Tests *bpf_ktime_get_ns()*

A cosa servono i test:

I test servono a verificare che la funzione `_bpf_ktime_get_ns()` restituisca il tempo corretto in nanosecondi. Per avere un'idea di quanto sia preciso il tempo restituito, si è deciso di confrontarlo con il tempo restituito da bpf-stats che espone:

`run_time_ns`: il tempo totale di esecuzione del programma, per ogni run.

`run_cnt`: il numero totale di run eseguite dall'inizio dell'osservazione.

Su che software sono stati eseguiti i test:

I test sono stati eseguiti su un semplice programma XDP che lavora esclusivamente su pacchetti ICMP ed esegue un numero `n` di volte la checksum. Quest'ultima viene eseguita foldando il risultato ritornato dalla funzione `bpf_csum_diff()` che esegue una differenza di checksum tra due aree di memoria.

Che cos'è *bpf_ktime_get_ns()*:

La funzione `bpf_ktime_get_ns()` è stata usata per misurare il tempo totale di esecuzione del programma XDP. Ritorna il tempo in nanosecondi dall'avvio del sistema (non considera la sospensione). Usa un clock di tipo `CLOCK_MONOTONIC` che è un clock non influenzato da cambiamenti di tempo.

Come è stata utilizzata *bpf_ktime_get_ns()*:

Per misurare il tempo totale di esecuzione, la funzione viene chiamata all'inizio e alla fine del programma XDP. Il risultato viene poi salvato in una mappa BPF.

Come sono stati eseguiti i test:

I test sono stati eseguiti tramite uno script bash che funziona in questo modo:

1. Avvia lo user space program che carica e attacca il programma XDP.
2. Attende che il programma XDP sia pronto e invia un solo pacchetto ICMP.
3. A questo punto il programma XDP è già stato eseguito e ha salvato il suo tempo di esecuzione in una mappa BPF.
4. Recupera il tempo di esecuzione secondo `run_time_ns` di bpf-stats accedendo ai dati tramite `bpftool`. Attenzione, non viene rieseguito il programma XDP, i tempi secondo bpf-stats si riferiscono alla run del punto 3.
5. Termina lo user space che in fase di chiusura effettua un cleanup, quindi rimuove il programma XDP.
6. Salva i tempi di esecuzione in un CSV.
7. Ricomincia dal punto 1 per un numero di volte definito.

Quante prove sono state effettuate:

Lo script è stato eseguito 4 volte, per una durata di 1000 run per ogni esecuzione. Ogni run invia un solo pacchetto ICMP e recupera i tempi di esecuzione secondo `run_time_ns` di bpf-stats e `bpf_ktime_get_ns()`.

Queste 4 esecuzioni sono state eseguite su 4 versioni differenti del programma XDP, ognuna con un numero di checksum da eseguire differente.

