報データベース内の車両と衝突検知処理を行い、UE へ応答メッセージを返す。その際、データベースに受信した車両環境情報を追加し、データベース内の最も古い車両環境情報を破棄する。また処理要求毎にスレッドを作成し、作成したスレッド上で衝突検知処理を行うことで、衝突検知処理の並列化を行っている。

表 5: エッジコンピューティング環境の諸元

使用 OS	Ubuntu 18.04 LTS
使用カーネル	GPOS:linux-5.2.17-general
	RTOS: linux-5.2.17-rt9（Preempt-RT）
CPU	Intel Xeon E-2174G（4 コア、3.8 GHz）

表 6: LTE 通信端末の仕様

使用 OS	Android 9 Pie
CPU	MT6739、Quad-Core、4xCortex-A53、1.5 GHz

表 7: 末端端末の仕様

使用 OS	Ubuntu 16.04 LTS
CPU	Intel Core i7-6600U（2 コア、2.6 GHz）

4.2 実験環境

4.2.1 OpenAirInterface による LTE 通信システム

OAI（OpenAirInterface） Software Alliance は 3GPP 準拠のモバイル技術をオープンソースで実装するプロジェクトである [5]。本報告では、OAI が開発する LTE のコアネットワーク EPC（Evolved Packet Core）と基地局（eNodeB）からなる LTE 通信システムを構築した。LTE 通信システムは、コアネットワークには加入者情報を管理する HSS（Home Subscriber Server）や UE とコアネットワーク、コアネットワークとパケットデータネットワークとの接続点となる SPGW（Servicing/Packet Data Network Gateway）、モビリティ制御を行う MME（Mobility Management Entity）が実装されている（図 7）。

4.2.2 実験システム

実験環境のネットワーク構成を図 8 に示す。本実験では、エッジサーバは RTOS と GPOS を搭載したものそれぞれ 1 台ずつ用意し、EPC の先に配置した。すなわち、UE からの通信は EPC を経由してエッジサーバーに送られる構成となっている。構築したエッジコンピューティング環境の諸元を表 5 に示す。また、UE 環境すなわちユーザ側の環境は末端端末と LTE 通信端末で構成される。末端端末はノート PC であり、LTE 通信端末はスマートフォンであ

