



BPF ksock: bringing network sockets to BPF

Mahé Tardy, Kornilios Kourtis



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026

Why you might care about this

Initial use-cases:

- ❖ **logging/alerts** and **telemetry events** from BPF

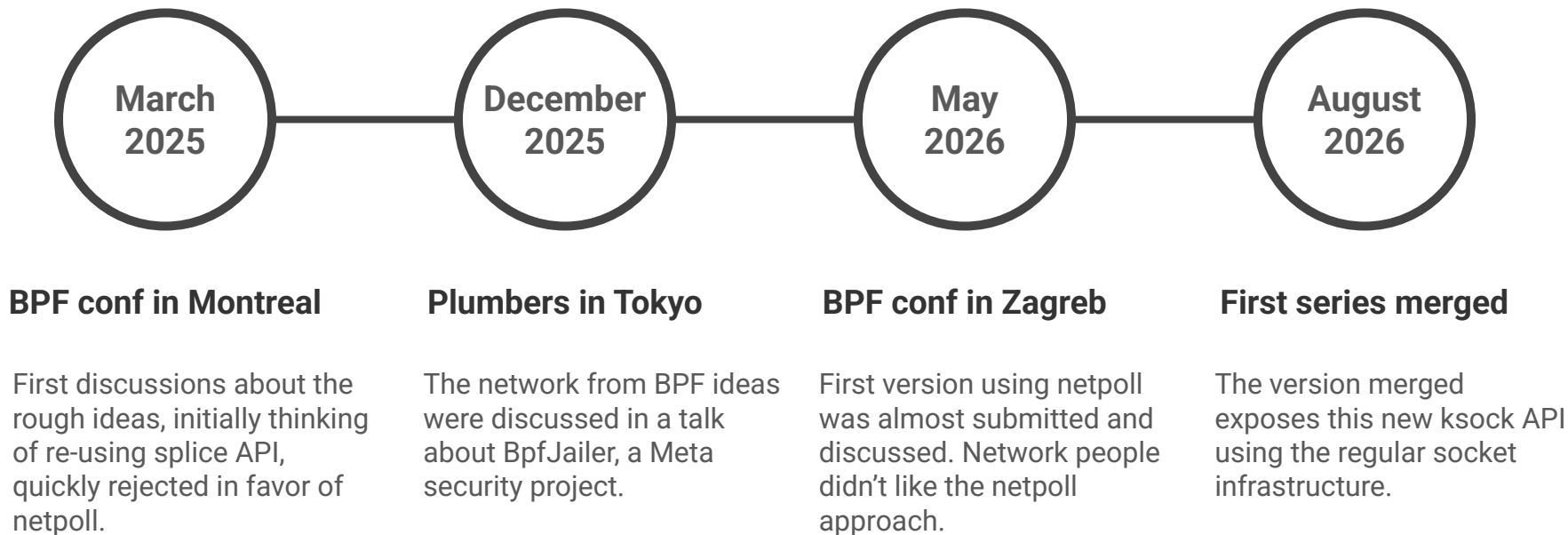
New use-cases:

- ❖ **heartbeats** (health probes, BFD heartbeat for BGP)
- ❖ VM migration (Zussman et al., *It's the Kernel's Fault!*, 2026)
- ❖ In kernel servers (DNS servers)

Benefit is to reduce overhead (memory, CPU) and reduce the attack surface.

Eventual goal: **agentless**: no trusted user-space component.

Timeline



The result so far: the bpf_ksock API

The API consists of five kfuncs:

- ❖ `bpf_ksock_create()`
- ❖ `bpf_ksock_connect()`
- ❖ `bpf_ksock_send()`
- ❖ `bpf_ksock_acquire()`
- ❖ `bpf_ksock_release()`

The result so far: the bpf_ksock API

The API consists of five kfuncs:

- ❖ `bpf_ksock_create()` ← only UDP
- ❖ `bpf_ksock_connect()`
- ❖ `bpf_ksock_send()`
- ❖ `bpf_ksock_acquire()`
- ❖ `bpf_ksock_release()`

The result so far: the bpf_ksock API

The API consists of five kfuncs:

- ❖ `bpf_ksock_create()` 🤔 ← only UDP
- ❖ `bpf_ksock_connect()` 🤔
- ❖ `bpf_ksock_send()` 🤔
- ❖ `bpf_ksock_acquire()`
- ❖ `bpf_ksock_release()`

The result so far: the bpf_ksock API

The API consists of five kfuncs:

❖ `bpf_ksock_create()` 🤔 ← only UDP

❖ `bpf_ksock_connect()` 🤔

❖ `bpf_ksock_send()` 🤔

❖ `bpf_ksock_acquire()`

❖ `bpf_ksock_release()`



SYSCALL progs

SYSCALL & LSM progs

How to setup ksock

```
SEC("syscall")
int ksock_setup(void *ctx)
{
    struct bpf_ksock_create_opts create_opts = {};
    union bpf_ksock_addr addr = {};
    struct bpf_ksock *ks;
    int err = 0;

    create_opts.family = AF_INET;
    create_opts.type = SOCK_DGRAM;
    create_opts.protocol = IPPROTO_UDP;

    ks = bpf_ksock_create(&create_opts, sizeof(create_opts), &err);
    if (!ks)
        return err;

    addr.sin.sin_family = AF_INET;
    addr.sin.sin_port = bpf_htons(remote_port);
    addr.sin.sin_addr.s_addr = ipv4_remote;

    err = bpf_ksock_connect(ks, &addr, sizeof(addr));
    if (err) {
        bpf_ksock_release(ks);
        return err;
    }

    err = ksock_ctx_insert(ks);
    if (err && err != -EEXIST)
        return err;
    return 0;
}
```

How to setup ksock: create

```
SEC("syscall")
int ksock_setup(void *ctx)
{
    struct bpf_ksock_create_opts create_opts = {};
    union bpf_ksock_addr addr = {};
    struct bpf_ksock *ks;
    int err = 0;

    create_opts.family = AF_INET;
    create_opts.type = SOCK_DGRAM;
    create_opts.protocol = IPPROTO_UDP;

    ks = bpf_ksock_create(&create_opts, sizeof(create_opts), &err);
    if (!ks)
        return err;

    addr.sin.sin_family = AF_INET;
    addr.sin.sin_port = bpf_htons(remote_port);
    addr.sin.sin_addr.s_addr = ipv4_remote;

    err = bpf_ksock_connect(ks, &addr, sizeof(addr));
    if (err) {
        bpf_ksock_release(ks);
        return err;
    }

    err = ksock_ctx_insert(ks);
    if (err && err != -EEXIST)
        return err;
    return 0;
}
```

