M-022

eBPF を用いたパケットトレーシングにおけるオーバーヘッドの評価
Evaluation of eBPF overhead in packet tracing

田井 佑樹[1)]　　乃村 能成[2)]
Yuki Tai　　Yoshinari Nomura

1. はじめに

ソフトウェア開発において，動作解析や性能測定のためにプログラムの動作ログを取得する場合，ログ出力処理自体が本来の処理に与える影響を見積る必要がある．ログ取得処理挿入のために被測定プログラムを改変する場合，以下の問題がある．1 つは，測定に与える影響の度合が開発者の技量に左右されやすいということ，もう 1 つは，ソースコードを入手できず改変自体が困難な場合もあることである．これに対して，ログ取得に eBPF[1][2] を用いる手法がある．eBPF とは，Linux カーネル内でイベントの発生を記録するために開発された仮想マシンであり，利点として被測定プログラムを改変せずに動作を記録できる点があげられる．また，eBPF は，カーネル内で動作するため，使用できるカーネル内関数やメモリの利用方法に厳しい制約が課されている．このため，eBPF を用いたログ取得処理によるオーバーヘッドは，開発者の技量や動作環境に影響されにくいと考えられる．そこで本稿では，被測定プログラムを改変した場合と eBPF を用いた場合についてログ取得によるオーバーヘッドを比較し，eBPF のオーバーヘッドを見積る．以降では，eBPF を用いてパケットトレーシングを行う方法とオーバーヘッドについて述べる．

2. eBPF を用いたパケットトレーシング

2.1 eBPF 概要

eBPF には，システムコールを監視する機能がある．その動作を図 **1** に示し，以下で説明する．

(1) ユーザが記述した eBPF 仮想マシンコードをカーネル内の eBPF 仮想マシンに読み込ませる．
(2) eBPF 仮想マシンコードは，イベント監視をカーネルに依頼する．
(3) 被測定プロセスによるイベント発生を契機にカーネルからの通知が来る．
(4) eBPF 仮想マシンコードは，eBPF 仮想マシン内で確保した保存領域にイベントの状態を保存する．
(5) eBPF を用いた測定プロセスは，上記の完了を待って，保存領域をユーザ空間に取得する．

以降では，通信プログラムの送受信に関するシステムコールを監視することで，通信プログラムのパケットトレーシングを行った結果について示す．

2.2 eBPF によるメッセージブローカの性能測定

本稿では，eBPF を用いてパケットトレースを行う対象としてメッセージブローカを扱う．メッセージブローカとは，分散システムに用いられるメッセージキューサーバであり，計算機から計算機へのパケット送信を中継する．メッセージブローカを用いたシステムの処理流れを図 **2** に示す．図 2 は，センダが送信したパケットをメッセージブローカがレシーバに中継する様子を示している．なお，通常，メッセージブローカは，複数対複数のセンダ – レシーバ間を中継するシステムであるが，図中では，各 1 プロセスとしている．

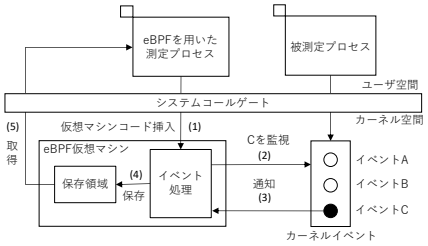

図 1　eBPF を用いた性能測定の流れ

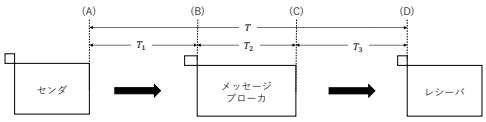

図 2　メッセージブローカを用いたシステムの処理流れ

eBPF を用いることで，各プロセスの送受信，つまり (A)，(B)，(C)，(D) を契機としてパケットのヘッダやペイロードにタイムスタンプを挿入できる．また，全パケットの id と通過時刻を eBPF 内で全て保存しておいて，後から突き合わせることも可能である．これによって，各処理にかかる時間 (T_1, T_2, T_3) を測定できる．これを本稿では，パケットトレーシングと呼ぶ．

また，eBPF を用いずに各プロセスのプログラムを改変することで同様の測定を行えるが，各プログラムを改変したり，場合によってはプロトコルの変更をしたりするための手間がかかる．

以降では，この環境において，メッセージブローカのパケット受信から送信にかかる時間 (T_2) を測定対象とし，eBPF による測定の有無で T_2 がどう変化するか，つまり eBPF による測定のオーバヘッドを評価する．

3. オーバーヘッドの測定

3.1 測定の考え方

2.2 節で述べたメッセージブローカがパケットを送受信した時刻を記録することで，eBPF による測定のオーバヘッドを評価する．具体的には，eBPF を用いて T_2 を測定する際の処理で生じるオーバヘッドを測定し，被測定プログラムの改変によって T_2 を測定した場合のそれと比較する．ここで，単純に図 2 中の (B) と (C) の時刻を取得して差分を取っただけでは，T_2 内のオーバヘッドが占める時間の測定や比較ができない．以下に理由を述べる．

- 測定しない場合の T_2 を測定できない

1）岡山大学大学院自然科学研究科，Graduate School of Natural Science and Technology, Okayama University
2）岡山大学学術研究院自然科学学域，Faculty of Natural Science and Technology, Okayama University