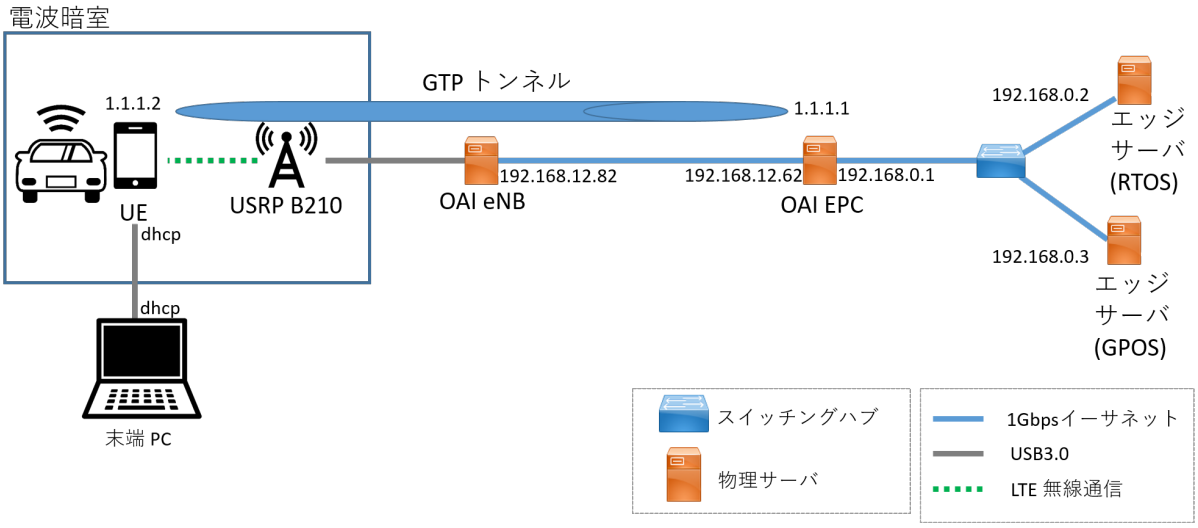

図 8: 実験ネットワークの論理構成

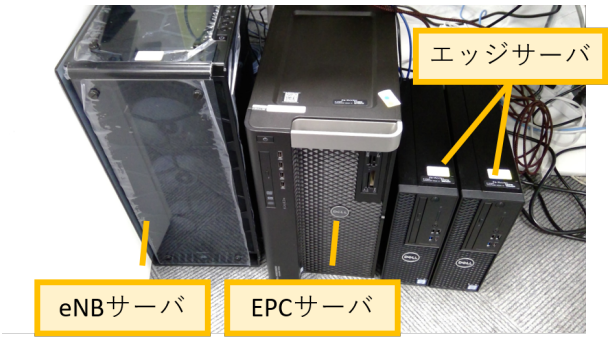

図 9: 実験ネットワーク： eNB、EPC、エッジサーバー

る。末端端末は USB 接続された LTE 通信端末を介してエッジサーバとの通信を行う。LTE 通信端末の諸元を表 6、末端端末の諸元を表 7 に示す。

UE・エッジサーバ間の通信には先述の LTE 通信システムを使用する。ここではエッジサーバから末端への情報送信を想定し、用いた機材の役割を順を追って述べる。図 9 に示すエッジサーバから EPC へパケットが送信される。EPC は LTE 端末が接続している基地局、eNB サーバへ GTP プロトコルを用いてパケットを転送する。次に eNB から USB を介しソフトウェア無線 USRP B210（図 10）へフレームが送信されると無線処理が行われ無線信号へ変換される。この信号は USRP B210 の DL ポートからデュプレクサへと送信され、DL（Down Link) と UL（Up Link）の信号それぞれに分波される。分波された信号はアンテナへ到達する（図 11)。アンテナからの信号を LTE 通信端末が受信し、LTE 通信端末は USB を介して末端端末へと転送する。なお、LTE 通信の周波数帯は Band7 を用いているため、電波暗室内に LTE 端末とアンテナを設置している。

