InXpect: Lightweight XDP Profiling

Vladimiro Paschali
University of Rome - Sapienza

Andrea Monterubbiano
CINECA

Francesco Fazzari
University of Rome - Sapienza

Michael Swift
University of Wisconsin - Madison

Salvatore Pontarelli
University of Rome - Sapienza

InXpect: Lightweight XDP Profiling

Vladimiro Paschali
University of Rome - Sapienza

Andrea Monterubbiano
CINECA

Francesco Fazzari
University of Rome - Sapienza

Michael Swift
University of Wisconsin - Madison

Salvatore Pontarelli
University of Rome - Sapienza

TOM BARBETTE

FULLY-FUNDED 2 YEARS POST-DOC POSITION ON PROGRAMMABLE NETWORKS TO ENABLE DIRECT DEVICES-TO-DEVICES COMMUNICATION

InXpect: Lightweight XDP Profiling

Vladimiro Paschali
University of Rome - Sapienza

Andrea Monterubbiano
CINECA

Francesco Fazzari
University of Rome - Sapienza

Michael Swift
University of Wisconsin - Madison

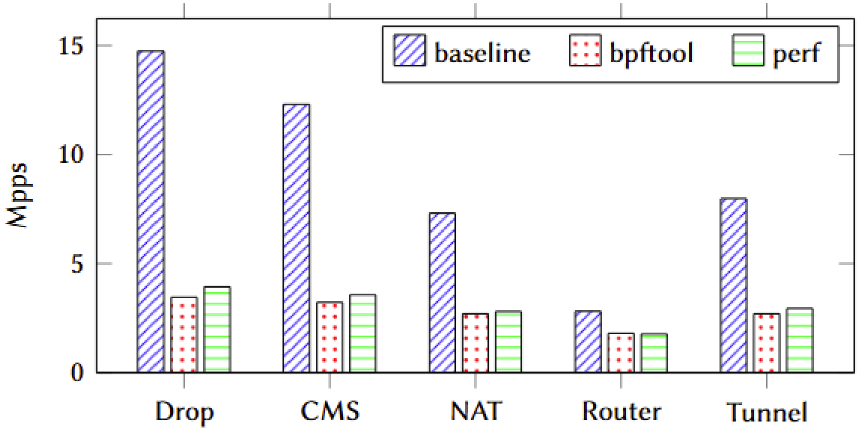
Salvatore Pontarelli
University of Rome - Sapienza

XDP Performance Monitoring

Linux `perf` and `bpftool` are among the most widely used profilers for **analysing eBPF and XDP applications**, but they introduce **excessive overhead** that degrades performance.

Profilers **can reduce throughput by up to 4×**

As a result, profiling **fast network functions** becomes more challenging.

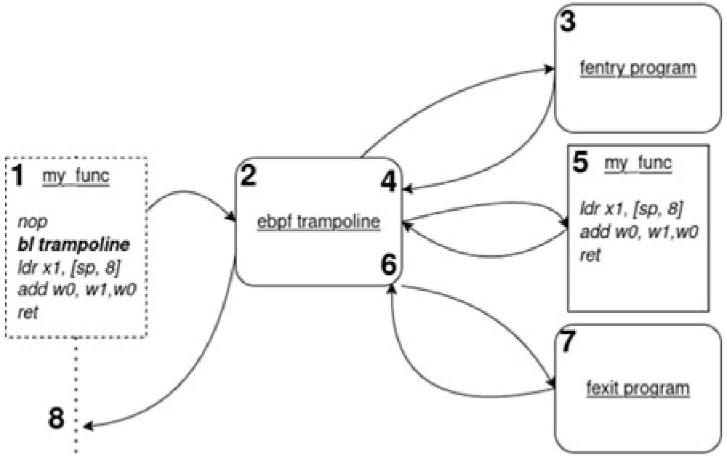


Profiling XDP Applications

Profilers rely on specialized **hardware registers (Performance Monitoring Counters, PMCs)** to track **Hardware events** such as **instructions, cycles, cache hits/misses, and branch misses**.

