

# 進捗報告資料

安達智哉

to-adachi@ist.osaka-u.ac.jp

平成 31 年 1 月 23 日

## 1 ベアラのタイムアウト時間に着目した CPU とメモリ間の負荷のオフロード

### 1.1 概要

MME および SGW、PGW などの EPC ノードは、主なリソースとして CPU とメモリを持っている。CPU は、アタッチやデタッチなどのシグナリング処理を実行するために必要とされるリソースである。一方メモリは、ベアラなどのセッション情報を保持するために必要とされるリソースである。これらのリソースは、モバイルネットワークにおける通信を可能にするために必須であるため、ネットワーク事業者は、どちらのリソースも枯渇することがないように、CPU とメモリをバランスよく割り当てる必要がある。

その一方で、近年は M2M/IoT 端末の急激な増加が注目されている。M2M/IoT 端末は通信特性において従来の端末 (携帯電話やスマートフォン) とは大きく異なり、データの送信に周期性や間欠性を持つという特徴がある。また、M2M/IoT 端末は消費電力を抑えることを目的に、データの送信ごとにデタッチ処理を実行し、セッションを解放することがある。また、NB-IoT では、DRX (Discontinuous Reception) や eDRX (extended DRX) などに代表される、間欠的なデータ受信により、受信していない期間では無線信号を送受信する機能部を停止させることで消費電力を抑えることを目的とした技術が検討されている [1]。この点においても、ネットワークから切り離されるまで、セッションを維持し続ける従来の端末とは異なる。

上述の M2M/IoT 端末の特性は、CPU やメモリなどのサーバリソースの効率的な割り当てを難しくすると考えられる。なぜなら、M2M/IoT 端末はその通信特性から、多くのアタッチ処理およびデタッチ処理を引き起こし、CPU 負荷を集中的に増加させるため、従来の端末と比較すると、CPU とメモリに与える負荷の割合が異なるからである。また、IoT 端末の接続台数の予測が難しいこともリソースの割り当てを難しくする一因である。IoT 端末は、スマートフォンのような従来の端末とは異なり、家電や自動車、電気メーター、センサなど様々な場所、様々な用途で使用される可能性があり、端末の台数およびその分布を予測することは困難であると考えられる。このように、通信特性が異なり、接続台数の予測が難しい IoT 端末の普及により、今後のネットワーク事業者は、コアネットワークノードへのサーバリソースの割り当てがより難しくなると予想される。

上述のようなネットワーク (サーバリソース消費の予測が難しく、変動が激しいネットワーク) において、収容可能な端末の増加を目的とした既存研究には、スケールアウトの考え方を採用しているものが多い。これらの研究では主に稼働するサーバやインスタンスの数をリソースの需要に応じて変動させることにより、ネットワークの変動に対応している。しかし、この方法では、本来必要とさ

れているリソース量 (需用量) よりも多くのリソースが供給される、オーバープロビジョニングが発生する問題がある。なぜなら、これらの研究では、サーバやインスタンス一台あたりのリソース量は一定であることを前提にした研究が多く、細かい粒度でリソースを制御できないためである。また、必要とされる CPU とメモリのリソース比があらかじめ分かっていることを前提とした研究が多く、必要とされるリソース比が未知の場合はリソースの効率的な利用ができないからである。例えば、CPU のリソース不足を解消するためにケールアウトを行った場合、CPU リソースと同時にメモリリソースも増加する。しかし、メモリは元々ボトルネックにはなっていないため、新たに追加されたメモリはオーバープロビジョニングされたことになる。

このような背景から、CPU とメモリのリソース消費の予測が難しいような状況や変動が大きいような状況においても、どちらかがボトルネックにならずに、効率的にリソースを活用するアーキテクチャを考えることは重要である。実際、CPU とメモリのリソースを効率よく活用する研究は、データセンターなどの分野では行われている [2]。しかし、モバイルネットワークに特化した研究は行われていない。そこで、私はモバイルネットワークに特化した、CPU とメモリのリソースのオフロードを可能にする仕組みを考案する。この方法により、CPU が過負荷である場合は、メモリの負荷を増加させる代わりに CPU の負荷を削減することが可能である。またその逆に、メモリが過負荷である場合は、CPU の負荷を増加させることによりメモリの負荷を削減できる。この仕組みにより、CPU とメモリのリソース消費の予測が難しい場合や、変動が激しいネットワークであっても、CPU およびメモリ双方のリソース利用率の向上が期待でき、収容可能な端末の増加が期待できる。