本実験環境では、エッジサーバーは EPC の出口に設置している。その他の選択肢として

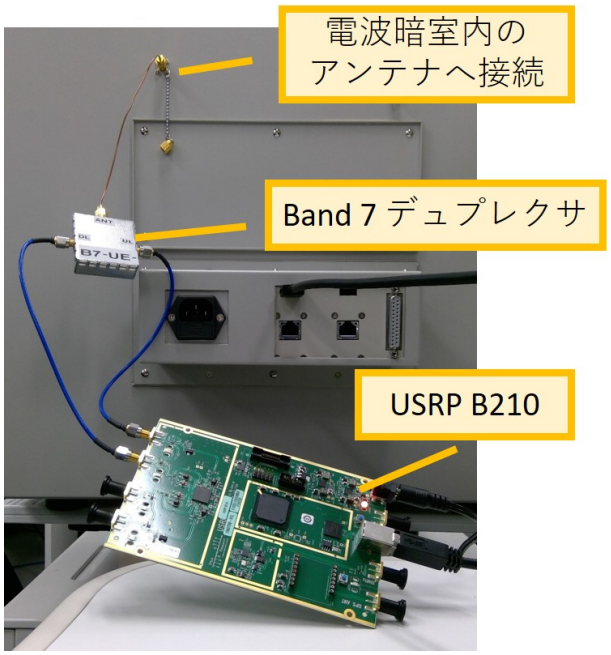

図 10: 実験ネットワーク： ソフトウェア無線機器

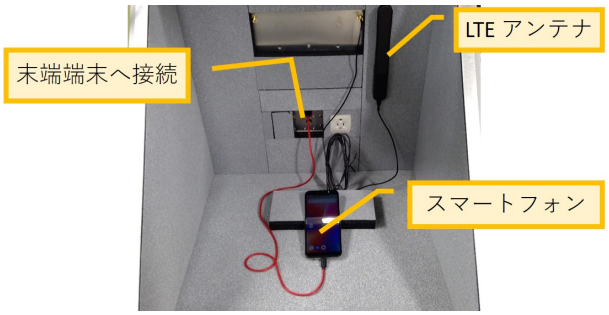

図 11: 実験ネットワーク： LTE 通信端末、LTE アンテナ

は、基地局もしくは複数の基地局を束ねる場所が考えられる。基地局もしくは複数の基地局を束ねる場所にエッジサーバーを設置する場合、GTP トンネルを経由するパケットのペイロード部から通信先を読み出して、エッジサーバーに転送する必要がある。なお、本実験環境では eNB と EPC は物理的に隣接しており、その間の通信遅延は 1 ms 以下であるため、パケットのペイロード部の解析は行わず、EPC の出口にエッジサーバーを設置した。

