

進捗報告資料

安達智哉

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

2019年11月20日

1 Idle タイマの制御方法

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、収容可能な UE 台数が最大となる Idle タイマの値を最適と定義する。また、CPU より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 $C_{N_{\text{UE}}}(T)$ 、 $M_{N_{\text{UE}}}(T)$ 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。

$$C_1(T) = \frac{C_{N_{\text{UE}}}(T)}{N_{\text{UE}}} \quad (1)$$

$$M_1(T) = \frac{M_{N_{\text{UE}}}(T)}{N_{\text{UE}}} \quad (2)$$

Idle タイマを T とした時に、収容可能な UE の総数を $N_{\text{UE}}^{\text{capa}}(T)$ とする。 $N_{\text{UE}}^{\text{capa}}(T)$ は、 $C_1(T)$ 、 $M_1(T)$ 、 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。

$$\begin{aligned} N_{\text{UE}}^{\text{capa}}(T) &= \lfloor \min\left\{\frac{C^{\max}}{C_1(T)}, \frac{M^{\max}}{M_1(T)}\right\} \rfloor \\ &= \lfloor N_{\text{UE}} \cdot \min\left\{\frac{C^{\max}}{C_{N_{\text{UE}}}(T)}, \frac{M^{\max}}{M_{N_{\text{UE}}}(T)}\right\} \rfloor \end{aligned} \quad (3)$$

Idle タイマを制御するまでの目的関数を以下の式 (4) に示す。

$$\text{maximize : } N_{\text{UE}}^{\text{capa}}(T) \quad (4)$$

$N_{\text{UE}}^{\text{capa}}(T)$ 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、 $N_{\text{UE}}^{\text{capa}}(T)$

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、 $N_{\text{UE}}^{\text{capa}}(T)$ を最大化するよう、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、 $N_{\text{UE}}^{\text{capa}}(T)$ 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 $N_{\text{UE}}^{\text{capa}}(T)$ 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、式 (3) を確認すると、 $\frac{C^{\max}}{C_{N_{\text{UE}}}(T)}$ は広義単調増加でありかつ、 $\frac{M^{\max}}{M_{N_{\text{UE}}}(T)}$ は広義単調減少であることがわかる。ここで、 $\frac{C^{\max}}{C_{N_{\text{UE}}}(T)}$ と $\frac{M^{\max}}{M_{N_{\text{UE}}}(T)}$ の差分を最小化するような T の集合を \mathbf{T} とする。また、 $N_{\text{UE}}^{\text{capa}}(T)$ を最大化するような T の集合を $\mathbf{T}_{\text{optimal}}$ とする。すると、 $T \in \mathbf{T}$ であることは $T \in \mathbf{T}_{\text{optimal}}$ 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 $N_{\text{UE}} \cdot \frac{C^{\max}}{C_{N_{\text{UE}}}(T)}$ と $N_{\text{UE}} \cdot \frac{M^{\max}}{M_{N_{\text{UE}}}(T)}$ 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 $N_{\text{UE}}^{\text{capa}}(T)$ との関係を示している。図 2b を見ると、 $N_{\text{UE}}^{\text{capa}}(T)$ を最大化する Idle タイマの値と $\frac{C^{\max}}{C_{N_{\text{UE}}}(T)}$ と $\frac{M^{\max}}{M_{N_{\text{UE}}}(T)}$ 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。

$$y(t) = \frac{C^{\max}}{C_{N_{\text{UE}}}(T)} - \frac{M^{\max}}{M_{N_{\text{UE}}}(T)} \quad (5)$$