How to setup ksock: connect

```
SEC("syscall")
int ksock_setup(void *ctx)
{
    struct bpf_ksock_create_opts create_opts = {};
    union bpf_ksock_addr addr = {};
    struct bpf_ksock *ks;
    int err = 0;

    create_opts.family = AF_INET;
    create_opts.type = SOCK_DGRAM;
    create_opts.protocol = IPPROTO_UDP;

    ks = bpf_ksock_create(&create_opts, sizeof(create_opts), &err);
    if (!ks)
        return err;

    addr.sin.sin_family = AF_INET;
    addr.sin.sin_port = bpf_htons(remote_port);
    addr.sin.sin_addr.s_addr = ipv4_remote;

    err = bpf_ksock_connect(ks, &addr, sizeof(addr));
    if (err) {
        bpf_ksock_release(ks);
        return err;
    }

    err = ksock_ctx_insert(ks);
    if (err && err != -EEXIST)
        return err;
    return 0;
}
```

How to send with ksock

```
SEC("lsm.s/socket_bind")
int BPF_PROG(ksock_socket_bind, struct socket *sock,
             struct sockaddr *address, int addrlen, int ret)
{
    struct bpf_ksock *ks;
    u32 pid = bpf_get_current_pid_tgid() >> 32;

    if (ret || pid != target_pid)
        return ret;

    ks = ksock_ctx_get();
    if (!ks) {
        send_ret = -ENOENT;
        return ret;
    }

    send_ret = bpf_ksock_send(ks, send_data, sizeof(send_data));
    bpf_ksock_release(ks);

    return ret;
}
```

How to send with ksock: send

```
SEC("lsm.s/socket_bind")
int BPF_PROG(ksock_socket_bind, struct socket *sock,
             struct sockaddr *address, int addrlen, int ret)
{
    struct bpf_ksock *ks;
    u32 pid = bpf_get_current_pid_tgid() >> 32;

    if (ret || pid != target_pid)
        return ret;

    ks = ksock_ctx_get();
    if (!ks) {
        send_ret = -ENOENT;
        return ret;
    }

    send_ret = bpf_ksock_send(ks, send_data, sizeof(send_data));
    bpf_ksock_release(ks);

    return ret;
}
```

Demo #1

syslogs



A basic example with syslogd

```
SEC("lsm.s/file_open")
int BPF_PROG(file_open, struct file *file)
{
    // [...]

    bpf_d_path(&file->f_path, path, sizeof(path));
    if (bpf_strcmp(path, "/etc/hostname") != 0)
        return 0;

    // [...]

    len = format_syslog_message(syslog_msg,
sizeof(syslog_msg), syslog_hostname, path, comm, pid);
    // [...]

    ks = ksock_ctx_get();
    if (!ks) {
        return 0;
    }

    err = bpf_ksock_send(ks, syslog_msg, len);
    if (err < 0) {
        return 0
    }

    bpf_ksock_release(ks);

    return 0;
}
```

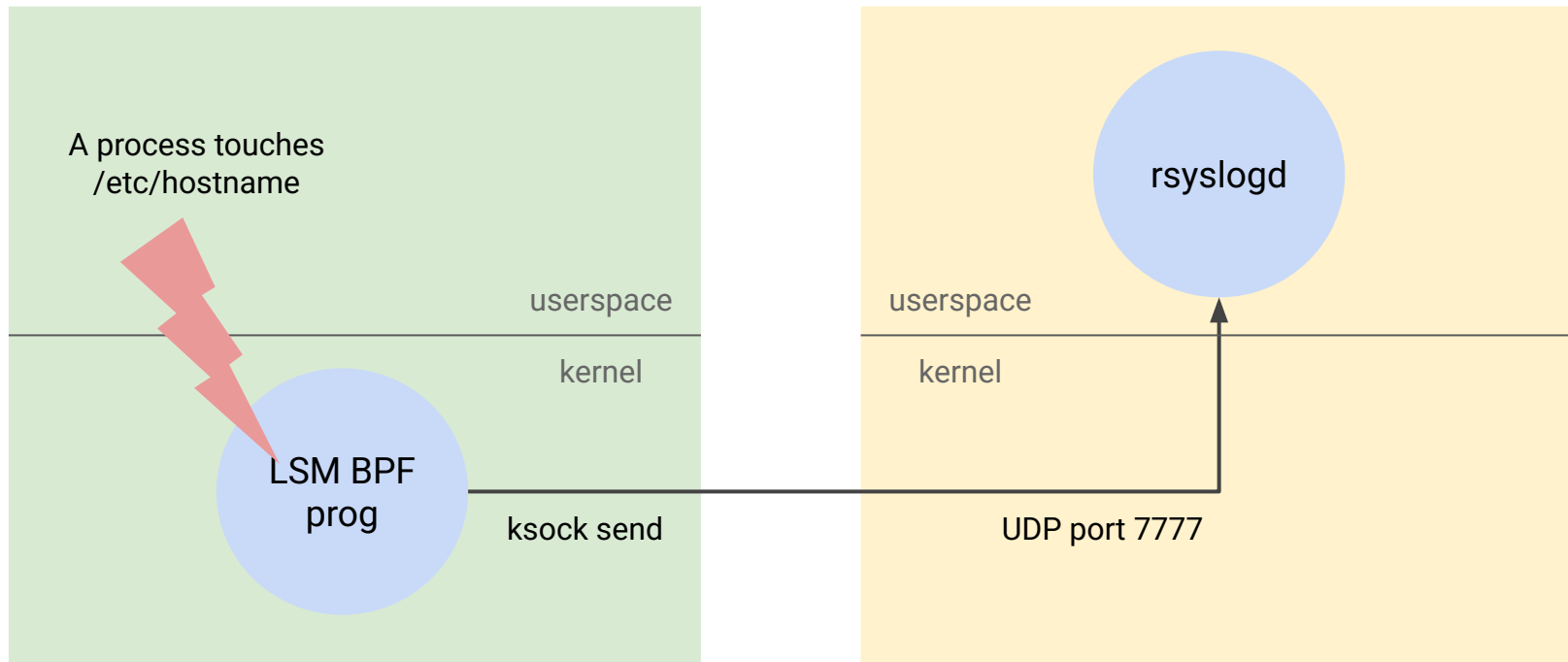
```
static __always_inline int format_syslog_message(
    char *buf, u32 buf_size, const char *hostname,
    const char *path, const char *comm, u32 pid)
{
    __u64 params[4];
    int len;

    // RFC 3164 Format: <PRI>TIMESTAMP HOSTNAME TAG[PID]:
MESSAGE
    params[0] = (u64)hostname;
    params[1] = (u64)pid;
    params[2] = (u64)path;
    params[3] = (u64)comm;

    len = bpf_snprintf(buf, buf_size,
        "<14>%s ksock-bpf[%d]: file_open path=%s comm=%s",
        params, sizeof(params));

    return len;
}
```

