ミーティング資料

安達智哉
to-adachi@ist.osaka-u.ac.jp

2019 年 10 月 17 日

1　Idle タイマの最適化

以前までの評価において、Idle タイマを適切に設定することにより、CPU 負荷およびメモリ使用量の削減が期待できることを示した。それと同時に、Idle タイマの最適な値は、UE の通信周期やその分布に依存して大きく変化することも示した。

一方で、現実的には UE の通信周期やその分布は明らかではない。また、それらは時間とともに変化するものである。そのため、UE の通信周期やその分布が不明であり、動的に変化するような環境においても、Idle タイマを適切な値に設定するような制御方法が必要となる。そこで本章では、Idle タイマの制御方法に関して述べる。

まず、Idle タイマを各 UE に適用するタイミングは、大きく分けて、UE の強制的な状態遷移を引き起こさない方法と引き起こす方法の 2 種類がある。まず、UE の強制的な状態遷移を引き起こさない方法として以下の 2 つが考えられる。

- UE がアタッチしたタイミング
- UE がデータ送信を行うタイミング

次に、UE の強制的な状態遷移を引き起こす方法として以下の 2 つが考えられる。

- UE の動作や状態に依存しない、定期的なタイミング
- 任意のタイミング

それぞれには、メリットデメリットが考えられる。

UE の強制的な状態遷移を引き起こさない方法の場合、更新タイミングが UE の動作に依存するため、新しい Idle タイマを UE に設定するまでにかかる時間が UE ごとに異なるという問題がある。これにより、異なる Idle タイマを持つ UE が同時に存在するような状況が発生するため、Idle タイマの制御が複雑になると考えられる。また、Idle タイマを変更した後、CPU 負荷とメモリ使用量に変化が現れるまでに遅延が発生するため、MME のリソースの制御が難しくなると考えられる。しかし、アタッチやデータ送信など、UE と MME が通信するタイミングで Idle タイマの更新を行うため、追加のシグナリングや状態遷移が少なく、オーバヘッドが小さい。

一方、更新タイミングが UE の動作や状態に依存しない場合、Idle タイマを更新するために UE の状態を変化させる必要がある場合があり、オーバヘッドが大きくなるという問題がある。具体的には、MME と通信できない状態にある UE の Idle タイマを変化させるためには、UE を一度接続状態へと遷移させる必要がある。この際に、状態遷移に伴うシグナリング処理が発生するため、

MME の CPU 負荷が増加する。しかし、Idle タイマを UE に反映させるまでにかかる時間は UE に依存しないため、全 UE の Idle タイマの値を一定期間内で更新できる。さらに、設定する Idle タイマの値を 0 にすることで、MME は任意のタイミングで任意の UE を強制的にアイドル状態へ遷移させることができる。これにより、UE の強制的な状態遷移を引き起こさない方法の場合と比較して MME のリソース制御が容易になると考えられる。

1.1 Idle タイマの制御方法 (UE の強制的な状態遷移を引き起こさない方法)

本節では、UE の強制的な状態変化を引き起こさないことを前提にする。つまり、Idle タイマが切れていない UE を強制的に Idle 状態へ遷移させることはないとする。また、Idle タイマの更新は、UE がデータ送信を行うタイミングで実行するものとする。MME は UE を収容するために使用されている CPU およびメモリリソース量を観測できるものとする。つまり、UE の収容とは無関係な処理によって発生する負荷を取り除いた CPU 負荷およびメモリ使用量を知ることができるとする。MME は現在収容されている UE 台数を観測できるものとする。

突発的な負荷の増加に対応するという観点から、現在収容している UE に加え、最も多くの UE を収容できるような Idle タイマの値が最適と考える。具体的には、現在収容している UE と同じ通信周期を持つ UE がネットワークに参加すると仮定し、最も多くの UE を追加で収容できる Idle タイマの値を最適と定義する。また、CPU よびメモリのどちらも過負荷状態でないことは、UE を収容可能であることの必要十分条件であるとする。