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

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。

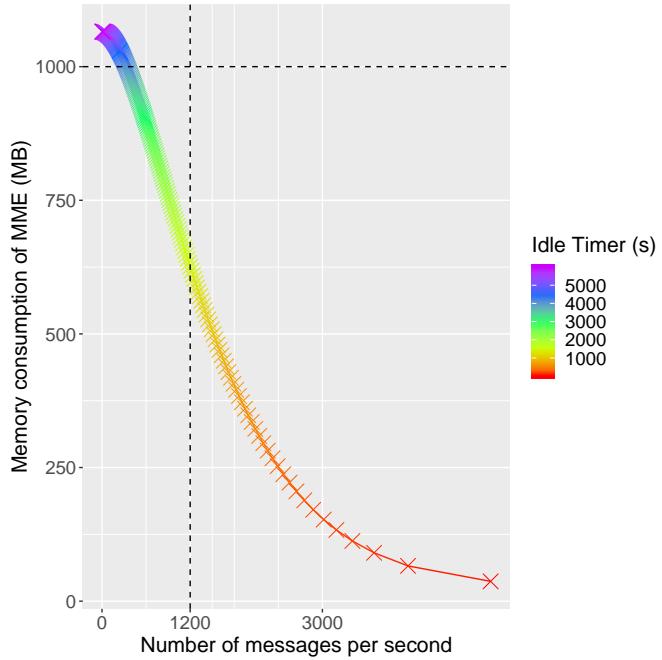
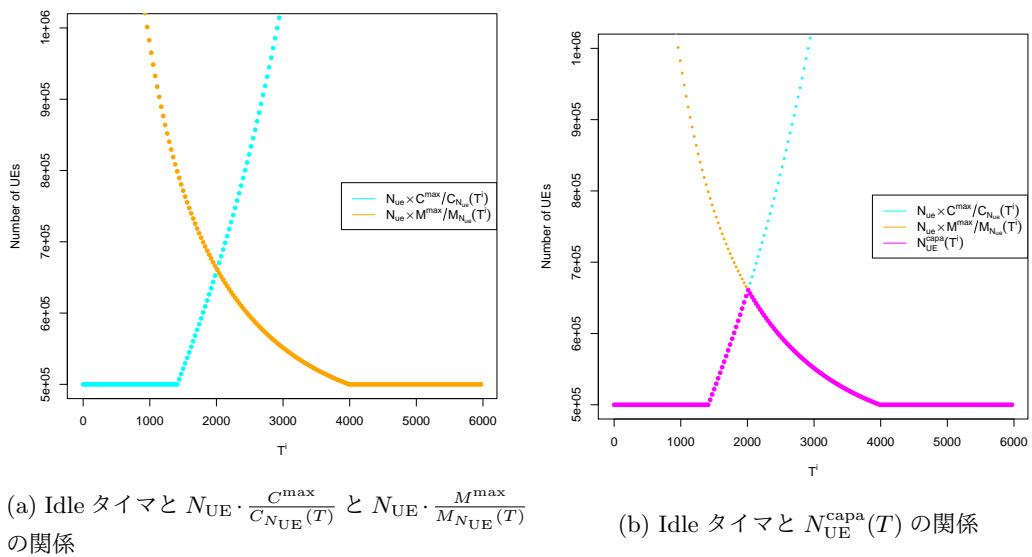


図 1: Idle タイマに対する、メッセージ処理頻度とメモリ使用量の関係



(a) Idle タイマと $N_{\text{UE}} \cdot \frac{C^{\max}}{C_{N_{\text{UE}}}(T)}$ と $N_{\text{UE}} \cdot \frac{M^{\max}}{M_{N_{\text{UE}}}(T)}$ の関係

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

図 2

表 1: ジーグラ・ニコルスの限界感度法

制御の種類	K_p	K_i	K_d
P	$0.5K_u$	0	0
PI	$0.45K_u$	$K_p/0.83P_u$	0
PID	$0.6K_u$	$K_p/0.5P_u$	$K_p \cdot 0.125P_u$

2 ジーグラニコルスの限界感度法

PID 制御においては、比例ゲイン (K_p)、積分ゲイン (K_i)、微分ゲイン (K_d) と呼ばれる 3 つの定数を設定する必要がある。これらの定数を“ジーグラ・ニコルスの限界感度法”と呼ばれる手順に基づき設定した。

ジーグラ・ニコルスの限界感度法に基づくゲインの求め方を以下に示す。

ステップ 1 積分ゲイン (K_i) および微分ゲイン (K_d) を 0 にして調節器が比例動作だけを行うようにする。

ステップ 2 比例ゲイン (K_p) を 0 から徐々に大きくしていき、制御量が安定限界に達して一定振幅振動を持続するようになった時に K_p の増加を止める。

ステップ 3 ステップ 2 の時の比例ゲインを限界感度 (K_u)、振動周期を限界周期 (P_u) とし、これらの値から各ゲインを以下の表 1 のように求める。

ステップ 4 必要に応じてステップ 3 で求めた各ゲインの値を調整する。

以下のシナリオにおいて、ジーグラニコルスの限界感度法を用いて、PID 制御のゲインを求めた。UE 台数は 648,000 台であり、UE の持つ通信周期は 1 day, 2 hours, 1 hour, 30 minutes のいずれかである。それぞれの通信周期を持つ UE の割合は表 3 の通りである。また、各パラメータを表 2 に示す。

まず、限界感度および限界周期を求めるために、 K_i および K_d を 0 にして、 K_p を 0 から徐々に大きくしていった。その結果を図 3 および図 4 に示す。これらの図は $K_p = 0.2$ 、 $K_p = 0.5$ 、 $K_p = 0.52$ 、 $K_p = 0.53$ の場合の結果を示している。左側の図は、タイムステップごとの $r(y)$ (式(7)より) の変化を示している。右図は、タイムステップごとの Idle タイマの変化を示している。これらの図を見ると、 $K_p = 0.52$ 以下の場合は制御が収束するが、 $K_p = 0.53$ の場合は制御が収束せず、一定振幅振動していることがわかる。このことから、限界感度 (K_u) を 0.53 とする。また、限界周期 (P_u) は図 4c より、7450 s とする。