A basic example with syslogd

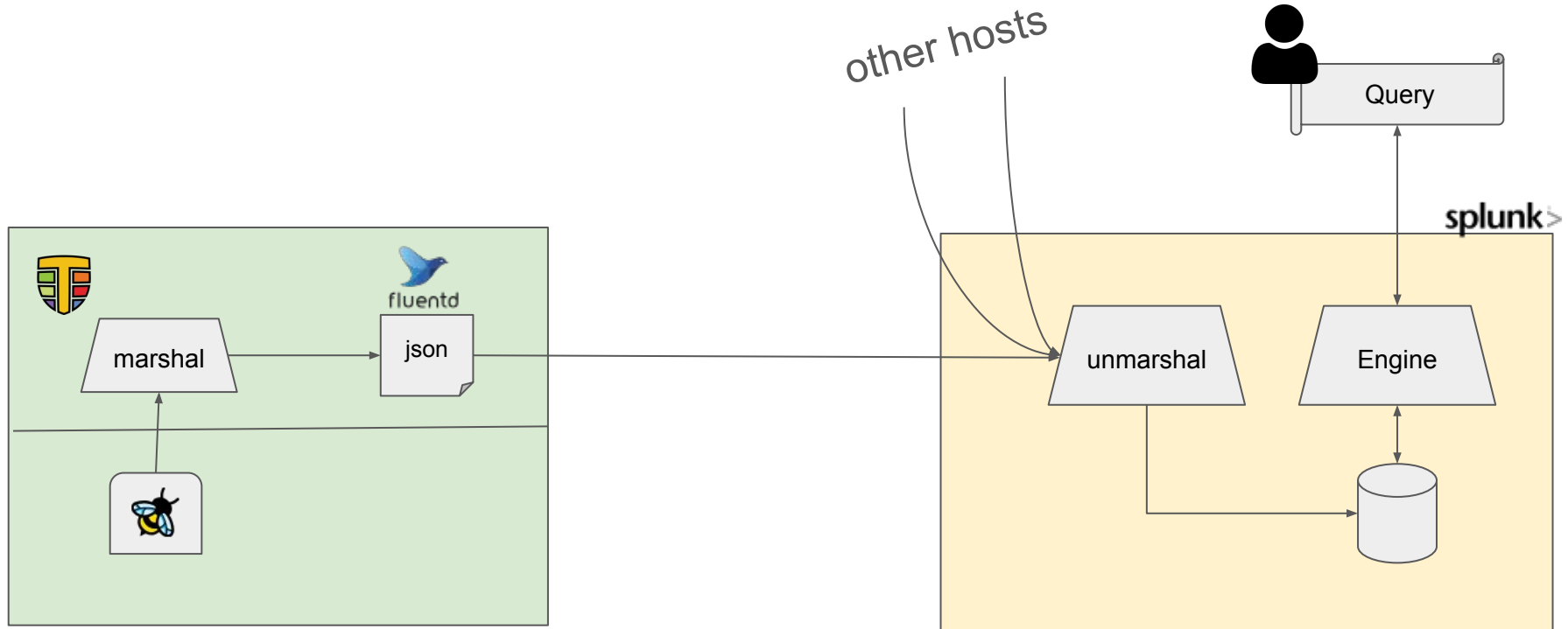


Demo #2

Data analytics on
BPF telemetry data



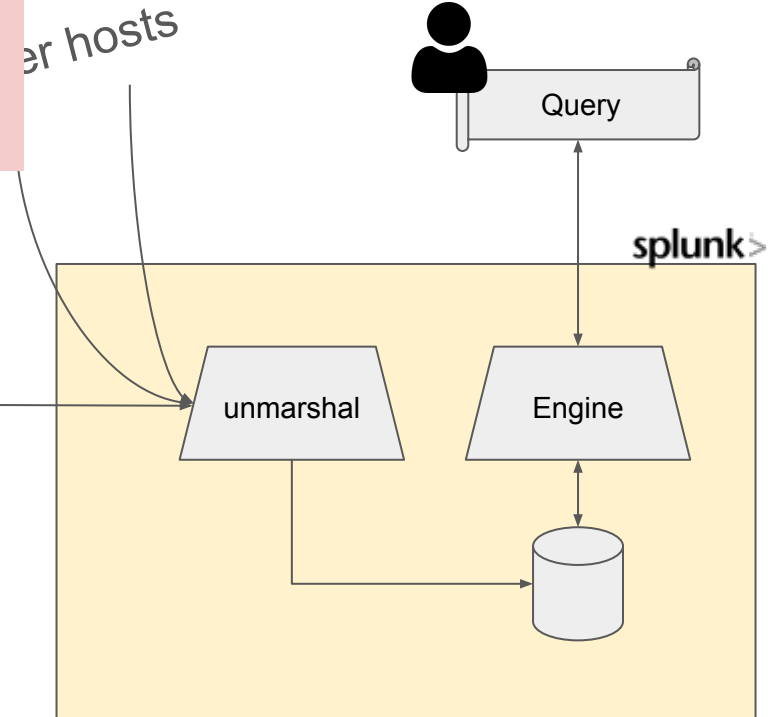
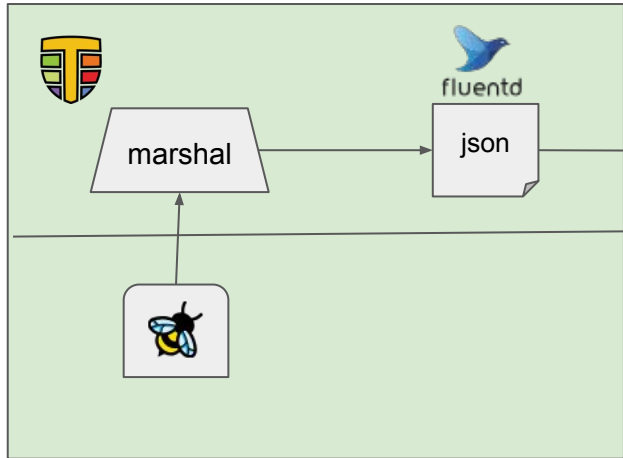
Log Analysis (e.g., investigation on alerts)



Data analytics on BPF telemetry events

Problems:

- inefficient copying / transformations
- lots of opportunities to mess with the data