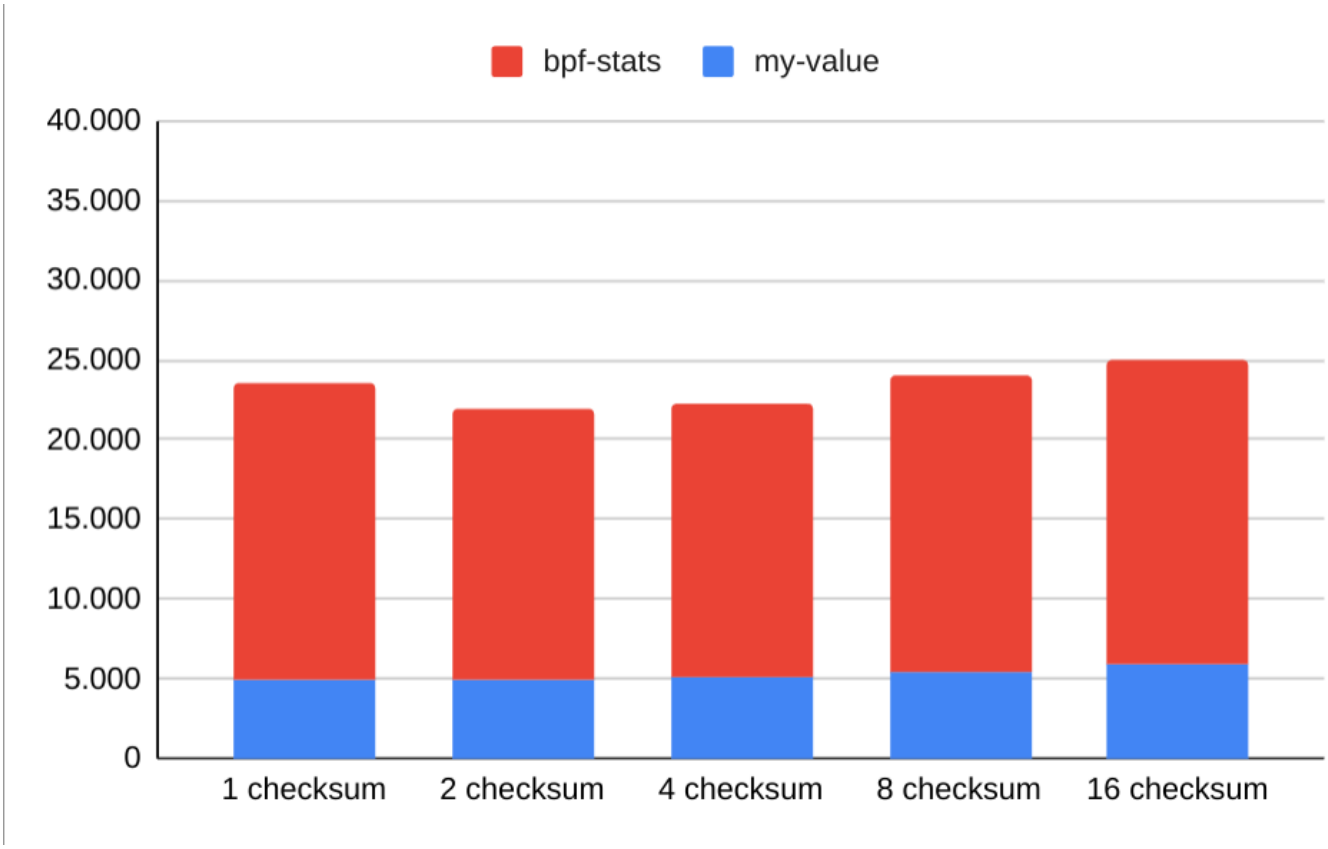
Più precisamente, il numero di checksum eseguite per ogni run è stato 1, 2, 4, 8, 16.

Risultati:

I risultati sono stati analizzati osservando l'andamento dei tempi per ognuno dei 4 test da 1000 run, e confrontando i tempi di esecuzione secondo `run_time_ns` di bpf-stats e `bpf_ktime_get_ns()`, tra le varie versioni del programma XDP con numeri di checksum differenti.

bpf_ktime_get_ns() vs bpf-stats

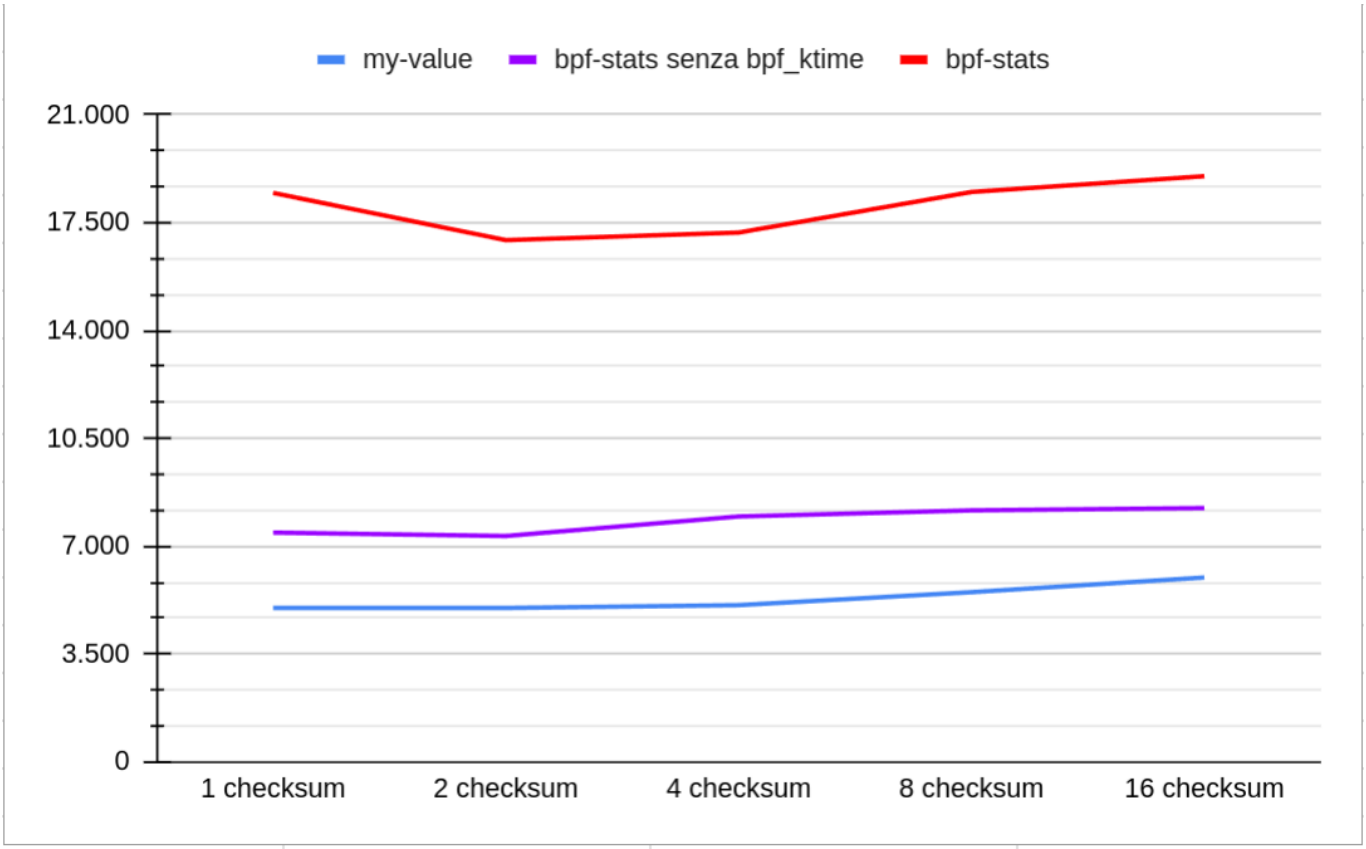
Di seguito vediamo il tempo medio di esecuzione dei 4 test per ogni versione del programma XDP, confrontato con il tempo di esecuzione secondo `run_time_ns` di bpf-stats.