まず、UE 一台あたりが各リソースに与える負荷の平均を推定する。現在収容している UE 台数を N_{UE} とする。UE 台数が N_{UE}、Idle タイマが T の時に観測される、CPU 負荷およびメモリ使用量をそれぞれ $C_{N_{\mathrm{UE}}}(T)$、$M_{N_{\mathrm{UE}}}(T)$ とする。この時、UE 一台あたりが与える CPU 負荷およびメモリ使用量の平均 ($C_1(T)$、$M_1(T)$) は以下の式 (1)、(2) で表せる。

$$C_1(T) = \frac{C_{N_{\mathrm{UE}}}(T)}{N_{\mathrm{UE}}} \tag{1}$$

$$M_1(T) = \frac{M_{N_{\mathrm{UE}}}(T)}{N_{\mathrm{UE}}} \tag{2}$$

Idle タイマを T とした時に、N_{UE} 台の UE を収容している状態から追加で収容可能な UE 台数を $N_{\mathrm{UE}}^{\mathrm{add}}(T)$ とする。$N_{\mathrm{UE}}^{\mathrm{add}}(T)$ は、$C_1(T)$、$M_1(T)$、$C_{N_{\mathrm{UE}}}(T)$、$M_{N_{\mathrm{UE}}}(T)$、$C^{\max}$ および $M^{\max}$ を用いて、以下の式 (3) で表せる。ここで、$C^{\max}$、$M^{\max}$ はそれぞれシグナリング処理および UE のセッション情報を保持するために使用可能な CPU リソース量およびメモリリソース量である。

$$N_{\mathrm{UE}}^{\mathrm{add}}(T) = \min\{\lfloor\frac{C^{\max}-C_{N_{\mathrm{UE}}}(T)}{C_1(T)}\rfloor, \lfloor\frac{M^{\max}-M_{N_{\mathrm{UE}}}(T)}{M_1(T)}\rfloor\} \tag{3}$$

Idle タイマを制御する上での目的関数を以下の式 (4) に示す。

$$\text{maximize}: \quad N_{\mathrm{UE}}^{\mathrm{add}}(T) \tag{4}$$

$N_{\mathrm{UE}}^{\mathrm{add}}(T)$ を最大化する Idle タイマの値が明らかである場合は、その値を Idle タイマに設定すれば良い。しかし一般的に、UE の台数や通信周期は未知であり時間的に変動するため、$N_{\mathrm{UE}}^{\mathrm{add}}(T)$ を最大化する Idle タイマの値を知ることは難しい。そのような場合は、$N_{\mathrm{UE}}^{\mathrm{add}}(T)$ を最大化するように、Idle タイマを適応的に制御する必要がある。具体的には、各リソースの使用量を観測して、$N_{\mathrm{UE}}^{\mathrm{add}}(T)$ を大きくする向きに Idle タイマを変化させる。このステップを複数回繰り返すことにより、Idle タイマを制御する。

この時、1 ステップごとの Idle タイマの変化量を考える必要がある。この値を小さく設定すると、最適な値に到達するまでに大きな時間がかかってしまう場合がある。逆に Idle タイマの変化量を大きく設定すると、Idle タイマが発振する可能性もあり、制御が不安定になる。また、UE の通信周期によって、Idle タイマが変化した時に各リソースの負荷の変化量が異なる点も考慮する必要がある。つまり、ネットワークの変化に短い時間スケールで対応しつつ、安定した制御を実現するためには、ネットワークの環境に応じて Idle タイマの変化量を制御する仕組みが必要である。このような制御には様々な手法が考えられるが、本報告では動作がシンプルであり、汎用性が高い PID 制御を用いる。T および $N_{\mathrm{UE}}^{\mathrm{add}}(T)$ をそれぞれ、PID 制御における入力値および出力値として捉えることで、Idle タイマの変化量を調整しつつ、最適値に近づけることができる。