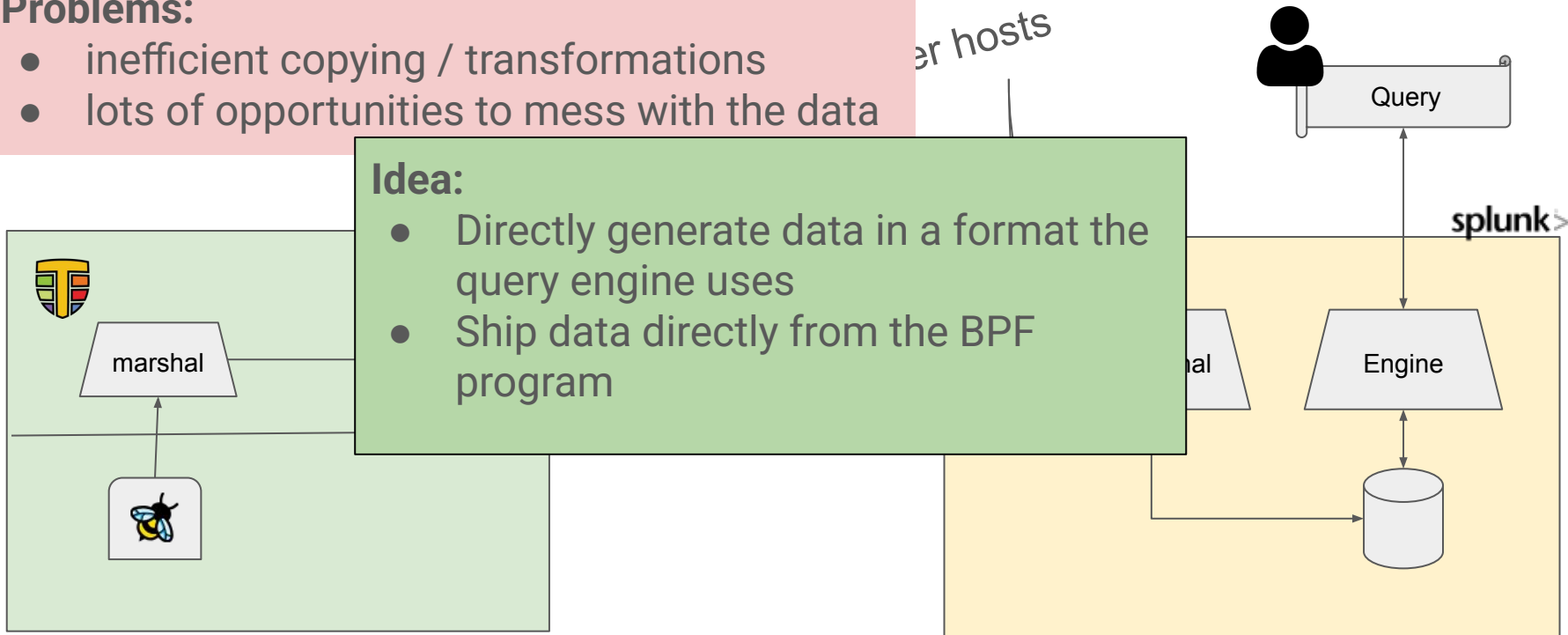
Data analytics on BPF telemetry events

Problems:

- inefficient copying / transformations
- lots of opportunities to mess with the data

Idea:

- Directly generate data in a format the query engine uses
- Ship data directly from the BPF program

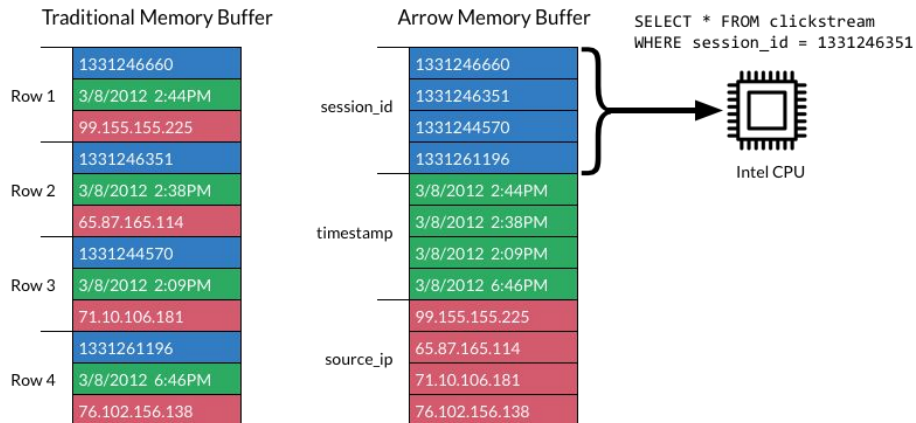


Arrow columnar format

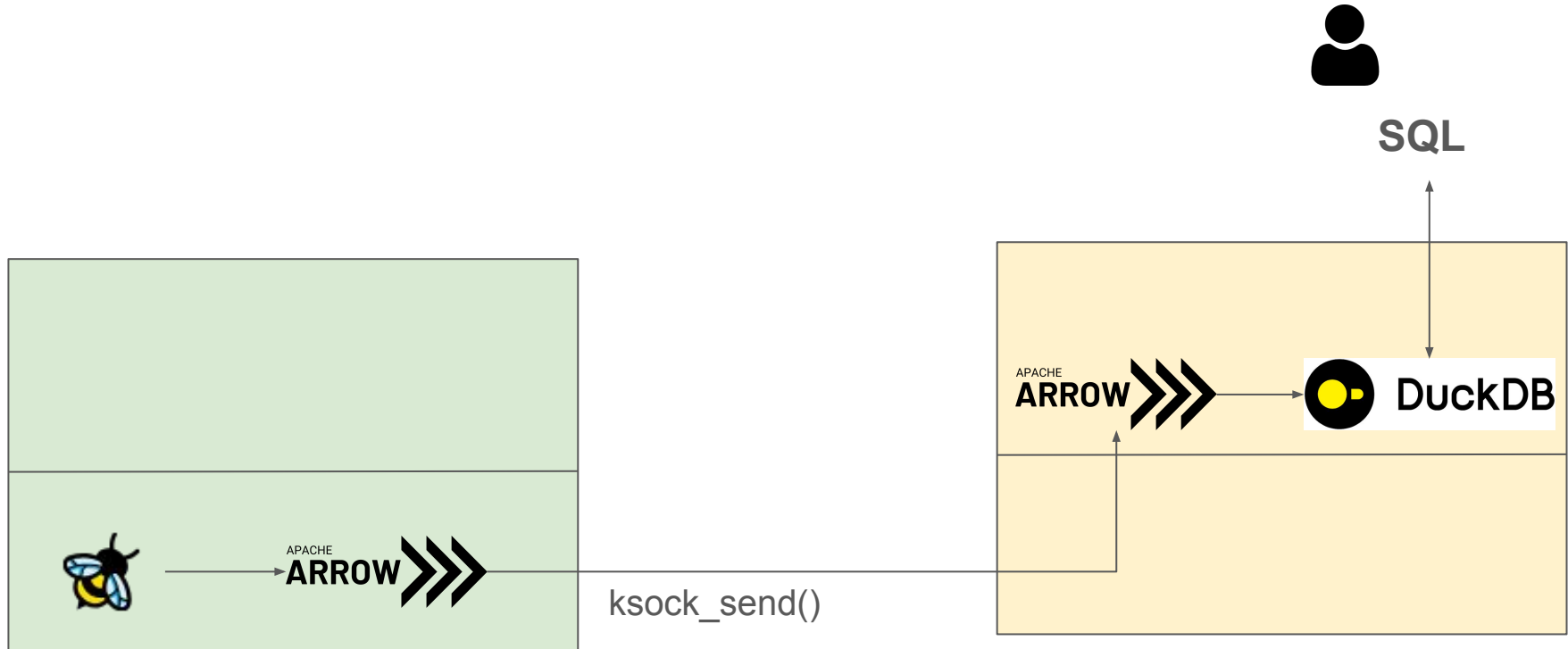


- Analytics engines use columnar formats for efficient queries
- Apache arrow:** common format for multiple engines / databases