For eBPF applications, `perf/bpftool` **attach two programs** `fentry()` and `fexit()` around the target application to **read PMC values** and save the monitored metric value.

Although simple, these scheme **introduces a significant overhead**.



Alexis Lothoré. 2025. Bouncing on trampolines to run eBPF programs

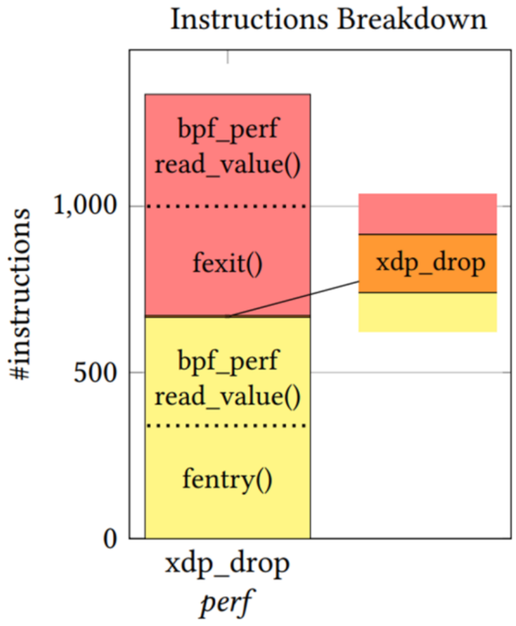
Profiling Overhead

In **eBPF profiling**, `perf` introduces over **600 additional instructions** for each `fentry()/fexit()` function.

This overhead is caused by both invoking the `fentry()/fexit()` and the `bpf_perf_event_read_value()` function used to **access PMCs**.

As a result, both **throughput and profiling accuracy are disrupted**.

For example, `perf` reports that a simple **XDP_DROP** application **executes 627 instructions**, when it only **executes 2**.




```
0: (b7) r0 = 1
1: (95) exit
```

XDP DROP xlated output

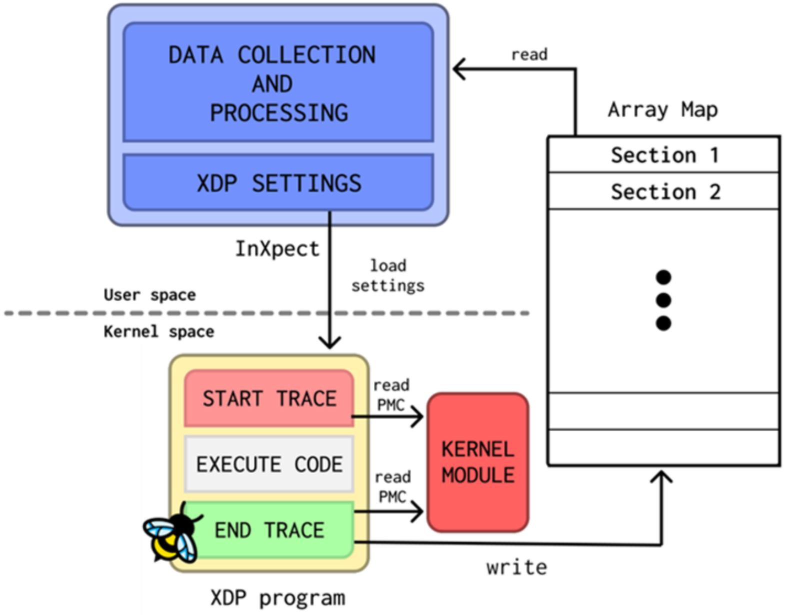
InXpect

To address these issues, we developed ***InXpect***, a **lightweight XDP profiler**.