まず、PID 制御における出力値 $y(t)$ および目標値 $r(t)$ を設定する。以前の評価より、UE 台数を固定した時、CPU 負荷は Idle タイマの値に対して広義単調減少でありかつ、メモリ使用量は Idle タイマの値に対して広義単調増加であることがわかっている。このことから、$C_1(T)$ および $C_{N_{\mathrm{UE}}}(T)$ は T に対して広義単調減少であることがわかる。同様に $M_1(T)$ および $M_{N_{\mathrm{UE}}}(T)$ は T に対して広義単調増加であることがわかる。以上を踏まえて式 (3) を確認すると、$\lfloor \frac{C^{\max}-C_{N_{\mathrm{UE}}}(T)}{C_1(T)} \rfloor$ は広義単調増加でありかつ、$\lfloor \frac{M^{\max}-M_{N_{\mathrm{UE}}}(T)}{M_1(T)} \rfloor$ は広義単調減少であることがわかる。ここで、$\lfloor \frac{C^{\max}-C_{N_{\mathrm{UE}}}(T)}{C_1(T)} \rfloor$ と $\lfloor \frac{M^{\max}-M_{N_{\mathrm{UE}}}(T)}{M_1(T)} \rfloor$ の差分を最小化するような T の集合を $\boldsymbol{T}$ とする。また、$N_{\mathrm{UE}}^{\mathrm{add}}(T)$ を最大化するような T の集合を $\boldsymbol{T}_{\mathrm{optimal}}$ とする。すると、$\lfloor \frac{C^{\max}-C_{N_{\mathrm{UE}}}(T)}{C_1(T)} \rfloor$ は広義単調増加でありかつ、$\lfloor \frac{M^{\max}-M_{N_{\mathrm{UE}}}(T)}{M_1(T)} \rfloor$ は広義単調減少であることを考慮すると、$T \in \boldsymbol{T}$ であることは $T \in \boldsymbol{T}_{\mathrm{optimal}}$ であるための十分条件になる。

以上の議論のイメージを図 1、図 2a および図 2b に示す。図 1 は UE 台数が 500,000 台，UE ごとの通信周期は 10 s から 6,000 s の範囲で一様分布とした時の、Idle タイマと各リソース負荷の関係を示したものである。図 2a は図 1 と同じ UE を収容した時の、Idle タイマと $\lfloor \frac{C^{\max}-C_{N_{\mathrm{UE}}}(T)}{C_1(T)} \rfloor$ と $\lfloor \frac{M^{\max}-M_{N_{\mathrm{UE}}}(T)}{M_1(T)} \rfloor$ との関係を示している。また、図 2b は図 1 と同じ UE を収容した時の、Idle タイマと $N_{\mathrm{UE}}^{\mathrm{add}}(T)$ との関係を示している。図 2b を見ると、$N_{\mathrm{UE}}^{\mathrm{add}}(T)$ を最大化する Idle タイマの値と $\lfloor \frac{C^{\max}-C_{N_{\mathrm{UE}}}(T)}{C_1(T)} \rfloor$ と $\lfloor \frac{M^{\max}-M_{N_{\mathrm{UE}}}(T)}{M_1(T)} \rfloor$ の差分を最小化する Idle タイマの値が一致していることが確認できる。

このことを踏まえ、PID 制御における出力値 $y(t)$ および目標値 $r(t)$ を以下の式 (5)、(6) のように定義する。t は時刻を表す変数である。

$$y(t) = \lfloor \frac{C^{\max}-C_{N_{\mathrm{UE}}}(T)}{C_1(T)} \rfloor - \lfloor \frac{M^{\max}-M_{N_{\mathrm{UE}}}(T)}{M_1(T)} \rfloor \quad (5)$$

$$r(t) = 0 \quad (6)$$

時刻 t における $y(t)$ と $r(t)$ の差を $e(t)$ として以下の式 (7) ように定義すると、PID 制御における操作量 ($u(t)$) は以下の式 (8) で表せる。

$$e(t) = r(t) - y(t) \quad (7)$$

$$u(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(\tau)d\tau + K_d \cdot \frac{de(t)}{dt} \quad (8)$$

ここで、K_p、K_i および K_d はそれぞれ、比例ゲイン、積分ゲインおよび微分ゲインと呼ばれる定数である。これらの定数は、$e(t)$ およびその積分値、微分値が $u(t)$ にどの程度寄与するのかを決定する。

PID 制御を機能させるためには、これら 3 つの定数を適切に設定する必要がある。しかし、一般的に、これらの定数の最適値を数学的に導出することは困難である。そのため、試行錯誤を繰り