## 1.2 負荷とそのオフロードの概要

### 1.2.1 CPU 負荷

CPU 負荷は、主にアタッチやデタッチなどのシグナリング処理を各ノードで実行する際に発生する。時間当たりのシグナリング処理の実行回数の増加に伴い、負荷が増加する。CPU 負荷の増加は、各ノードにおける処理時間の増大を引き起こす。また、CPU が過負荷状態になると、それ以上のシグナリング処理の実行が困難になる。

### 1.2.2 メモリ負荷

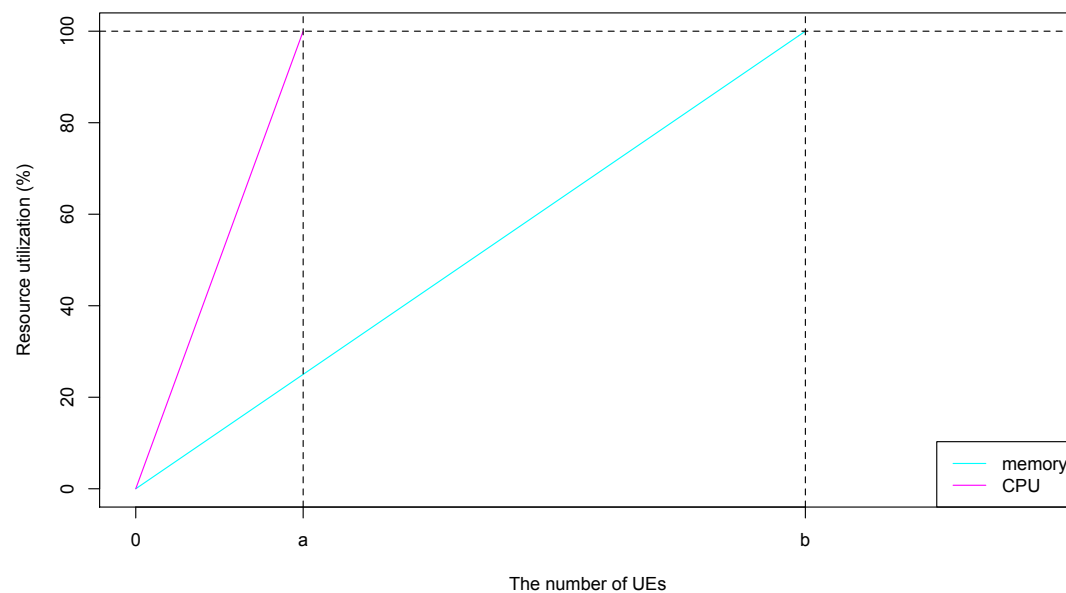
メモリ負荷は、主にベアラなどのセッションを維持するために必要な情報 (ベアラ識別子、UE 情報、ステート情報など) を保持する際に発生する。維持するベアラの増加に伴い、負荷が増加する。メモリが過負荷状態になると、新しくベアラを確立するために、古いベアラを解放する必要がある。