T_2 内のログ取得処理が占める時間を知るには，(B) と (C) に探針を入れた場合と入れない場合を測定したいが，直接それを取得できない．

- 測定手法によって (B) と (C) の位置が異なる
  eBPF を用いた場合では，システムコール発行直後のカーネル内を測定ポイントとするのに対して，被測定プログラムの改変による場合は，ユーザランドのプログラム実行中にある．従って，測定した T_2 の比較では，オーバヘッドの差が分からない．

以上より，T_2 を (B) と (C) の時刻の差によって求めて比較することは不適であるといえる．そこで，以下の方法によって，T_2 中のオーバヘッドを計算する．

(1) 各 (A)，(D) の時刻を取得し，その差から T を得る
これは，全ての測定において，同一の手法で取得する．
(2) 各 (B)，(C) の時刻を目的の測定手法で取得する
これは，オーバヘッドを測定したい手法 (例えば，eBPF) によって測定処理を実行して，オーバヘッドを生じさせるのが目的で，これによって T が変化する．
(3) (2) の手法を変えて各 (1) で得られた T を比較する
T の時間の差は，(2) において T_2 を測定する手法の差によって生じている筈であるから，これにより各手法間のオーバヘッドの差が分かる．また，(2) において何も処理を行わなければ，オーバヘッドそのものを測定できる．

本稿では，送受信される各パケットにおいて図 2 の (A)，(D) の時刻をセンダ，レシーバのプログラムを改変することで取得し，T を算出する．

ここで，センダの送信間隔が短い場合，パケットがメッセージブローカからレシーバに送信される前に次のパケットがセンダから送られる．この際，バッファにパケットが滞留するため，最初に受信したパケットにおける T_2 と最後に受信したパケットにおける T_2 には大きな差が生じる．なぜなら，最後に受信したパケットは，滞留している他のパケットが送信されるまで送信されないためである．このため，メッセージブローカのバッファにパケットが滞留しないように十分な送信間隔を取り，測定を行う必要がある，以降の測定では，送信間隔を 60 μs とした．

3.2 オーバーヘッドの測定環境

本稿では，以下 4 つの場合の図 2 に示す T を測定する．

(1) ログ取得をしない場合
メッセージブローカを改変しないで，eBPF によるログ取得も行わない場合である．この測定結果を基準に他の場合のオーバーヘッドを算出する．
(2) ログ取得にメモリ格納を用いた場合
メッセージブローカを改変して，送受信のたびに時刻をプログラム中に確保した配列に保存する．これは，ログ取得による処理時間の増大が小さくなるようなメッセージブローカの改変例である．
(3) ログ取得にファイル出力を用いた場合
メッセージブローカを改変して，送受信のたびに時刻をファイル出力する．これは，ログ取得による処理時間の増大がやや大きくなるようなメッセージブローカの改変例である．
(4) eBPF を用いた場合
メッセージブローカを改変しないで，eBPF プログラムを動作させ，メッセージブローカの送受信のたびに時刻をカーネル内に確保したメモリに格納する例である．

表 1　測定環境

項目	内容
カーネル	Linux 5.11.0-43-generic
CPU	AMD Ryzen 9 5950X 16-Core Processor
CPU クロック	3400 MHz
メモリ	64 GB
SSD	CSSD-M2B1TPG3VND, M.2-2280 (NVMe)

表 2　測定手法の違いによるオーバーヘッドの比較結果 (送信間隔: 60 μs)

測定手法	測定時間 (ns)
プログラムを変更しない場合	8423 (+0)
ログ取得にメモリ格納を用いた場合	8543 (+120)
ログ取得にファイル出力を用いた場合	8991 (+568)
eBPF を用いた場合	9357 (+934)

4. オーバーヘッドの測定結果

測定環境を**表 1** に示す．

本測定では，2.2 節で述べたメッセージブローカを用いたシステムで 100,000 パケットを送受信した．記録する時刻はメッセージブローカがパケットを送受信した時刻すべてである．測定手法の違いによるオーバーヘッドの比較結果を**表 2** に示す．

結果より，各場合におけるオーバーヘッドは，メモリ格納を用いた場合が 120 ns，ファイル出力を用いた場合が 568 ns， eBPF を用いた場合オーバーヘッドが 934 ns となった．つまり， eBPF を用いた場合，メモリ格納を用いた場合の約 7.7 倍，ファイル出力を用いた場合の約 1.6 倍の遅延が発生するものの，1 パケットの処理に対して 1 μs 程度であることが分かった．このため，以上の遅延を許容できる測定において eBPF は有効であるといえる．

5. おわりに

本稿では，eBPF を用いたログ取得オーバーヘッドを評価し，他の手法と比較した．eBPF によるパケットトレーシングにおいて，1 パケットあたり 1μs 程度のオーバヘッドが生じることが分かった．

謝辞　本研究の一部は，JSPS KAKENHI 21K11830 による．

参考文献

[1] Gregg, B.: *BPF Performance Tools*, Addison-Wesley Professional (2019).
[2] Steven McCanne and Van Jacobson: The BSD Packet Filter:A New Architecture for User-level Packet Capture, Lawrence Berkeley Laborator (online), available from ⟨http://www.tcpdump.org/papers/bpf-usenix93.pdf⟩ (accessed 2022-06-23).