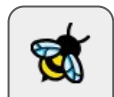
	session_id	timestamp	source_ip
Row 1	1331246660	3/8/2012 2:44PM	99.155.155.225
Row 2	1331246351	3/8/2012 2:38PM	65.87.165.114
Row 3	1331244570	3/8/2012 2:09PM	71.10.106.181
Row 4	1331261196	3/8/2012 6:46PM	76.102.156.138



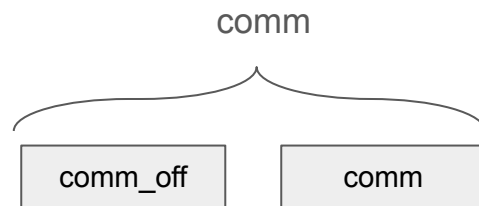
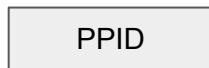
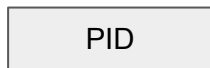
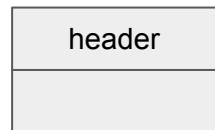
PoC: process list (ps)



PoC: how it works?



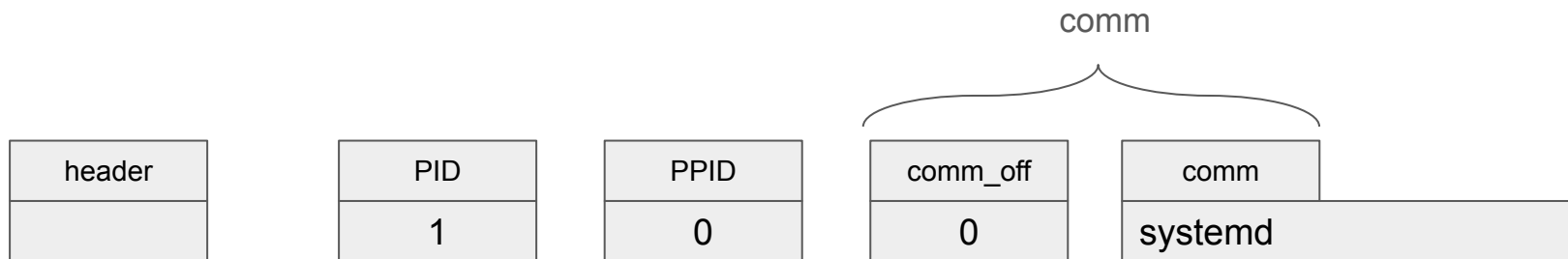
task iterator



PoC: how it works?



task iterator



PoC: how it works?

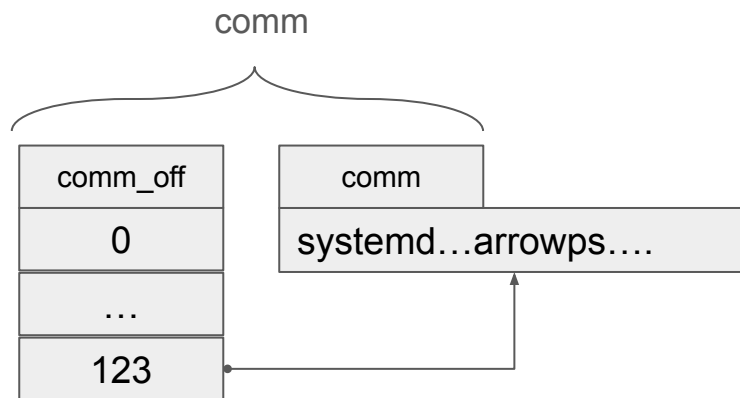


task iterator

header

PID
1
...
1245

PPID
0
...
1111



PoC: how it works?

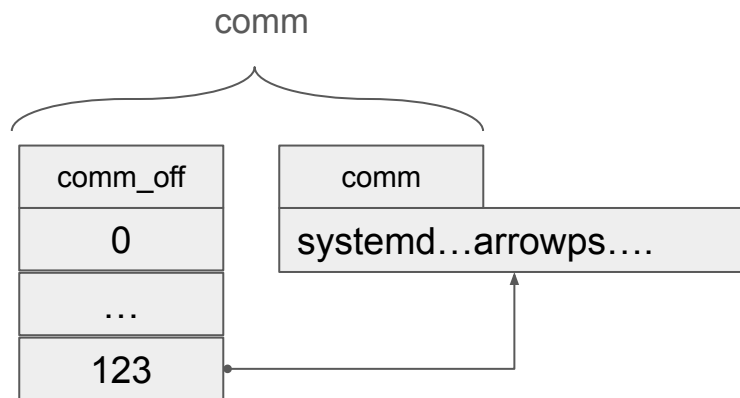


task iterator

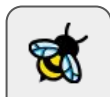
header
20

PID
1
...
1245

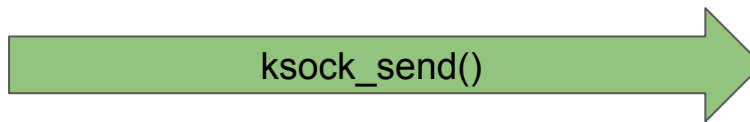
PPID
0
...
1111



PoC: how it works?



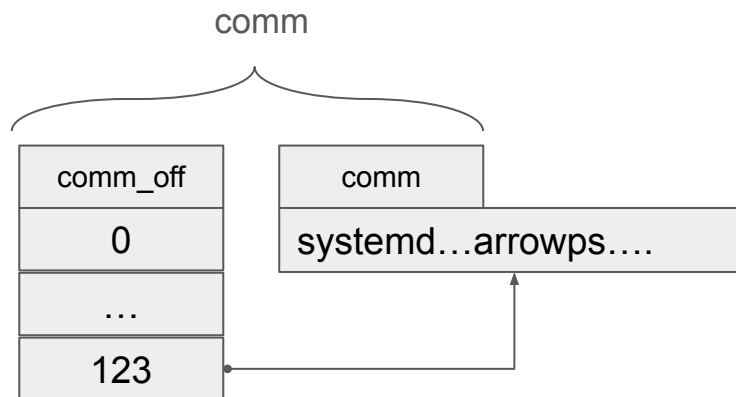
task iterator



header
20

PID
1
...
1245

PPID
0
...
1111



Demo #2

Demo #2

Current limitations:

- Extend ksock to sleepable tracing programs (iterators)
- sendmsg() interface (scatter-gather) to avoid final copies
- TCP support for larger data / reliable delivery



More use cases!