### 1.2.3 負荷のオフロード

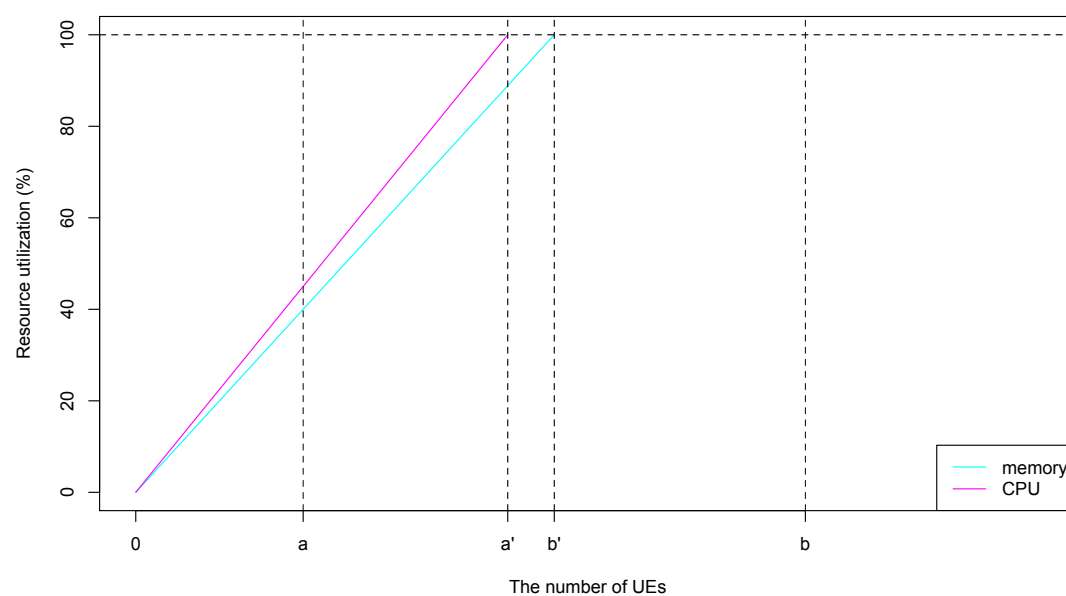
負荷のオフロードに関するイメージ図を図 1 に示す。各図は端末台数と CPU およびメモリリソースの使用率の関係をそれぞれ赤および青の線で示している。端末台数とは、システムが収容している UE の数である。リソースの使用率とは、システムが利用できるリソースの最大値 (物理的な性能の限界値や、事前にオペレータが設定した閾値などが想定される) に対して、UE を収容す

るために割り当てる必要のあるリソース量の割合である。一台あたりの UE を収容するために必要なリソース量の平均値はグラフの傾きに表れており、この値は UE の通信特性に応じて変化する。

図 1 は、メモリリソースと比較して CPU リソースを大きく消費するような通信特性を持つ UE を収容した場合のグラフである。図 1(a) を見ると、端末台数が  $a$  台の時点で CPU 使用率が 100% に達していることがわかる。そのため、収容可能な UE の台数は  $a$  台である。一方、図 1(b) では、第 1.3 節で述べる方法により、一台あたりの UE を収容するために必要なリソース量を変化させた場合のモデルである。この図の例では、ボトルネックであった CPU の負荷をメモリにオフロードすることにより、収容可能な端末台数が  $a$  台から  $a'$  台に増加している。このように、CPU リソースが不足し、メモリリソースが余っているような場合においては、CPU の負荷をメモリにオフロードすることにより収容可能な UE の台数が増加する可能性がある。同様に CPU リソースが余っている状態で、メモリリソースが不足している場合においては、メモリから CPU へ負荷をオフロードすることが可能である。



(a) オフロードを行わない場合



(b) オフロードを行った場合

図 1: CPU 負荷をメモリへオフロード

### 1.3 負荷のオフロード方法

ベアラのタイムアウト時間 (UE がネットワークから切り離されたあと、デタッチ処理が行われるまでの時間) を  $t$  とする。通常、この時間は一定である。私は、 $t$  を変化させることによって、CPU とメモリの間で負荷をオフロードできるのではないかと考えた。図 2 は、M2M/IoT 端末が最初にネットワークに接続した際の様子を示したものである。通常通り、アタッチの処理を行い、ベアラを確立する。図 3 は、M2M/IoT 端末がネットワークから切断された際の様子である。通常は、数秒後にデタッチ処理が行われ、ベアラは全て解放されるが、 $t$  を長く設定した場合、デタッチ処理が行われないため、ベアラはそのまま維持される (UE-eNodeB 間の無線帯域は解放する)。図 4 は M2M/IoT 端末が再びネットワークに接続した時の様子を示す。この場合、以前使用したベアラが残っているため、M2M/IoT 端末は無線部分以外のアタッチ処理をスキップしてデータの送受信を開始できる。結果的に、EPC ノードで行うアタッチ処理およびデタッチ処理の回数が削減されるため、CPU に与える負荷が削減される。一方、ネットワークに接続されていない M2M/IoT 端末のセッション情報を保持する必要があるため、メモリが圧迫されることが予想される。