以上の結果を用いて、各ゲインの値は以下の表 4 のように求まる。

表 2: パラメータ設定

Parameter	Numerical setting
T^{ci}	10 s
$s_{\text{MME}}^{\text{c} \rightarrow \text{c}}$	0 messages
$s_{\text{MME}}^{\text{ci} \rightarrow \text{ci}}$	0 messages
$s_{\text{MME}}^{\text{c} \rightarrow \text{ci}}$	0 messages
$s_{\text{MME}}^{\text{ci} \rightarrow \text{c}}$	0 messages
$s_{\text{MME}}^{\text{ci} \rightarrow \text{i}}$	5 messages
$s_{\text{MME}}^{\text{i} \rightarrow \text{c}}$	5 messages
$m_{\text{MME}}^{\text{c}}$	17878 bits
$m_{\text{MME}}^{\text{ci}}$	17878 bits
$m_{\text{MME}}^{\text{i}}$	408 bits
C^{\max}	1200 messages/s
M^{\max}	1,000 MB
d_h	1

表 3: UE の通信周期の分布

	通信周期			
	1 day	2 hours	1 hour	30 minutes
UE 台数の割合	40%	40%	15%	5%

表 4: ジーグラ・ニコルスの限界感度法に基づく設定

制御の種類	K_p	K_i	K_d
P	0.265	0	0
PI	0.2385	0.0000463	0
PID	0.318	0.0000854	296.14

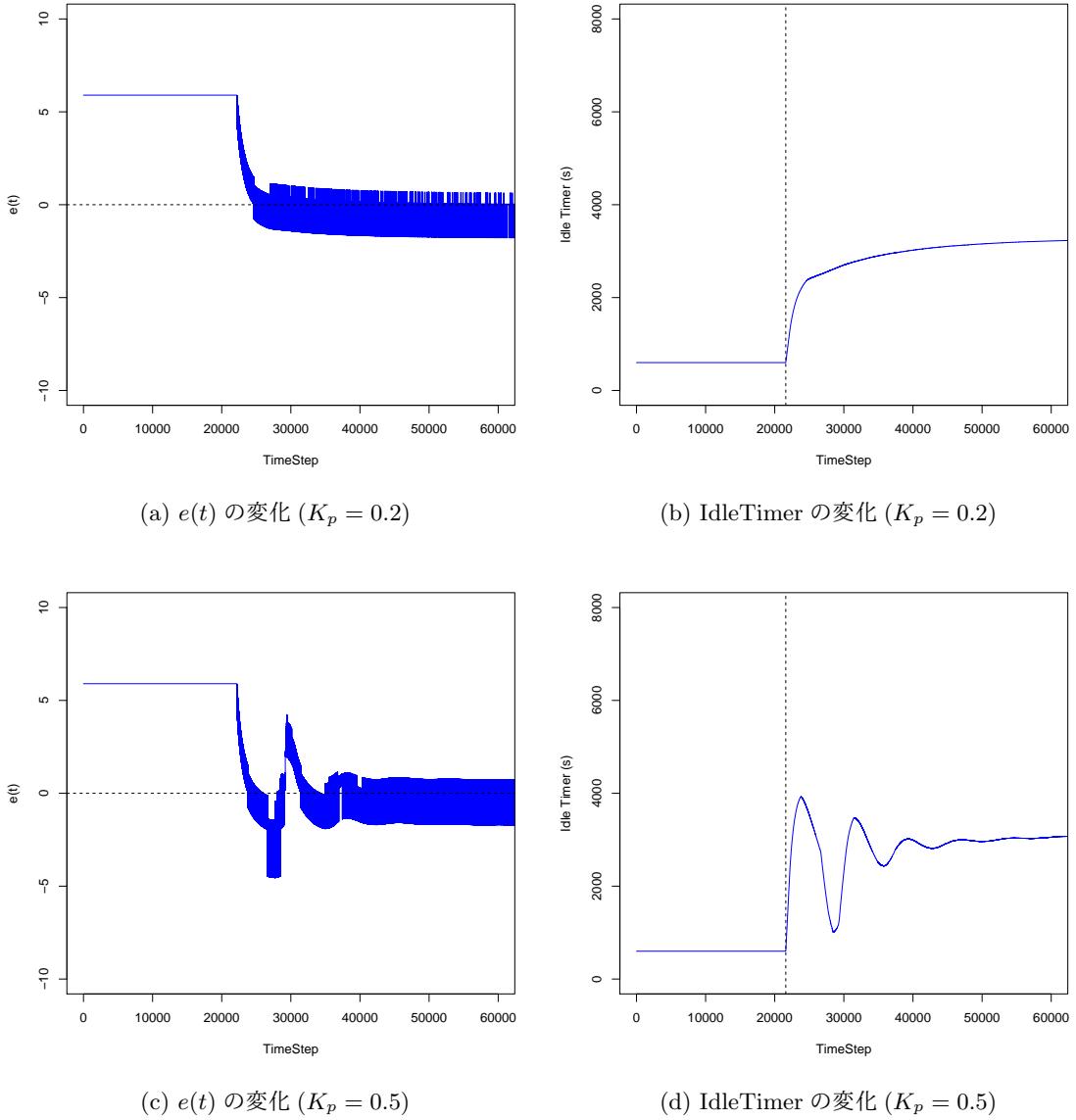


図 3

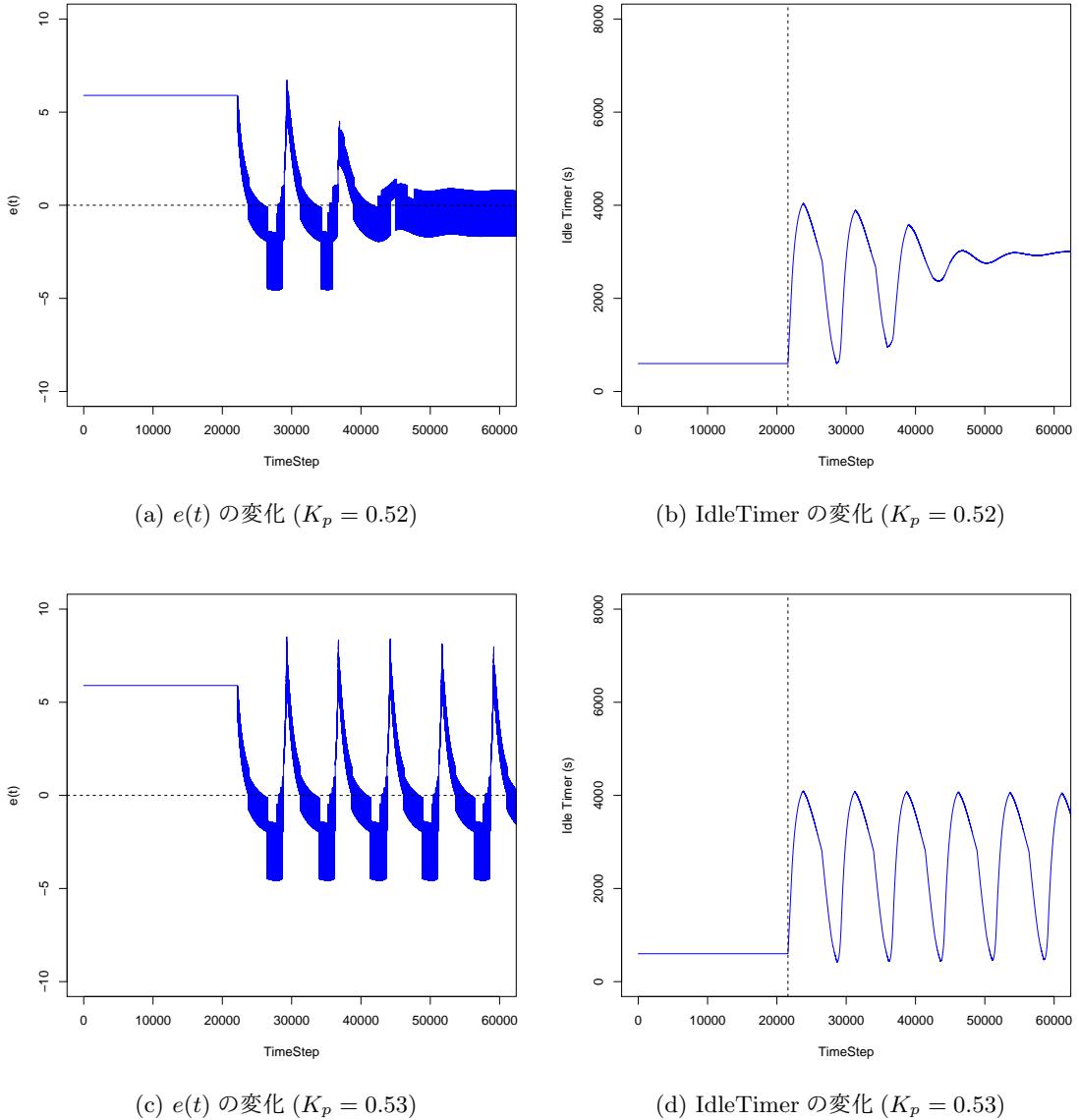


図 4