Allow post-copy live VM migration



It's the Kernel's Fault! Custom Page Fault Handling With `bpf_fault`

Tal Zussman
Columbia University

Riju Dey
Columbia University

Hasan Zengin
Columbia University

Yiming Fang
Columbia University

David Hildenbrand
Technical University of Munich and
Arm Germany GmbH

Asaf Cidon
Columbia University

Abstract

Page faults, which occur when a program accesses a virtual memory page that is not mapped to physical memory,

ACM Reference Format:

Tal Zussman, Riju Dey, Hasan Zengin, Yiming Fang, David Hildenbrand, and Asaf Cidon. 2026. It's the Kernel's Fault! Custom Page Fault Handling With `bpf_fault`. In *Symposium on*

[...]

parent's `struct_ops`. This avoids link setup in the fork fast path. If the child later needs a full link (e.g., to register a region), it issues a `BPF_FAULT_CLAIM` command through the `bpf()` syscall, which promotes the lightweight link.

During fork, the kernel duplicates VMAs one at a time. Since a single fault context may span multiple VMAs, this would wrongly allocate a new context for each VMA. A deduplication list solves this: the first VMA encountered for a given context allocates a child context and appends it to

work. Our VM snapshot integration works around the former by streaming pages to a userspace writer (§5.1), but the latter prevents use cases such as post-copy live VM migration, where the handler reads data from the network. These gaps are closing as eBPF evolves. Dynptrs provide access to variable-sized memory [86], and a recent proposal supports sending UDP packets from eBPF, with TCP support under discussion [7]. Supporting socket reads would enable an eBPF handler to implement post-copy migration in-kernel.

Heartbeats, BFD use case

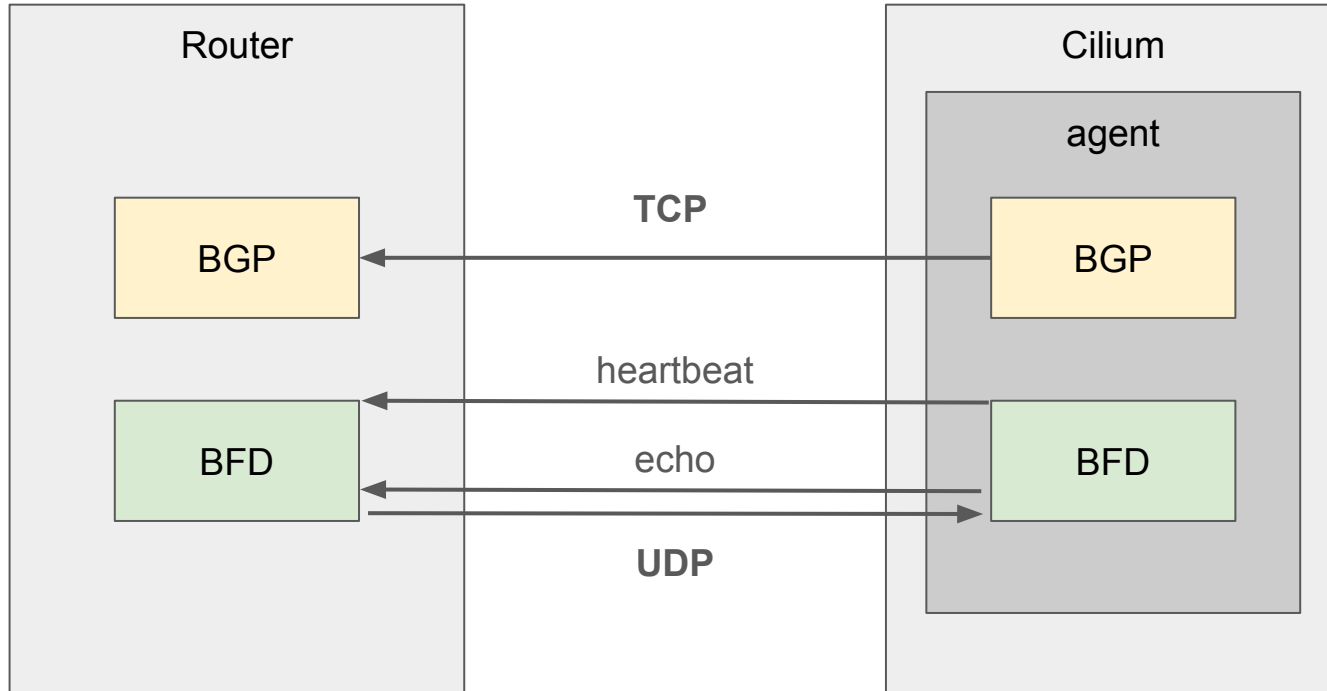


Diagram from a chat with Yutaro Hayakawa

DNS server in BPF, again

Implement simple local DNS server with BPF

11 Dec 2025, 18:00

30m

"Main Hall A" (Toranomom Hills Mori Tower)

Speaker

Raman Shukhau

Description

This talk explores the idea of capturing and identifying DNS requests with BPF and responding to them "in-place" with BPF.

DNS is a relatively simple UDP protocol, and a typical DNS query over UDP usually involves just one packet for the query and one packet for the response. If BPF parses structure of the packet and is able to resolve address from the request, e.g. from the hash map attached to the program, it's also able to rewrite all fields of the incoming packet with the resolved address and place this packet back in the queue as a response.

In the talk we will cover an actual implementation of TC/XDP BPF Programs, potential performance benefits and real world applications, as well as, relevant topics such as DNS DDos protection.