4.3 応答遅延の測定方法

衝突検知アプリケーションに期待されるのは、処理要求が如何に集中しようとも応答が衝突の一定時間前に確実に返ってくることである。そこで、本報告における衝突検知アプリ

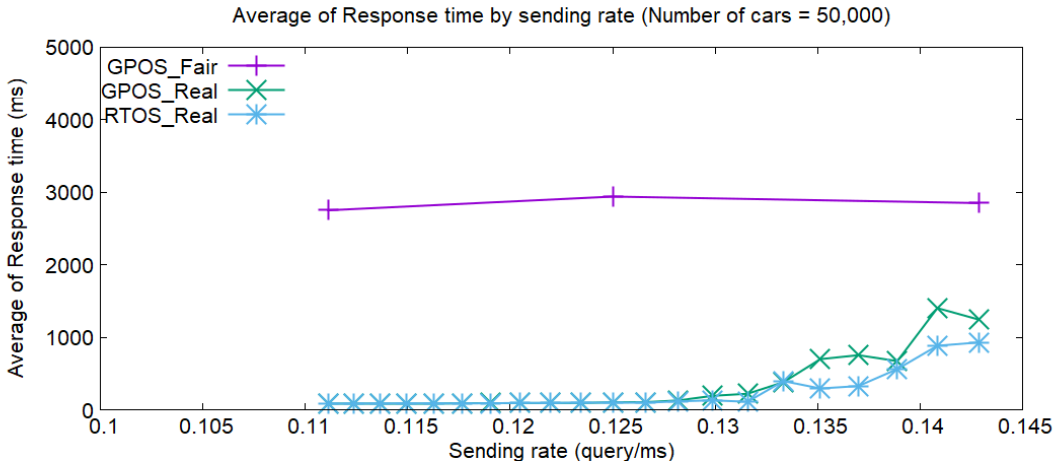

図 12: 処理要求到着レートに対する平均値：車両台数 50,000 台

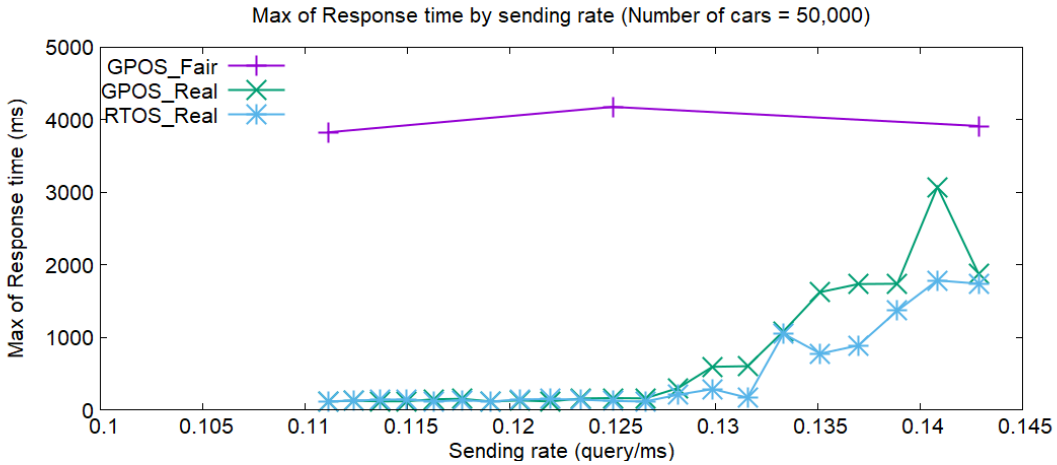

図 13: 処理要求到着レートに対する最悪時応答遅延：車両台数 50,000 台

ケーションの性能評価指標として、処理要求の到着レートに対する応答遅延の最悪値と、処理要求のロス率を用いる。

本測定においては、エッジサーバーの設置場所の違いを想定し、車両環境情報データベースのエントリ数を 1,000、10,000、50,000 とした。また、送信する処理要求数は 4,000 とした。処理要求の到着率は $1/(7+0.1*k)$ [query/ms]（但し $0 \leqq k \leqq 20$）とし、各 k に対応する到着レートに対して応答遅延の計測を行う。なお 3.3 節の処理遅延の計測と同様、1) GPOS で標準スケジューリングポリシー、2) GPOS のリアルタイムスケジューリングポリシー、3) RTOS のリアルタイムスケジューリングポリシーそれぞれを適用した場合で計測する。同様にリソースの競合した状況を想定し stress コマンドを用いてバックグラウンド負荷を掛けている。