表4に示したP制御の値を K_p 、 K_i および K_d にそれぞれ設定した場合の評価結果を図5に示す。

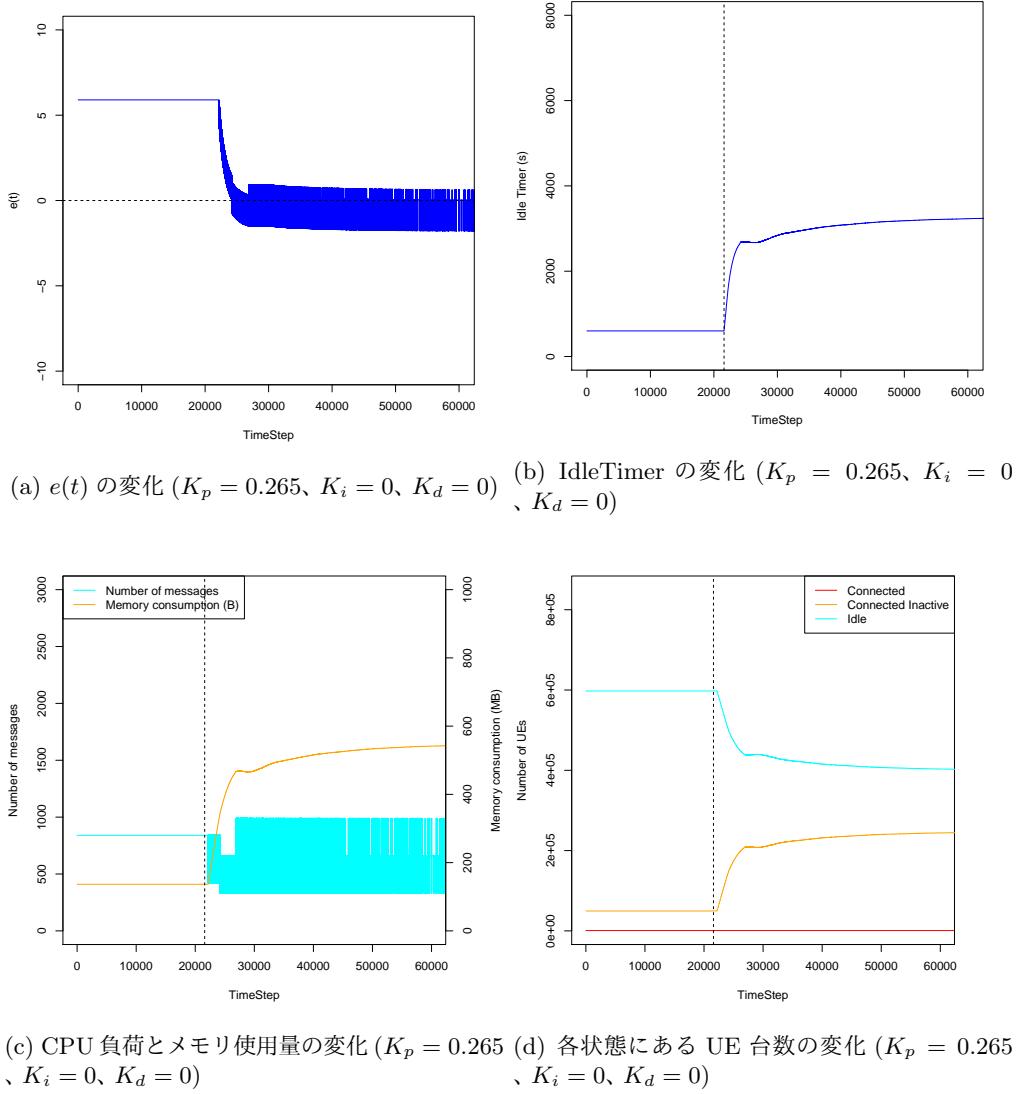


図 5

表 4 に示した PI 制御の値を K_p 、 K_i および K_d にそれぞれ設定した場合の評価結果を図 6 に示す。

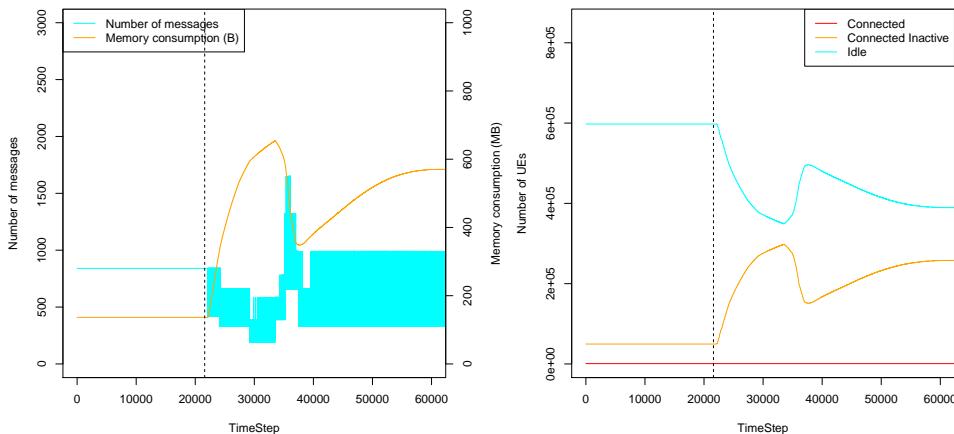
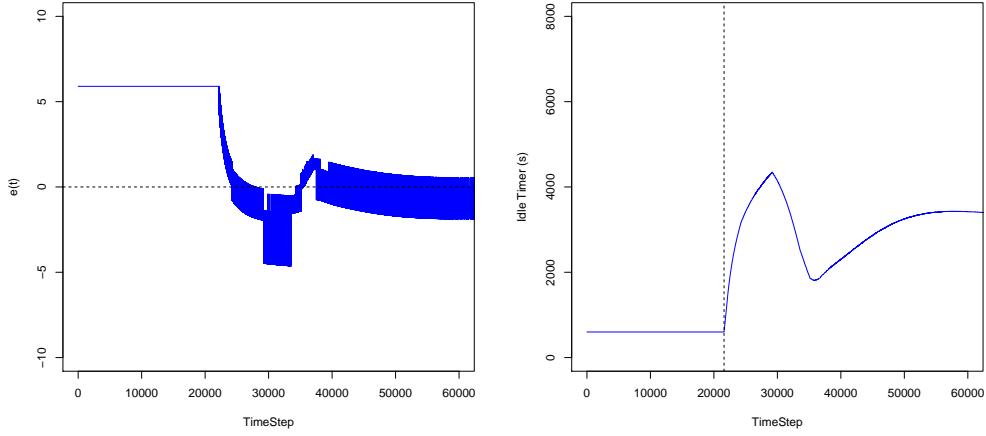
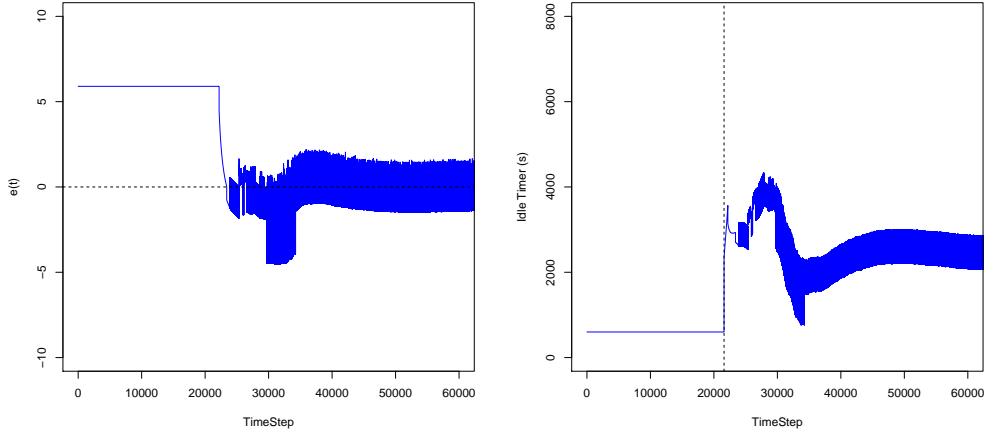
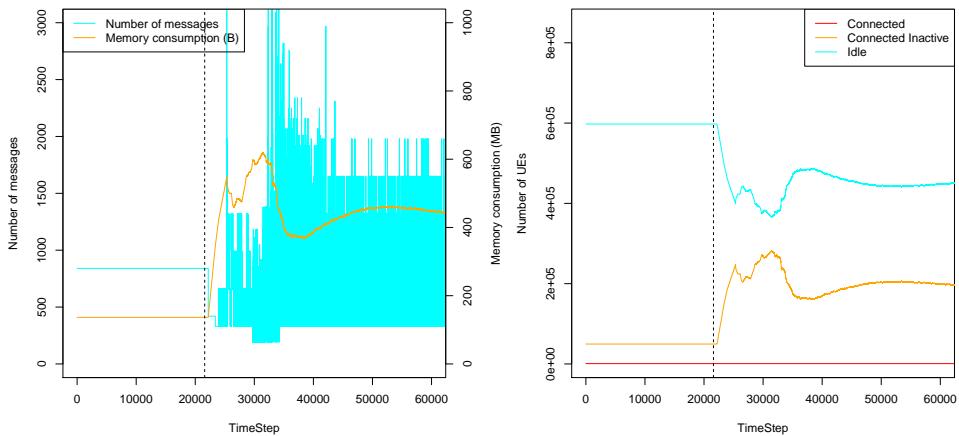