- **User-space component**
  configures which **CPU events** to record

- **Tracing macros**
  delimit **profiling sections**

- **Kernel module**
  reads PMC values efficiently



InXpect Macros

Programmers **delimit profiling sections** with two macros: **START_TRACE** and **END_TRACE**.

- **START_TRACE**
  reads PMC values, **stores** them. Also manages **region activation** and **sampling rate**.

- **END_TRACE**
  reads PMC again, computes the **difference**, and **stores the result in an eBPF map.**

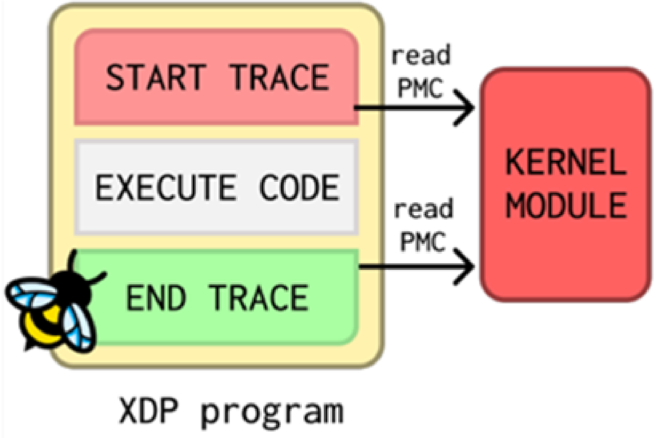
```
SEC("xdp")
int parsing(struct xdp_md *ctx)
{
    START_TRACE(<SectionName>);
    [...]

    END_TRACE(<SectionName>);
    return XDP_DROP;
}
```

INXPECT instrumentation sample code

InXpect Kernel Module

eBPF ISA does not allow the use **of native CPU instructions** such as **rdpmc**, which are **needed to read PMC values**.

To **overcome** this limitation, we developed a **Linux kernel Module** that exposes an **eBPF kfunc** to invoke **rdpmc**.

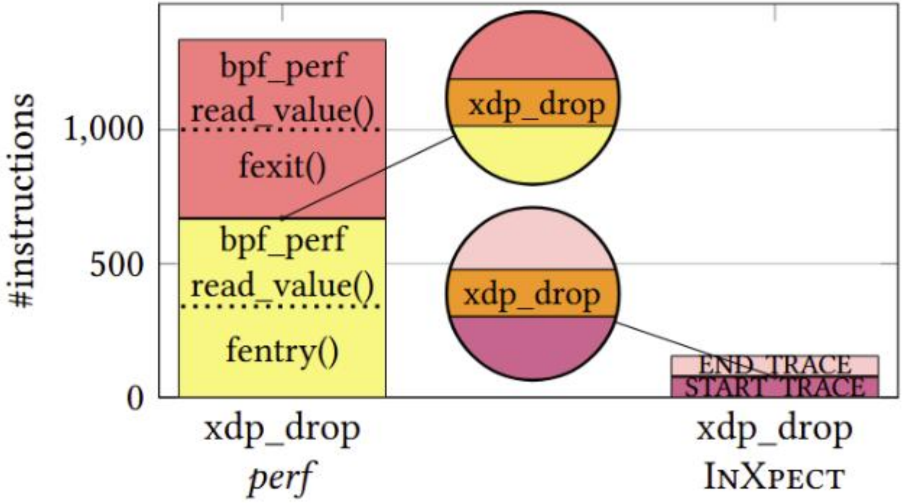


Reducing `perf` Overhead

Optimizations in **InXpect**:

- **Macros**:
  Using `kfuncs` inside eBPF removes the `fentry` and `fexit`, **saving ~200 instructions per call**

- **Direct PMC access**:
  Using the **native rdpmc x86 instruction**, instead of indirect access like `perf`, **saving ~280 instructions per read**