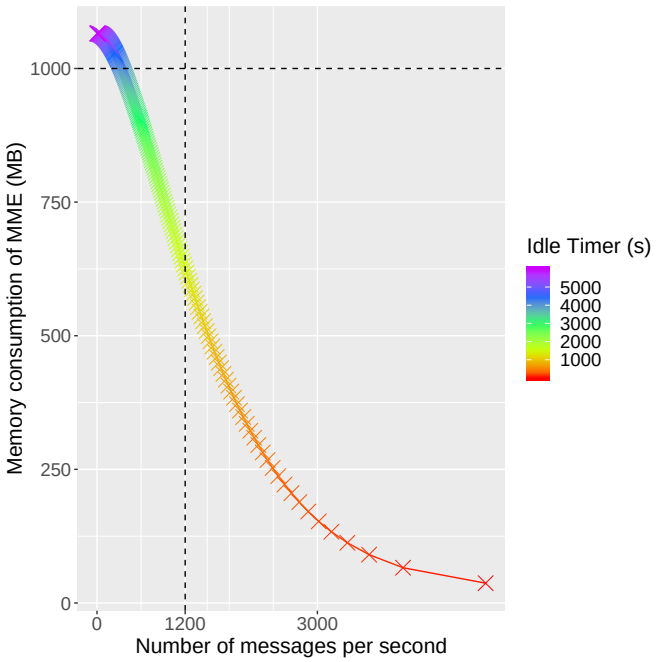
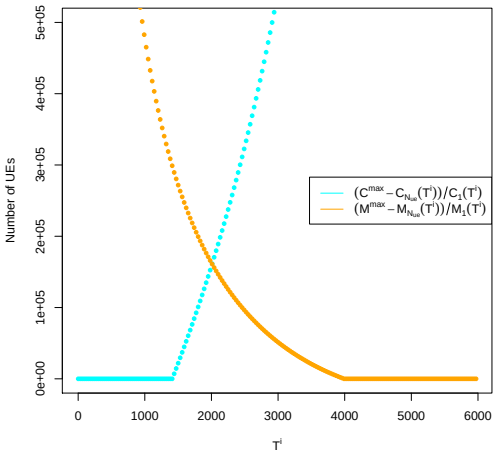
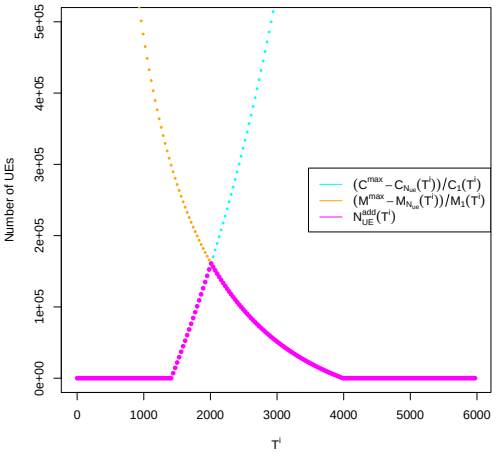


図 1: Idle タイマに対する，メッセージ処理頻度とメモリ使用量の関係



(a) Idle タイマと $\lfloor \frac{C^{\max} - C_{N_{\mathrm{UE}}}(T)}{C_1(T)} \rfloor$ と $\lfloor \frac{M^{\max} - M_{N_{\mathrm{UE}}}(T)}{M_1(T)} \rfloor$ の関係



(b) Idle タイマと $N_{\mathrm{UE}}^{\mathrm{add}}(T)$ の関係

図 2

返しながら経験的にパラメータ調整を行う必要があると言われている。しかし一方で、パラメータの設定方法に関しては、いくつか有名な手順が存在するため、それらに従って設定することもできる。以下に代表的なパラメータの設定手順を示す。

- ジーグラ・ニコルス法
- CHR 法

1.2 Idle タイマの制御方法 (UE の強制的な状態遷移を引き起こす方法)

現在検討中である。

1.3 シミュレーション環境

Idle タイマの制御方式の妥当性を評価するためには、UE および MME の動作をシミュレートする必要がある。UE のシミュレートとは、各 UE1 台ごとの状態や状態遷移、データ送信等の挙動を再現することである。MME のシミュレートとは、全 UE の状態および状態遷移から、MME に発生する負荷を再現することである。Idle タイマの制御機能とは、上述のシミュレータから MME の負荷と UE 台数を取得し、Idle タイマを制御し、更新された Idle タイマを上述のシミュレータへ出力する機能である。Idle タイマの制御機能および MME と UE のシミュレータを図 3 のクラス図に示す。

UE のシミュレートは UE シミュレータというクラスで実装する。UE シミュレータは、各 UE の状態や状態遷移、データ送信等の動作をシミュレートする。これは、UE 台数分のインスタンスを生成し、各インスタンスが特定の UE の通信周期や Idle タイマを保持することにより実現する。