ベアラのタイムアウト時間  $t$  を超えても端末が再接続しない場合は、デタッチ処理を行う (図 5)。また、タイムアウト後に再接続する場合は、全てのベアラを新たに確立する必要がある (図 6)。そのため、 $t$  を短く設定した場合は、デタッチ処理およびアタッチ処理が頻発するため、CPU の負荷が大きくなることが予想される。一方で、各ノードでセッション情報を保持する時間を短くすることができ、メモリの負荷の削減が期待できる。

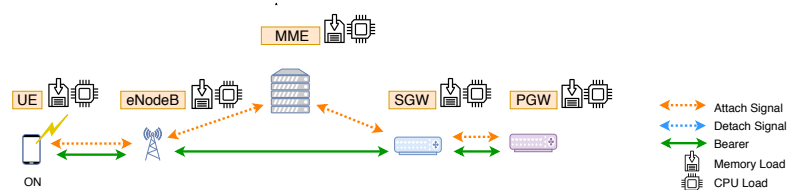


図 2: 端末が最初にネットワークに接続した時と処理

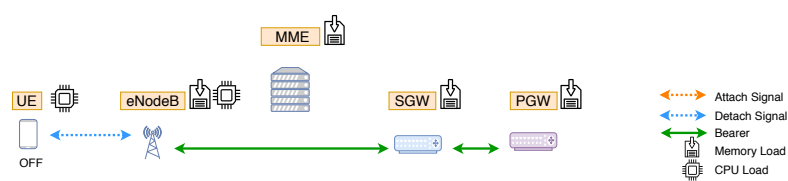


図 3: 端末がネットワークから切り離された時の処理

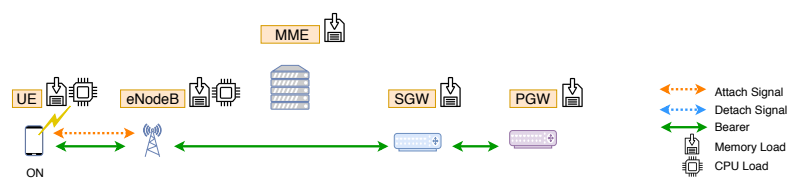


図 4: ベアラのタイムアウト前に端末が再接続した時の処理

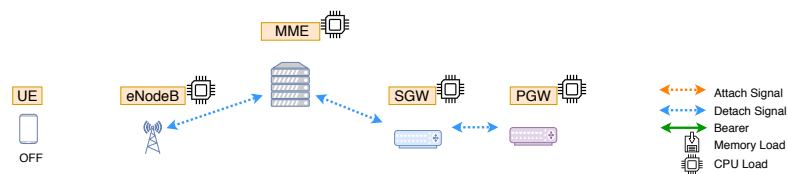


図 5: ベアラのタイムアウトが発生した場合の処理

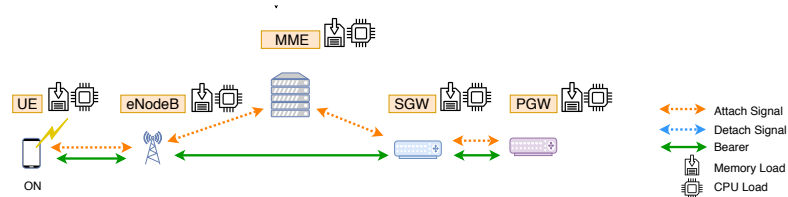


図 6: ベアラのタイムアウト後に端末が再接続した場合の処理

前述のように、ベアラのタイムアウト時間を変更することにより、CPU とメモリの間で負荷をオフロードできると考えられる。これに関する処理のフローチャートを図 7 に示す。なお、この処理を実行することおよび各ノードの負荷を監視することによる追加の負荷は発生しないと仮定して評価を行う。