XDP Applications used for evaluation

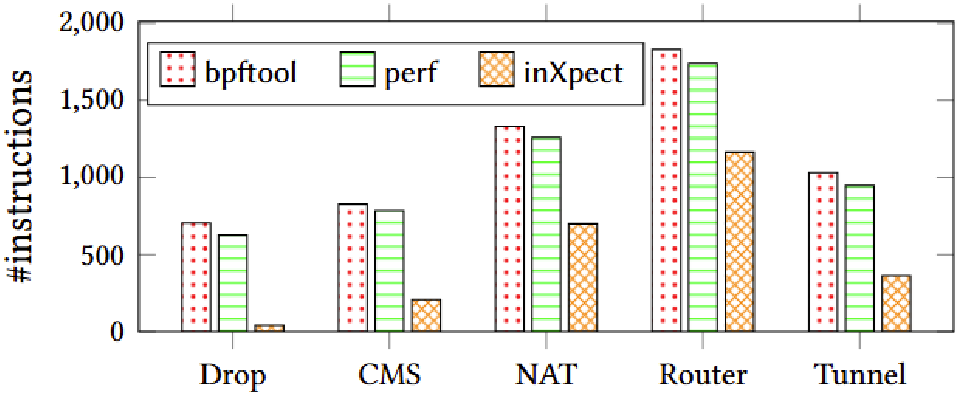
These are the **applications** used to **evaluate the profilers**.

For each one, we report the corresponding **action performed by the Linux kernel on every packet**.

Program	Description	Action
Drop	Drops each packet.	XDP_DROP
CMS	Count Min Sketch 4 by 2^{13}.	XDP_DROP
NAT	Network Address Translation.	XDP_TX
Router	IP Look up in a routing table.	XDP_TX
Tunnel	IP header encapsulation.	XDP_TX

Instruction Count Inaccuracy

- Profilers **add their own instructions** to the measured count
- Extra instructions come from code **executed before the second PMC read**
- **More complex** profilers → **more noise** in the results



Case study: Drop

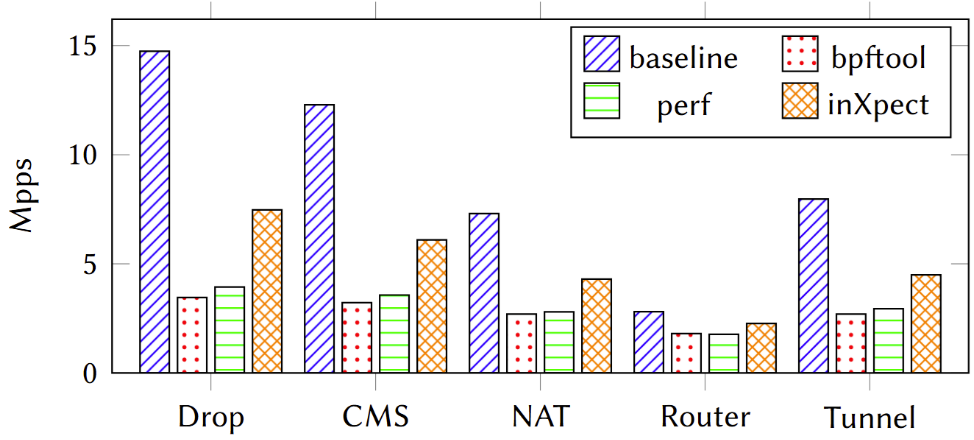
Expected instruction count: ~2
`perf`: ~600 instructions
InXpect: ~40 instructions

Throughput

InXpect is faster than `perf` **and** `bpftool` but still **halves the performance** of most applications when profiling every packet.