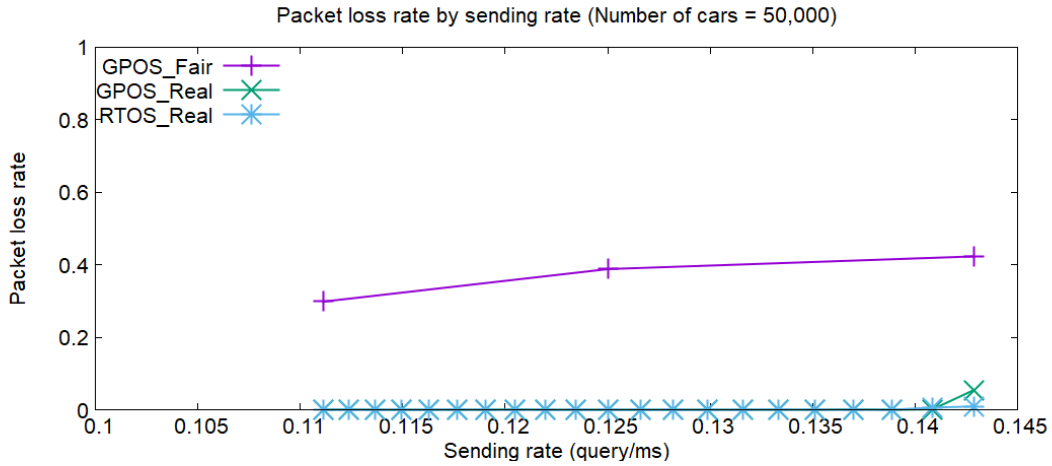

図 14: 処理要求到着レートに対する要求損失率：車両台数 50,000 台

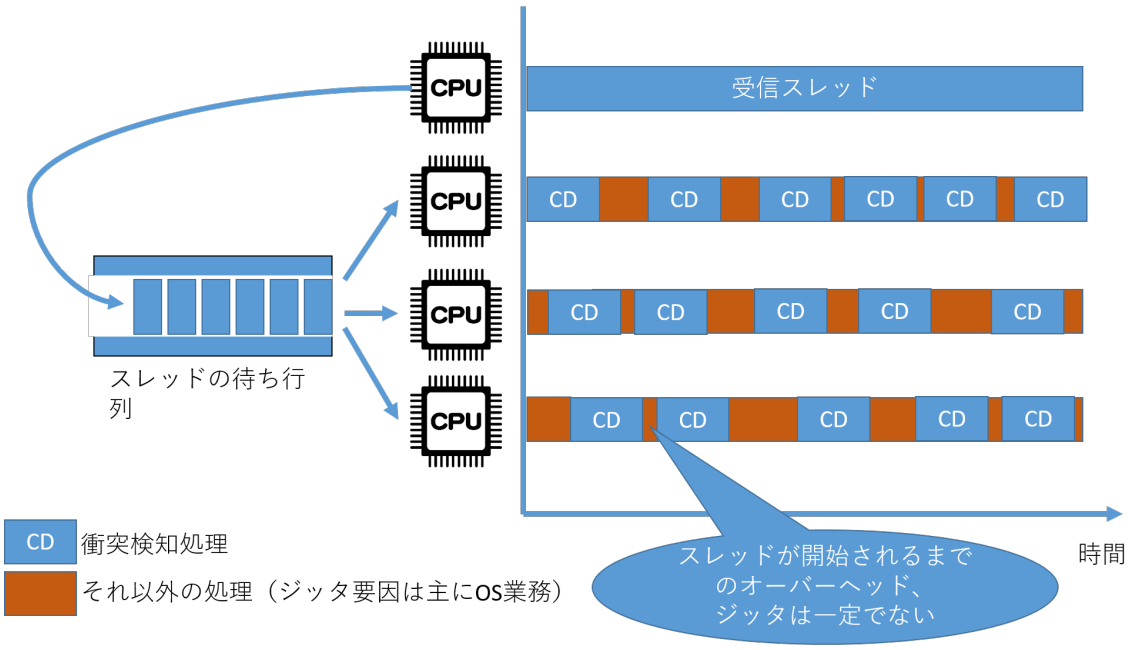

図 15: GPOS における衝突検知処理スレッドの CPU 割当の例

4.4 評価結果

図 13 に処理要求到着率と最悪時応答遅延の関係、図 14 に処理要求到着率と要求損失率の関係を示す。衝突検知アプリケーションの性能比較においては、衝突検知要求の集中に対する応答遅延の最悪値及び要求の損失率が指標となるので、応答遅延の増大及び要求の損失が発生し始める、すなわち処理待ちが発生し始める到着率を選択している。