MME のシミュレートは MME シミュレータというクラスで実装する。MME シミュレータは全 UE のリストを保持しており、そのリストを参照することにより、各 UE の状態および状態遷移を取得する。そしてそれらの情報から、CPU 負荷とメモリ使用量をシミュレートする。

最後に Idle タイマの制御機能は、Idle Timer コントローラというクラスで実装する。Idle Timer コントローラは、“Idle タイマの更新 ()”というメソッドを持つが、このメソッドの実装は事前に決定した Idle タイマの制御アルゴリズム (PID 制御など) に依存する。このクラスでは、MME シミュレータから MME の負荷と UE 台数を取得し、Idle タイマの制御に用いる。

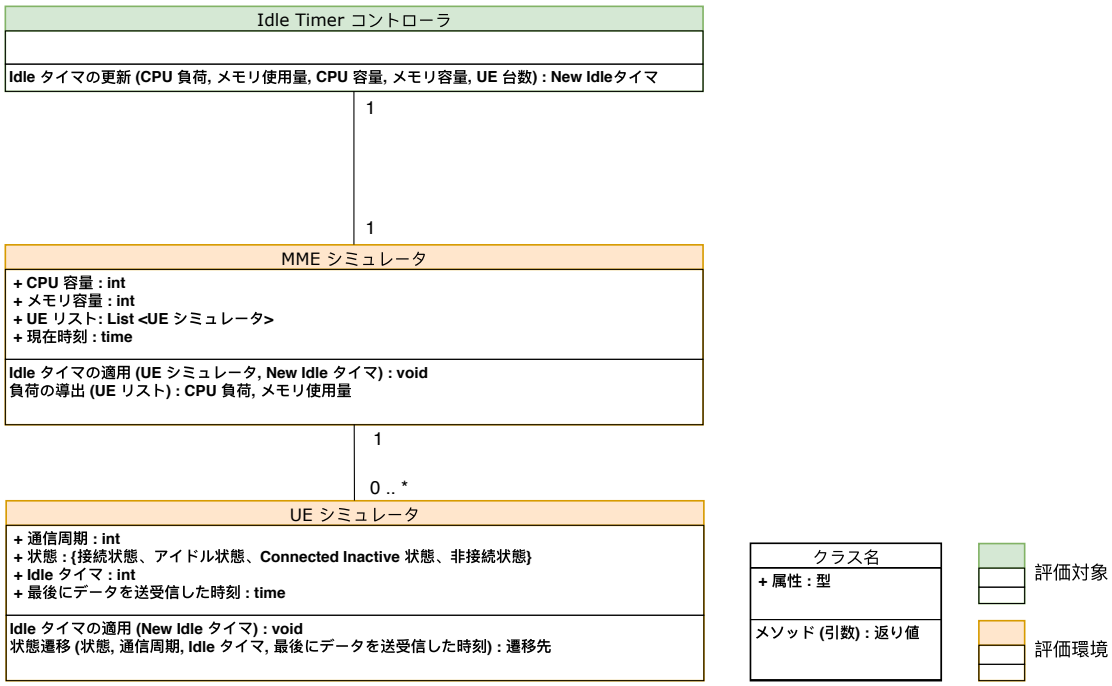


図 3: Idle タイマに対する，メッセージ処理頻度とメモリ使用量の関係

1.4　シミュレーション環境の実装

第 1.3 節で述べたシミュレーション環境を現在実装している。言語は Java である。次回までには完成する予定である。

また、第 1.3 節で述べたように、本評価では UE 台数分のインスタンスを生成するため、PC のメモリや CPU 等に負荷がかると予想される。そこで、シミュレーション環境の実装と並行して、大量の UE を収容したシナリオにおいてもシミュレーションが可能かどうかのチェックを行った。具体的には、100 万台の UE をシミュレーション環境で再現した場合における、PC のメモリ使用量および実行時間を確認した。その結果、メモリに関しては全く問題がないことがわかった (UE の台数分のインスタンスを生成しても、メモリ使用量は高々 50 MB ほどしか増加しなかった)。一方、実行時間に関しては、少し長くなることがわかったため、(約 2 時間程度) プログラムを効率化して、実行速度を向上させる予定である。

2　今後の予定

- PID 制御に関する学習
- シミュレーション環境の実装
- 発表スライドの作成: ～10/18