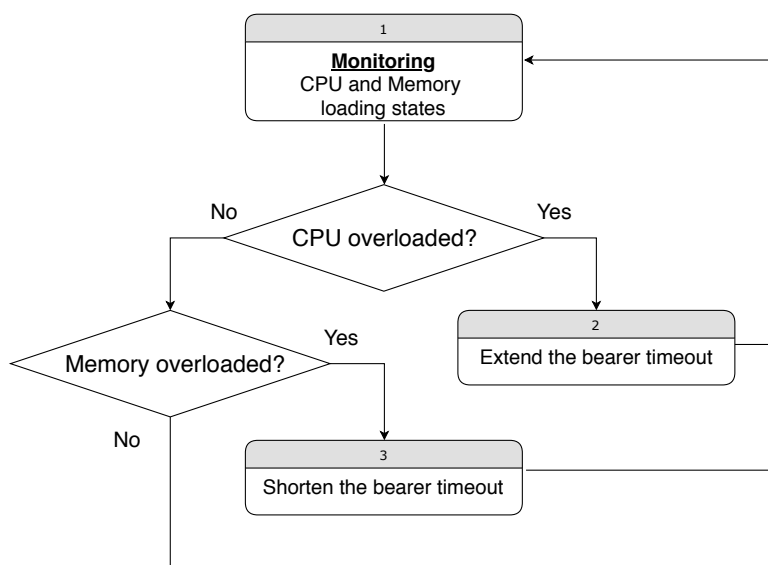


図 7: ベアラのタイムアウト時間の更hands順

## 1.4 評価方法

### 1.4.1 ネットワークモデル

単一の EPC で構成される LTE/EPC ネットワークを対象として評価を行う。図 8 に評価対象のネットワークを示す。図 8 は以下のノードから構成される。

- UE
- eNodeB
- MME
- S/PGW
- HSS

UE および eNodeB は複数台、その他のノードは 1 台ずつ存在する。

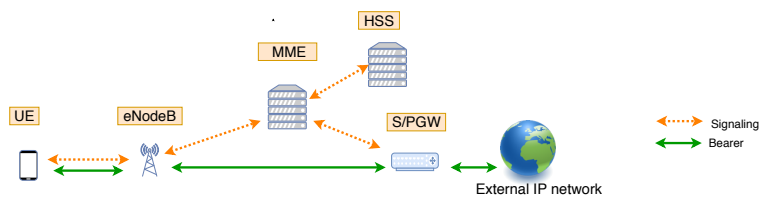


図 8: LTE/EPC ネットワークモデル

#### 1.4.2 シグナリング手順

OpenAirInterface(OAI)[3] に基づき、アタッチ処理およびデタッチ処理のシグナリング手順を決定した。また、シグナリング処理の過程で各ノードにかかる CPU 負荷は、OAI のソースコードの命令文数を指標として用いる。

#### 1.4.3 サーバ資源の割り当て

#### 1.4.4 UE の通信特性

#### 1.4.5 CPU 負荷の導出

CPU にかかる負荷はこれまで行ってきた研究と同様に、プログラム行数に基づく待ち行列理論で求める。アタッチ処理およびデタッチ処理、ベアラのサスペンド処理にかかる時間を待ち行列理論によって導出する。

### 1.5 メモリ負荷の導出

上野さんの実験データを基にメモリ負荷を導出する。実験データから、ベアラの確立が EPC の各ノード (MME、SPGW、HSS) のメモリに与える影響を測定した。まず、上野さんの実験の際に計測されたメモリに関するデータを図 9、図 10、図 11 に示す。図 9、図 10、図 11 はそれぞれ、MME、SPGW、HSS における free memory の推移を表しており、横軸が時間 (s)、縦軸がメモリの空き容量 (KB) を示している。また、UE の接続が発生した時刻を 0 としてプロットしている。これらのグラフを見ると、MME、SPGW、HSS それぞれ、ベアラの確立に伴いメモリ負荷が増加していることが分かる。