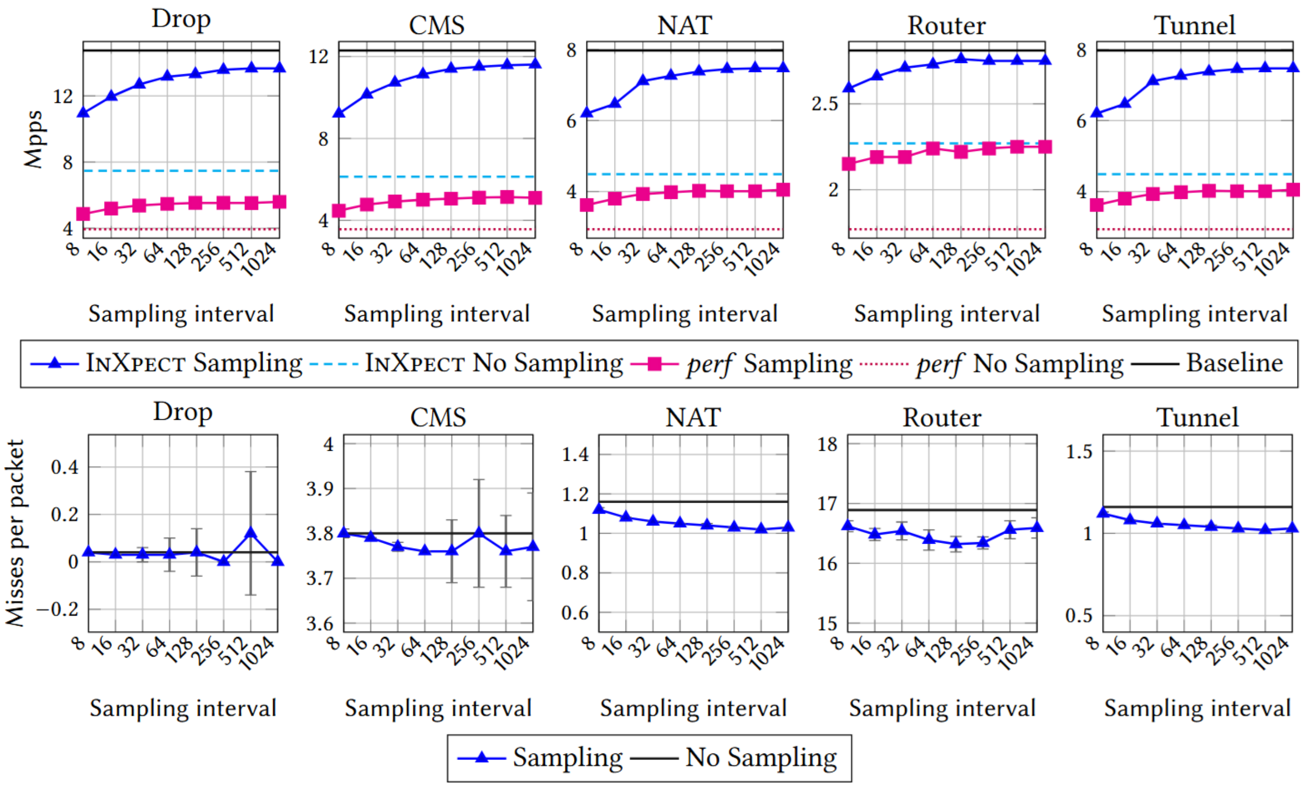
The **worst** performance degradation occurs in the `Drop` application, since the **relative weight of the profiler is proportionally high**.

We implemented a **sampling mechanism** that significantly **improves performance**.



Sampling

- Sampling **reduces profiling overhead**
- Works best when triggering **samples is lightweight**
- Excessive sampling **lowers profiling accuracy**



Recap

We identified the **main sources of overhead** in standard **XDP profilers**

- `fentry()`
- `fexit()`
- `bpf_perf_read_value()`

Then, we developed **InXpect**, a **lightweight** profiling framework for XDP applications

- **Removed** main sources of **profiler overhead**
- Used **kfunc** to directly and **efficienly access PMCs**
- Implemented a **sampling** functionality to **further reduce overhead**

InXpect vs. **standard profilers** on XDP applications

- **71% faster** than `perf`
- **122% faster** than `perf` (**with sampling**)
- **73% less instruction noise**