Da una analisi visiva dei dati, si può notare che i tempi di esecuzione secondo `run_time_ns` di bpf-stats e `bpf_ktime_get_ns()` non sono per niente simili, anche l'andamento dei tempi è un po' diverso. I valori secondo `run_time_ns` sono molto più alti rispetto a quelli di `bpf_ktime_get_ns()`. C'è comunque da considerare l'overhead aggiunto dalla funzione `bpf_ktime_get_ns()` e dalla mappa BPF.

Quanto overhead aggiunge bpf_ktime_get_ns()?

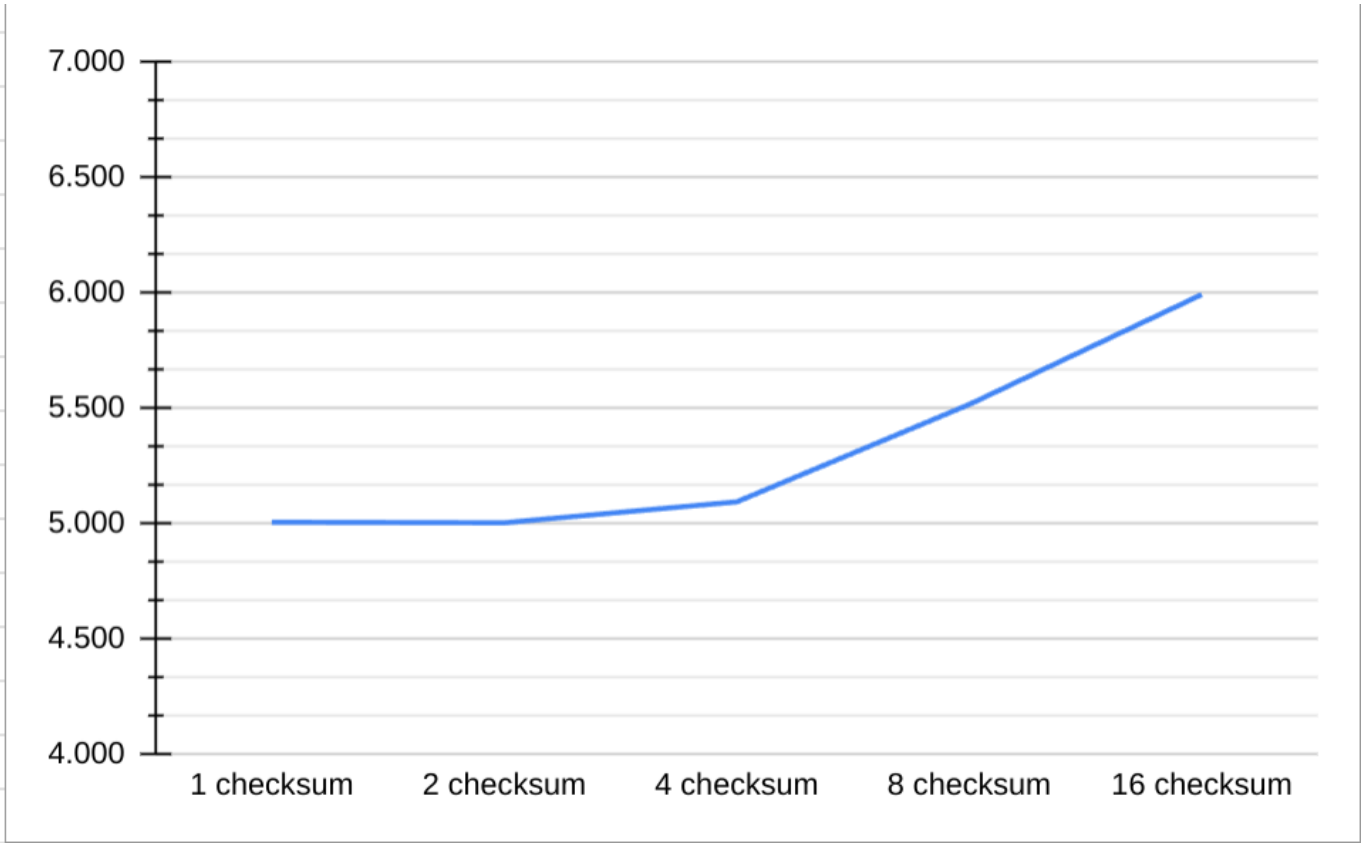
Facendo una media dei tempi, e confrontandoli con una versione del programma senza `bpf_ktime_get_ns()`, rimane comunque una differenza di circa 2000 ns.