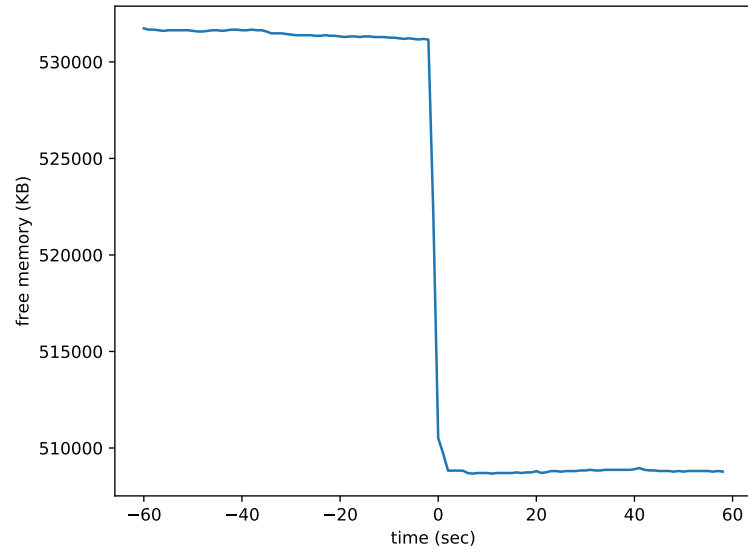


図 9: MME におけるメモリ空き容量の推移

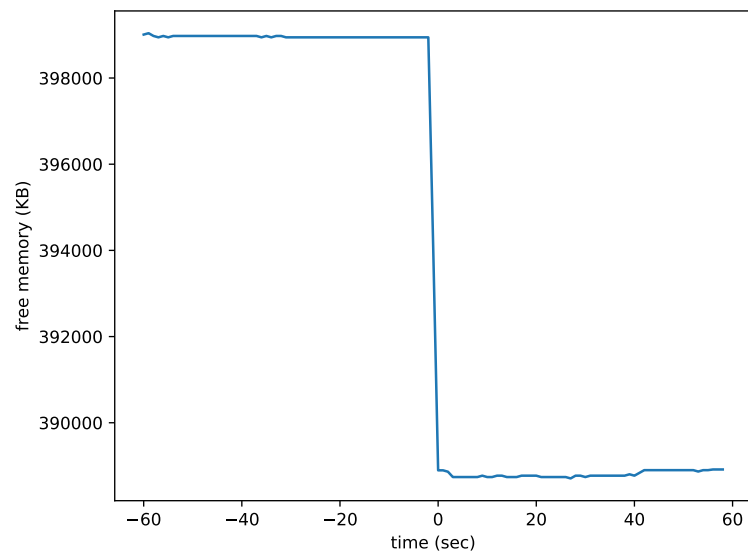


図 10: SPGW におけるメモリ空き容量の推移



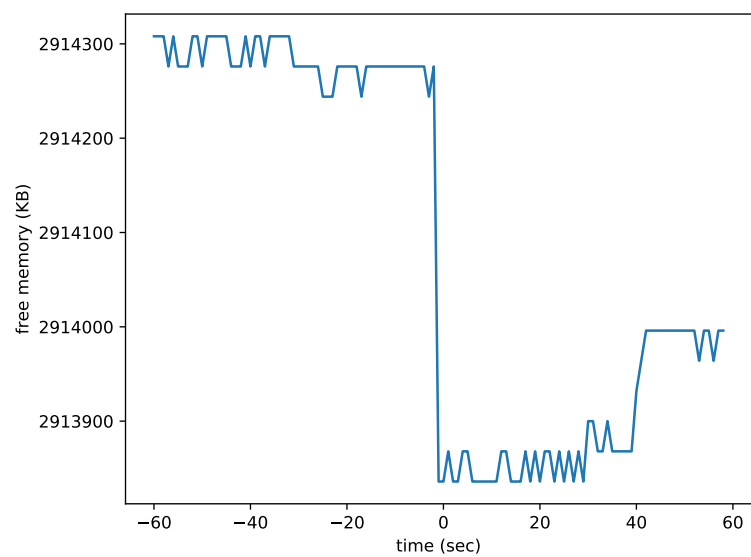


図 11: HSS におけるメモリ空き容量の推移

以上の結果から、ベアラを確立する前のメモリ空き容量とベアラを確立した後のメモリ空き容量を比較することで、ベアラの確立が各ノードのメモリに与える負荷を調べることが可能である。上野さんの実験データ 100 回分それぞれにおいてベアラ確立の前後のメモリ使用量の増加量をプロットした結果を図 12 に示す。この結果より、全てのノードにおいてメモリの使用量が増加していることが分かる。特にメモリ使用量の増加が大きいノードは MME であり、平均 23,371KB 増加している。また、SPGW は MME の 49%、HSS は MME の 1.5%ほどメモリの使用量が増加することが分かった。以下の表 1 にベアラの確立に伴うメモリ使用量の変化の平均と標準偏差を示す。

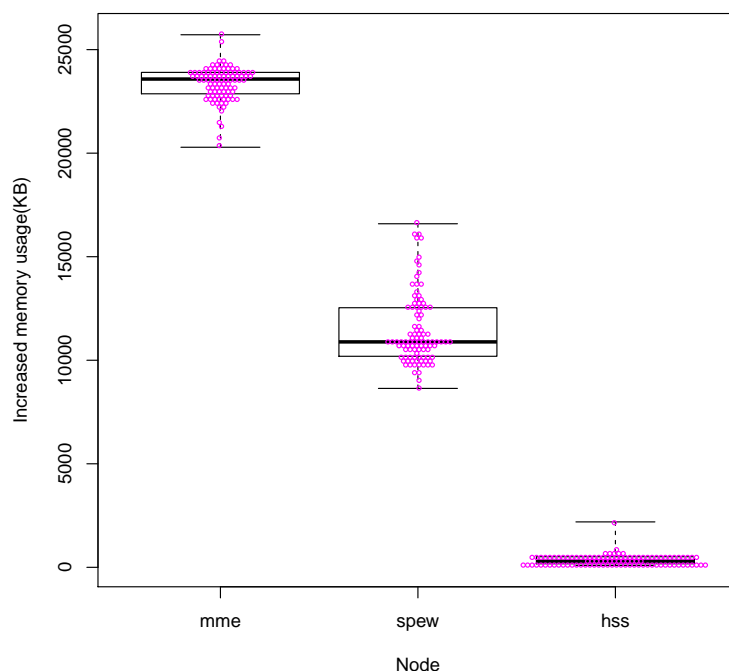


図 12: ベアラの確立によって増加したメモリ使用量

表 1: 各ノードにおける、ベアラの確立に伴うメモリ使用量の増加

Node	平均 (KB)	標準偏差 (KB)
MME	23,371.59	822.42
SPGW	11,458.94	1,698.56
HSS	350.72	251.53