Authors

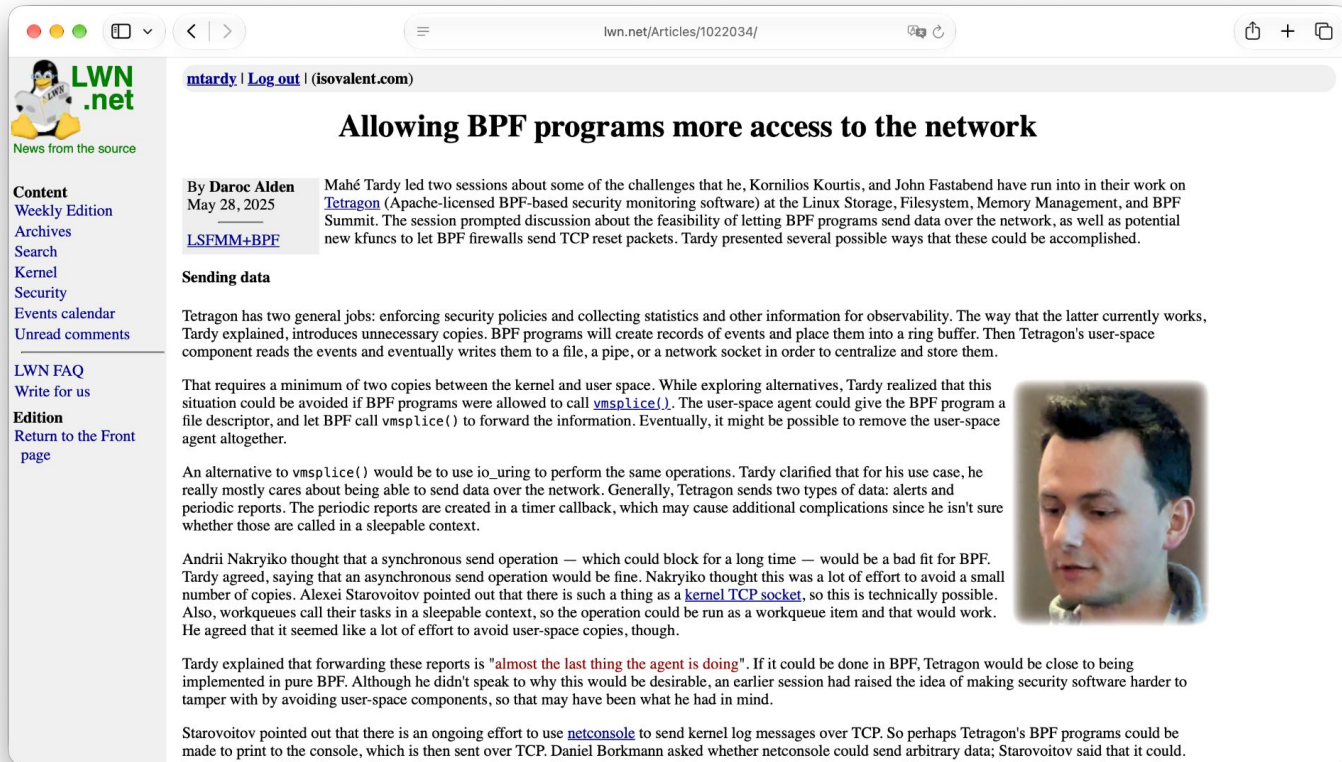
Martin Lau

Raman Shukhau

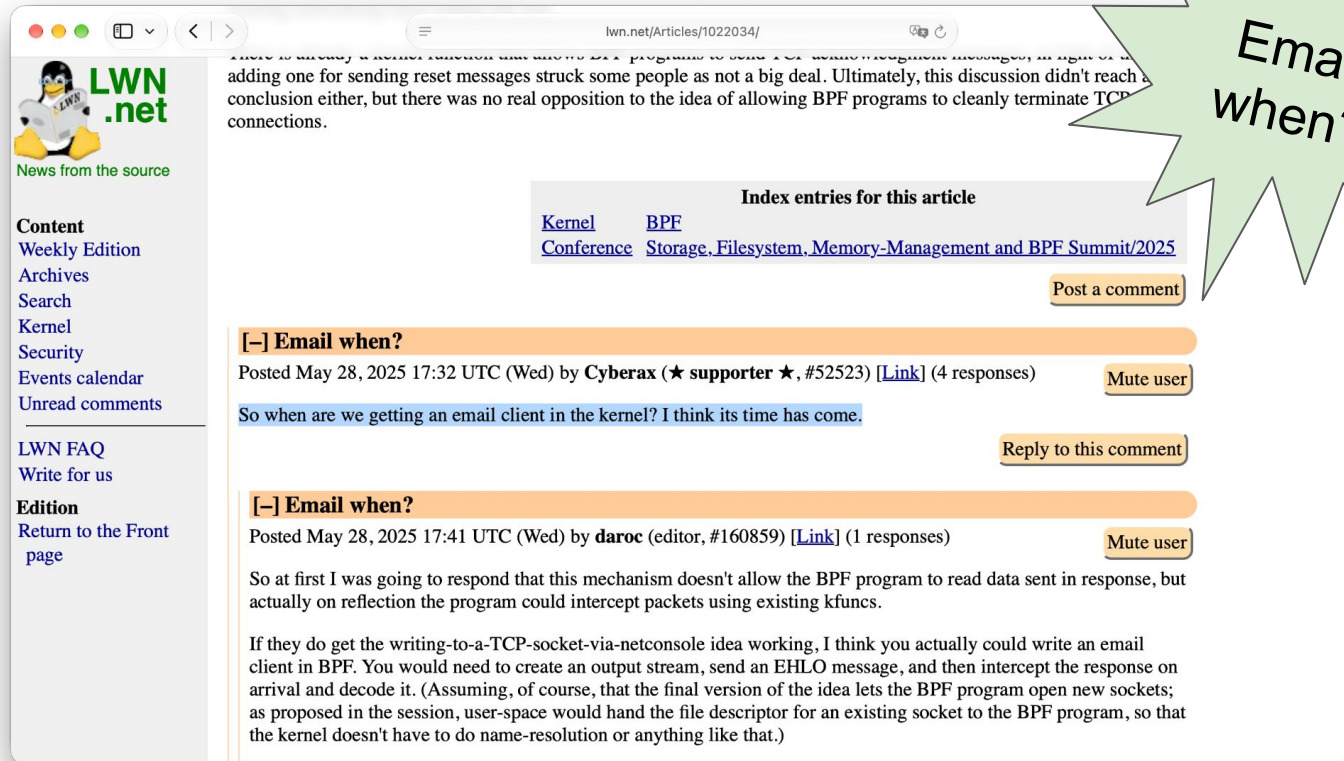
What program types we can use?

- Cgroup Types - **Not Supported**
 - routing decision is made at this point
 - change to SKB structure limited or not allowed
- TC/TCX - **Supported**
 - Can modify SKB and use `bpf_redirect`
- XDP - **Supported**
 - Can change XDP struct and do `XDP_TX`

A well-anticipated example



A well-anticipated example



The screenshot shows a web browser window displaying an LWN.net article. The browser's address bar shows the URL `lwn.net/Articles/1022034/`. The LWN.net logo and navigation links are visible on the left. The article content discusses BPF programs and TCP connections. A green starburst callout with the text "Email when?" is positioned to the right of the article. Below the article, there are two comment sections, each titled "[-] Email when?". The first comment is by "Cyberax" and the second is by "daroc". Both comments discuss the possibility of writing an email client in BPF.

LWN.net
News from the source

Content
[Weekly Edition](#)
[Archives](#)
[Search](#)
[Kernel](#)
[Security](#)
[Events calendar](#)
[Unread comments](#)

LWN FAQ
[Write for us](#)

Edition
[Return to the Front page](#)

There is already a kernel extension that allows BPF programs to send TCP window management messages, in light of adding one for sending reset messages struck some people as not a big deal. Ultimately, this discussion didn't reach a conclusion either, but there was no real opposition to the idea of allowing BPF programs to cleanly terminate TCP connections.

Index entries for this article
[Kernel](#) [BPF](#)
[Conference](#) [Storage](#), [Filesystem](#), [Memory-Management](#) and [BPF Summit/2025](#)

[Post a comment](#)

[-] Email when?
Posted May 28, 2025 17:32 UTC (Wed) by **Cyberax** (★ supporter ★, #52523) [\[Link\]](#) (4 responses) [Mute user](#)

So when are we getting an email client in the kernel? I think its time has come.