まず測定結果の GPOS の標準スケジューリングポリシーとリアルタイムスケジューリン

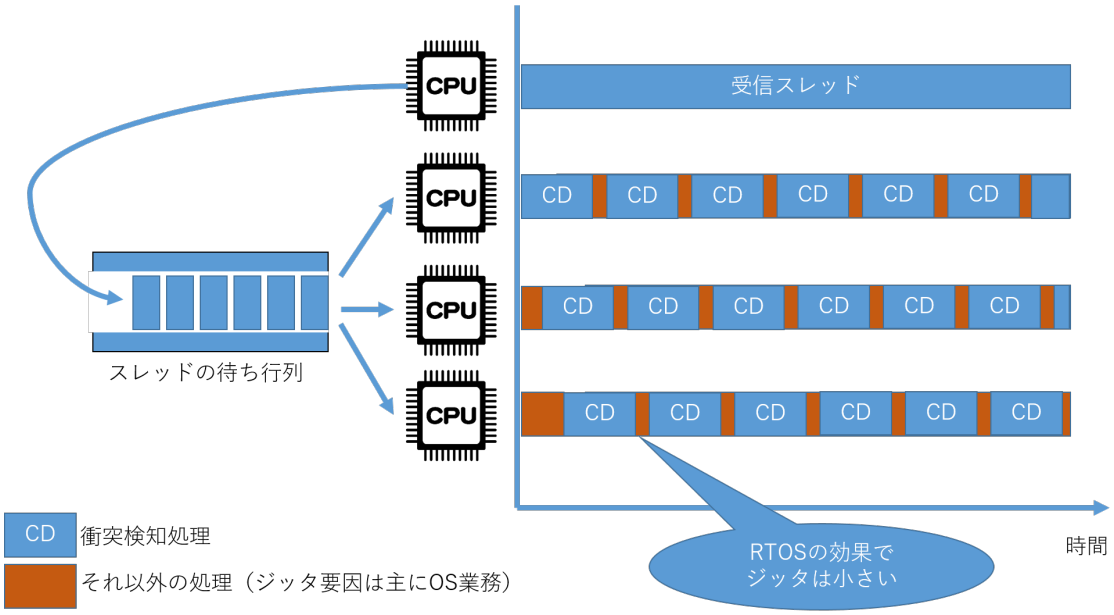

図 16: RTOS における衝突検知処理スレッドの CPU 割当の例

グポリシーを用いた場合に注目する。すると、リアルタイムスケジューリングポリシーの効果により通常ポリシーと比較して応答遅延の平均値、最悪値、また要求損失率ともに大きく減少することが読み取れる。平均値に関しては 1,000ms から 2,000ms 程度、最悪値に関しては 500ms から 4,000ms 程度、要求の損失率に関しては 40%程度の減少効果がみられる。

次に、測定結果の GPOS のリアルタイムスケジューリングポリシーと RTOS のリアルタイムスケジューリングポリシーを用いた場合に注目する。処理要求の到着率が高く処理が集中するような状況では RTOS の効果により応答遅延の平均値が 300ms から 500ms 程度、最悪時の遅延が 500ms から 1,000ms 程度減少していることが読み取れる。また、図 14 から GPOS では間欠的に処理要求の損失が発生するのに対して、RTOS では要求の損失は緩やかに増大していくことが読み取れる。

これらの結果は RTOS の応答遅延を低減する効果が累積的に発揮されるためと考えられる。図 15、図 16 に受信スレッドで作成された衝突検知スレッドがスケジューラの待ち行列に入る様子及び CPU 毎に処理するスレッドの時間的遷移の想定図を示す。

この図のように RTOS では GPOS と比べて応答時間が小さく、単位時間当たりに処理できる衝突検知処理が多くなると考えられる。また、待ちが発生する場合他のスレッドを待つ間スレッドのウェイクアップとディスパッチが繰り返されるため、待ち行列の後方ほど応答時間の影響は累積していく。結果からも、RTOS の最悪時の応答時間が 1ms 低減される僅かな効果であっても、累積的に大きな効果となっていることが確認できる。

5 おわりに

本報告では、エッジサーバーで RTOS（Real-time Operating System）を動作させ、リアルタイム処理が求められるアプリケーションの応答遅延に対する効果を、実機を用いて明らかにした。まず、車両の衝突検知アルゴリズムを実装し、バックグラウンド負荷を高めた条件下でサーバーの処理遅延の最悪値を測定した。その結果、GPOS の Fair スケジューリングの場合に 14 ms であった処理遅延の最悪値が、RTOS のリアルタイムスケジューリングの場合に 0.026 ms となることがわかった。また、GPOS のリアルタイムスケジューリングの場合は、1.169 ms となることが明らかとなった。次に、研究室内に LTE 通信システムを構築し、LTE 使用した UDP 通信により車両位置情報を送り衝突検知結果を受信する衝突検知アプリケーションの応答遅延を測定した。その結果、RTOS によりアプリケーションの応答遅延の最悪値が最大で 30%以上削減される。RTOS の効果は、CPU にタスクを割り当てるまでのディスパッチ時間の短縮化と安定化にあるため、衝突検知の処理要求が高レートで到着する場合に応答遅延の削減効果が高まることがわかった。

本報告の測定では、エッジサーバーに RTOS を導入することで応答遅延は向上するものの、極めて多数の処理要求が発生する場合は 500 ms 以上の応答遅延を要していることが明らかとなっている。従って、アプリケーションのリアルタイム性を確保しつつ処理要求の集中に対応するには、複数のエッジサーバーが処理要求を分担して処理する連携が必要であり、その方策を検討することが今後の課題として挙げられる。