## 2 先行研究調査 (Server Disaggregation)

サーバのリソース分離に関する文献 [2] を調査した。この文献では、データセンタのコストを削減することを目的し、サーバリソースの分離を提案している。そして、サーバごとにリソース構成が固定されている従来のデータセンタと、CPU とメモリのリソース構成をサーバから分離した

データセンタとの比較を行っている。評価の結果、リソースを分離することによって、CPU とメモリのオーバプロビジョニングを削減できることを示している。そして、コスト面では（アプリケーションにも依存するが）最大 40% の削減が期待できると結論づけてる。

### 3 今後の課題

デタッチの処理負荷を求めるため、OAI に基づくデタッチ処理のプログラム行数を調査する。

### 参考文献

- [1] 武田 和晃, ウリ A. ハプサリ, 高橋 秀明, 藤島 大輔, 繆 震, “LTE Release 13 における IoT を実現する新技術,” *NTT Docomo テクニカル ジャーナル*, vol. 24, no. 2, 2016 年 7 月.
- [2] M. Mahloo, J. M. Soares, and A. Roozbeh, “Techno-Economic Framework for Cloud Infrastructure: A Cost Study of Resource Disaggregation,” in *Proceedings of 2017 Federated Conference on Computer Science and Information Systems (FedCSIS)*, Sep. 2017, pp. 733–742.
- [3] “OpenAirInterface.” [Online]. Available: <http://www.openairinterface.org/>