図 6

表4に示したPID制御の値を K_p 、 K_i および K_d にそれぞれ設定した場合の評価結果を図7に示す。



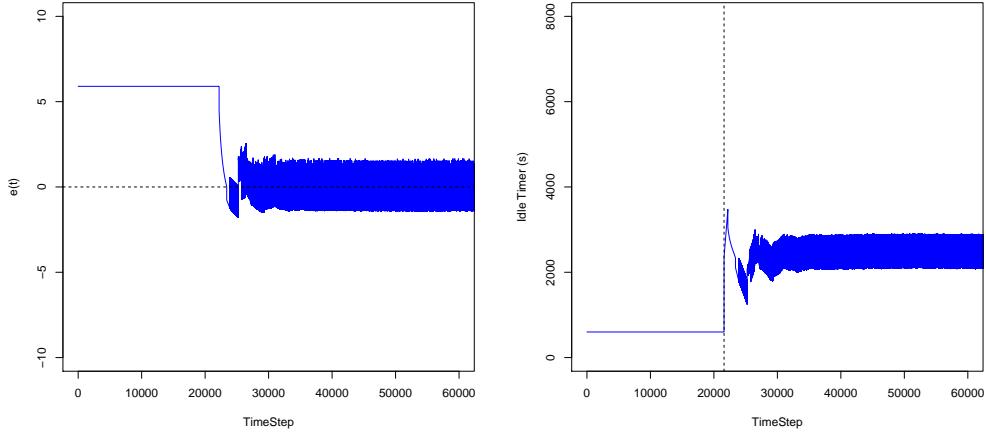
(a) $e(t)$ の変化 ($K_p = 0.318$, $K_i = 0.0000854$, $K_d = 296.14$) (b) IdleTimer の変化 ($K_p = 0.318$, $K_i = 0.0000854$, $K_d = 296.14$)



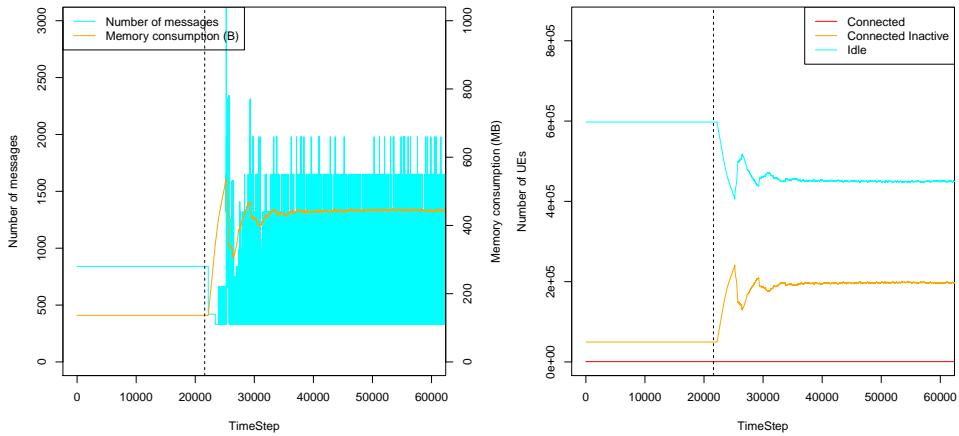
(c) CPU負荷とメモリ使用量の変化 ($K_p = 0.318$, $K_i = 0.0000854$, $K_d = 296.14$) (d) 各状態にあるUE台数の変化 ($K_p = 0.318$, $K_i = 0.0000854$, $K_d = 296.14$)

図7

表 4 に示した PID 制御の値を、 K_p および K_d にそれぞれ設定し、 K_i に 0 を設定した場合の評価結果を図 8 に示す。



(a) $e(t)$ の変化 ($K_p = 0.318$ 、 $K_i = 0$ 、 $K_d = 296.14$) (b) IdleTimer の変化 ($K_p = 0.318$ 、 $K_i = 0$ 、 $K_d = 296.14$)



(c) CPU 負荷とメモリ使用量の変化 ($K_p = 0.318$ 、 $K_i = 0$ 、 $K_d = 296.14$) (d) 各状態にある UE 台数の変化 ($K_p = 0.318$ 、 $K_i = 0$ 、 $K_d = 296.14$)

図 8

以上の図 5、6 図 7 を比較すると、図 5 に示した P 制御が最も制御が安定していることがわかる。また、 $E(t)$ が 0 に収束するまでの時間も最も短くなっている。以上の結果より、比例ゲインに 0.265 を設定した P 制御が Idle タイマの制御に最も適していると言える。