謝辞

本報告を終えるにあたり、大阪大学大学院情報科学研究科村田正幸教授にはお忙しい中貴重なお時間を割いていただき多大なご助言を賜りましたこと深く感謝いたします。大阪大学大学院情報科学研究科荒川伸一准教授には、ご多忙の中進捗と方針の確認や論文執筆指導、時には私が手をこまねいていた LTE システムの構築に遅くまでお付き合いいただくなど手厚くご指導していただきました。心より厚く感謝申し上げます。また、平素よりご指導いただきました先導的学際研究機構大下裕一准教授並びに大阪大学大学院経済学研究科小南大智助教授に心より感謝申し上げます。最後に、的確なご助言をくださった津久井佑樹氏をはじめとする研究室の皆様に感謝の意を表し、謝辞とさせていただきます。

参考文献

[1] N. Lu, N. Cheng, N. Zhang, X. Shen, and J. W. Mark, "Connected vehicles: Solutions and challenges," *IEEE Internet of Things Journal*, vol. 1, pp. 289–299, Aug. 2014.

[2] 5G Automotive Asociation, "C-V2X use cases methodology, examples and service level requirements." `https://5gaa.org/wp-content/uploads/2019/07/5GAA_191906_WP_CV2X_UCs_v1.pdf`, July 2019.

[3] Y. Yu, "Mobile edge computing towards 5G: Vision, recent progress, and open challenges," *China Communications*, vol. 13, pp. 89–99, Nov. 2016.

[4] F. Reghenzani, G. Massari, and W. Fornaciari, "The real-time linux kernel: A survey on Preempt_RT," *ACM Computing Surveys*, vol. 52, pp. 1–36, Feb. 2019.

[5] "OpenAirInterface, 5G software alliance for democratising wireless innovation." `https://www.openairinterface.org/`. accessed: 2019-12-9.

[6] G. Avino, P. Bande, P. Pantelis A Frangoudis, C. Vitale, C. Casetti, C. Fabiana Chiasserini, K. Gebru, A. Ksentini, and G. Zennaro, "A MEC-based extended virtual sensing for automotive services," *IEEE Transactions on Network and Service Management*, vol. 16, pp. 1450–1463, Dec. 2019.

[7] T. E. Trimble, J. F. Bishop, R.and Morgan, and M. Blanco, "Human factors evaluation of level 2 and level 3 automated driving concepts: Past research, state of automation technology, and emerging system concepts," *Report No. DOT HS 812 043, National Highway Traffic Safety Administration*, 2014.

[8] M. Fink, Y. Liu, A. Engstle, and S.-A. Schneider, "Deep learning-based multi-scale multi-object detection and classification for autonomous driving," in *Proceedings of Fahrerassistenzsysteme*, Jan. 2019.

[9] 岡田 隆三、田辺 淳、伴野 守保, "自動車の運転支援・自動化のための画像センシング技術とその実践," 情報処理学会 デジタルプラクティス, vol. 8, pp. 112–119, Apr. 2017.

[10] 児島 亨、廣瀬 敏也、竹内 俊裕、波多野 忠, "歩車間及び車車間通信を活用した自動走行システムのドライバ受容性に関する基礎的研究," 自動車技術会論文集, vol. 49, pp. 1080–1086, Sept. 2018.

[11] M. Malinverno, G. Avino, C. Casetti, C. F. Chiasserini, F. Malandrino, and S. Scarpina, "Performance analysis of C-V2I-based automotive collision avoidance," in *Proceedings of 2018 IEEE 19th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pp. 1–9, June 2018.

[12] "Cyclictest." `https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/cyclictest/start`. accessed: 2020-1-18.

[13] C. S. V. Gutiérrez, L. U. S. Juan, I. Z. Ugarte, and V. M. Vilches, "Real-time Linux communications: An evaluation of the Linux communication stack for real-time robotic applications," *CoRR*, vol. abs/1808.10821, Aug. 2018.