[Reply to this comment](#)

[-] Email when?
Posted May 28, 2025 17:41 UTC (Wed) by **daroc** (editor, #160859) [\[Link\]](#) (1 responses) [Mute user](#)

So at first I was going to respond that this mechanism doesn't allow the BPF program to read data sent in response, but actually on reflection the program could intercept packets using existing kfuncs.

If they do get the writing-to-a-TCP-socket-via-netconsole idea working, I think you actually could write an email client in BPF. You would need to create an output stream, send an EHLO message, and then intercept the response on arrival and decode it. (Assuming, of course, that the final version of the idea lets the BPF program open new sockets; as proposed in the session, user-space would hand the file descriptor for an existing socket to the BPF program, so that the kernel doesn't have to do name-resolution or anything like that.)

A well-anticipated example



A well-anticipated example

I always dreamed of that!
Not.

Email when?

Nobody seemed to disagree with this, which made it unclear whether netpoll really was the best approach to use. Unfortunately, at that point the session was out of time. Since the conference, Tardy and company have continued working on the problem, posting [a new patch set](#) on July 6 that allows BPF programs to create and use UDP kernel sockets, instead of using netpoll.

Index entries for this article
[Conference](#) [Storage, Filesystem, Memory-Management and BPF Summit/2026](#)

[Post a comment](#)

[–] Emails from the kernel
Posted Jul 14, 2026 14:51 UTC (Tue) by **Cyberax** (★ supporter ★, #52523) [\[Link\]](#) (1 responses) [Mute user](#)

Finally I'll be able to send alert emails directly from the kernel. I always dreamed of that!

Not.

[Reply to this comment](#)

[–] Emails from the kernel
Posted Jul 15, 2026 12:52 UTC (Wed) by **quodemstr** (subscriber, #45331) [\[Link\]](#) [Mute user](#)

It's on its way to becoming a WASM-equivalent alternative userspace.

[Reply to this comment](#)

[–] ... if an attacker can kill Tetragon's user-space agent, it won't be able to properly report on the situation

ity
its calendar
Unread comments
LWN FAQ
Write for us
Edition
Return to the Front page

Demo #3

Let's send emails
from BPF



Let's send emails from the kernel: we need TCP

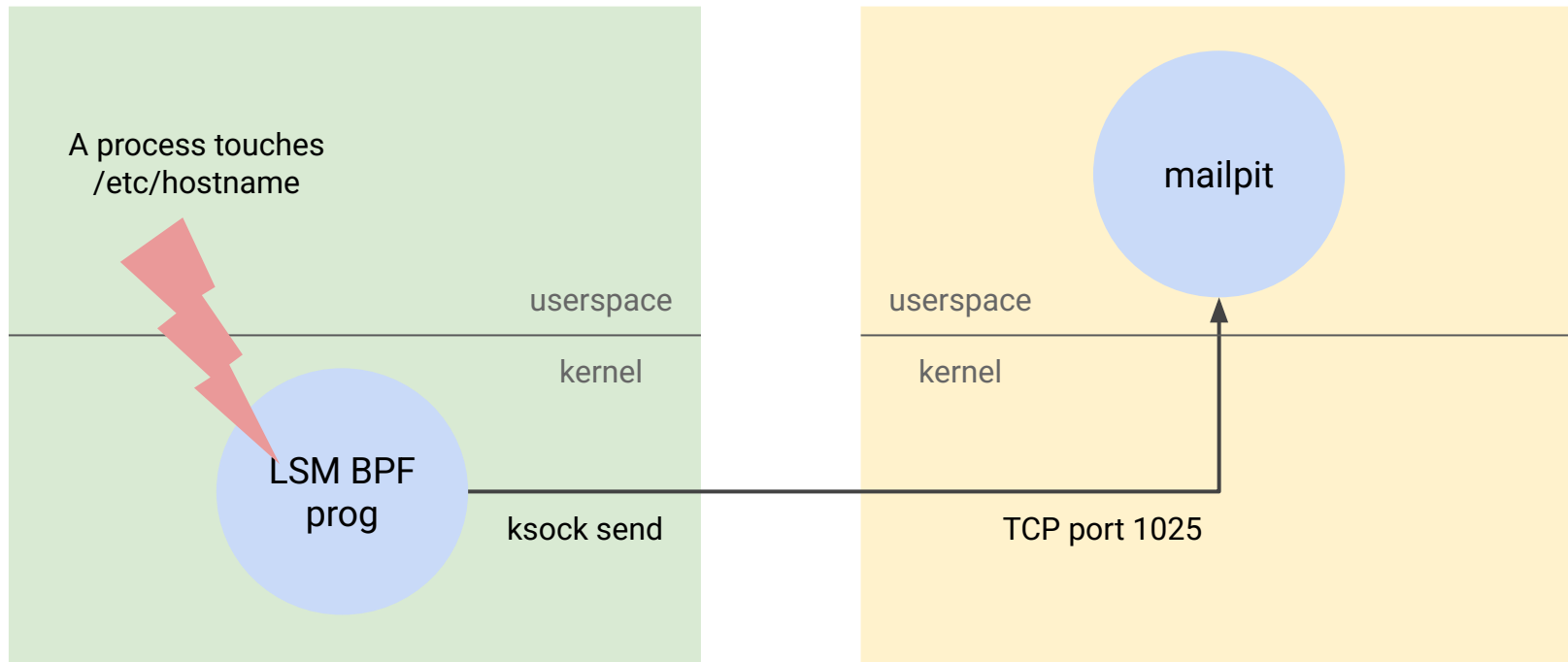
```
diff --git a/net/core/bpf_ksock.c b/net/core/bpf_ksock.c
index e9943aeeadd3..e3495b0d9795 10
--- a/net/core/bpf_ksock.c
+++ b/net/core/bpf_ksock.c
@@ -105,12 +105,12
     struct bpf_ksock_create_opts *opts, u32
     goto out;
 }

- if (opts_copy.type != SOCK_STREAM) {
+ if (opts_copy.type != SOCK_STREAM) {
     err = -EPROTOSPEC;
     goto err;
 }

- if (opts_copy.protocol != 0) {
+ if (opts_copy.protocol != IPPROTO_TCP && opts_copy.protocol != 0) {
     err = -EPROTOSPEC;
     goto err;
 }
```



Let's send emails from the kernel



The future of the ksock API

- ❖ Plumb more of the socket API in ksock BPF
 - Add `sendto()`, `sendmsg()`, and `bind()`
 - Allow to `connect()` from non-syscall BPF progs
- ❖ Allow the ksock API from more BPF progs
 - Support more sleepable program types (iterators, tracing)
 - Support non-sleepable progs with workqueues (non-sleepable (timers?) → workqueues)
- ❖ Support TCP
- ❖ Support receive from BPF