一方で、PI 制御は、P 制御と比較して良い結果が得られなかった。積分ゲインを利用しても制御が改善しなかった理由は、比例ゲインのみで制御した場合に発生するオフセット（定常偏差）が小さいことがあげられる。オフセットとは、定常状態の時に出力値と目標値の差である。一般的にオフセットを 0 に収束させるために積分制御が用いられるが、今回評価している Idle タイマの制御では、元々オフセットが小さい。よって、積分制御を導入するのメリットが小さくなっている。

また、PID 制御では、微分ゲインを利用することにより、偏差発生から定常状態に至るまでの過渡応答特性を改善することができた。これは一般的に微分ゲインを導入するメリットの一つである。しかし、一般的に、微分制御は、「対象の変化」ではなく、ノイズにより過敏に反応する危険性がある。本評価では、離散的なノイズの影響で Idle タイマの制御が不安的になっている。

3 実用 PID 制御についての調査

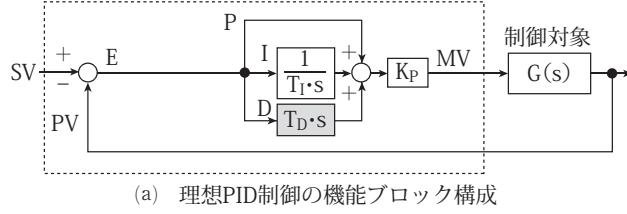
第 2 章の評価において、PID 制御の微分動作がノイズの影響を受けやすいため、今回の評価においては制御が不安定になることがわかった。通常の微分（完全微分）を用いて動作する PID 制御のことを“理想 PID 制御”とよび、以下の式 (9) で表せる。ここで、積分ゲイン $K_i = K_p/T_i$ 、微分ゲイン $K_d = K_p \cdot T_d$ と表し、式 (9) に代入すると、式 (10) になる。この式をラプラス変換すると式 (11) になり、伝達関数 $C(s)$ は式 (12) となる。以下の図 9 に、理想 PID 制御のブロック図を示す。

$$u(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(\tau) d\tau + K_d \cdot \frac{de(t)}{dt} \quad (9)$$

$$u(t) = K_p \cdot \left\{ e(t) + \frac{1}{T_i} \cdot \int_0^t e(\tau) d\tau + T_d \cdot \frac{de(t)}{dt} \right\} \quad (10)$$

$$U(s) = K_p \cdot \left\{ 1 + \frac{1}{T_i \cdot s} + T_d \cdot s \right\} \cdot E(s) \quad (11)$$

$$C(t) = \frac{U(s)}{E(s)} = K_p \cdot \left\{ 1 + \frac{1}{T_i \cdot s} + T_d \cdot s \right\} \quad (12)$$



(a) 理想PID制御の機能ブロック構成

図 9: 理想 PID 制御のブロック図

理想 PID 制御の持つ、ノイズに弱いという欠点を解決するために、ノイズを抑制するローパスフィルタ（一次遅れフィルタ）を組み込んだ制御を実用 PID 制御という。実用 PID 制御のブロック図を以下の図 10 に示す。実用 PID 制御を伝達関数で表現すると、以下の式 (??) になり、プロッ

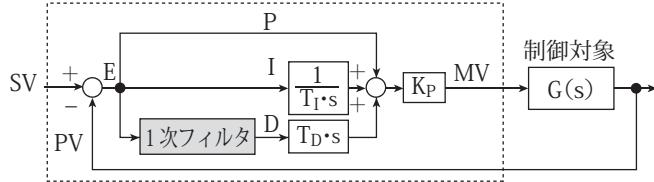


図 10: 実用 PID 制御のブロック図

ク図は図 11 のようになる。ここで η は微分係数という定数であり、通常 0.1 から 0.125 の値を設定する [1]

$$C(t) = \frac{U(s)}{E(s)} = K_p \cdot \left\{ 1 + \frac{1}{T_i \cdot s} + \frac{T_d \cdot s}{1 + \eta \cdot T_d \cdot s} \right\} \quad (13)$$

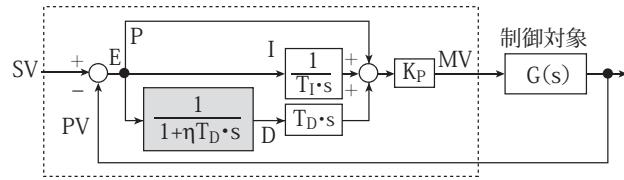


図 11: 実用 PID 制御のブロック図 (伝達関数での表現)

4 今後の予定

- PID 制御に関する学習
- ラプラス変換に関する学習

参考文献

- [1] “実用 PID に向けての工夫（その 1）.” [Online]. Available: https://www.nikko-pb.co.jp/products/k_data/28P_31P_13.pdf