Dal grafico si può notare come la linea viola, che rappresenta i tempi secondo `run_time_ns` senza `bpf_ktime_get_ns()`, sia nettamente più vicina ai risultati ottenuti da `bpf_ktime_get_ns()` rispetto a quelli ottenuti da `run_time_ns` con `bpf_ktime_get_ns()`.

Andamento dei tempi di bpf_ktime_get_ns()

Analizzando invece l'andamento di `bpf_ktime_get_ns()` si può notare che i tempi di esecuzione sembrano essere influenzati dal numero di checksum eseguite.



Anche in questo grafico il valore mostrato è la media della media di ognuno dei 4 test da 1000 run per ogni versione del programma XDP.

Conclusioni:

I test hanno mostrato una differenza significativa tra i tempi di esecuzione secondo `run_time_ns` di bpf-stats e `bpf_ktime_get_ns()`.

Overhead aggiunto

Non escludo che questo sia dovuto all'overhead aggiunto dalla funzione `bpf_ktime_get_ns()` e dalla mappa BPF, come abbiamo osservato durante i test senza bpf_ktime_get_ns, la differenza tra i tempi di esecuzione senza `bpf_ktime_get_ns()` calano di molto per `run_time_ns`, ma rimane comunque una differenza in media di circa 2000 ns rispetto `bpf__bpf_ktime_get_ns()`.

Forse questi 2000 ns di differenza si potrebbero giustificare sovvermandosi un attimo a studiare il peso del blocco di instruzioni che fa partire il tracing, precedente a `bpf_ktime_get_ns()`.

```
    struct perf_trace_event __event = {};
    __event.timestamp = bpf_ktime_get_ns();
    __event.bytes = 0;
    __event.processing_time_ns = 0;
```

Cioè, l'istruzione che instanzia la struttura `perf_trace_event` e la istruzione che setta il timestamp, potrebbero aggiungere un overhead di circa 2000 ns. Forse è un po' troppo eccessivo, non saprei.

Un'altra osservazione da fare è come variano i tempi secondo `run_time_ns` con e senza `bpf_ktime_get_ns()`. Sembrano dimostrare che la chiamata a `bpf_ktime_get_ns()` e la mappa BPF aggiungano un overhead non indifferente.

Andamento dei tempi di bpf_ktime_get_ns()

L'andamento dei tempi di `bpf_ktime_get_ns()` sembra essere influenzato dal numero di checksum eseguite. Questo è il comportamento che ci si aspettava, in quanto più checksum si eseguono, più tempo ci si aspetta di impiegare.

Valenza dei test

Non sono sicuro che questi test siano stati eseguiti nel massimo della correttezza, ma mi hanno permesso di fare delle osservazioni su due approcci differenti per misurare il tempo di esecuzione di un programma XDP.