# BP Socket

## Linux Kernel API for DTN applications

Non-Terrestrial Networks (NTN) Days 2025
Open Source Project
pierrot.sylvain14@gmail.com

# BP Socket

## Linux Kernel API for DTN applications

**Address Family: AF_BP**

Non-Terrestrial Networks (NTN) Days 2025
Open Source Project
pierrot.sylvain14@gmail.com

# Agenda

About me

Story of BP Socket

The Bundle Protocol Socket

Open-ended Question

First Results

# 1. About me

Final-year engineering student at Polytech Montpellier (France)

Specializing in Cloud & DevOps engineering

Passionate about low-level & space technologies

Working on BP-Socket since November  2024


Sylvain Pierrot

# 2. Story of BP Socket

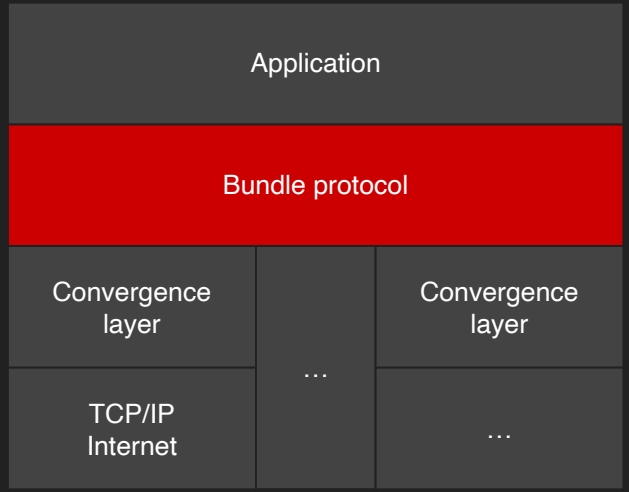
# Why BP Socket?

## Tackle a core DTN problem

- ➢ Lack of a clean, interoperable API for DTN applications
- ➢ Application-level adoption of BP remains limited due to its non-standard APIs

*A POSIX-style socket lets devs reuse existing code (switch the address family to **AF_BP** + minimal code changes)*

Bundle Protocol overlay



# How ? STINT 2024 Hackathon

BP Socket prototype was born at the STINT 2024 Hackathon

An afternoon event organized by Scott Burleigh (July 19th)

Collaborative project with 8 participants:

- Scott Burleigh (JPL)
- Felix Walter (D3TN)
- Olivier De Jonckère (LIRMM)
- Juan Fraire (Inria)
- Brian Sipos (APL)
- Samo Grasic (ICTP)
- Brian Tomko (NASA)
- Ricardo Lent (UH)

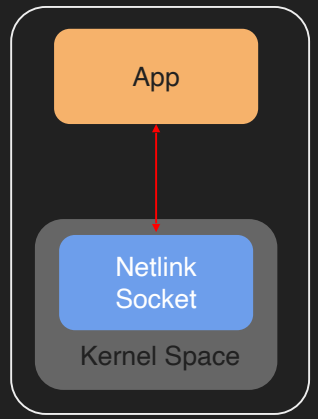
# 3. The Bundle Protocol Socket

*POSIX Sockets meet Delay-Tolerant Networking*

# What is a Linux Socket Interface?

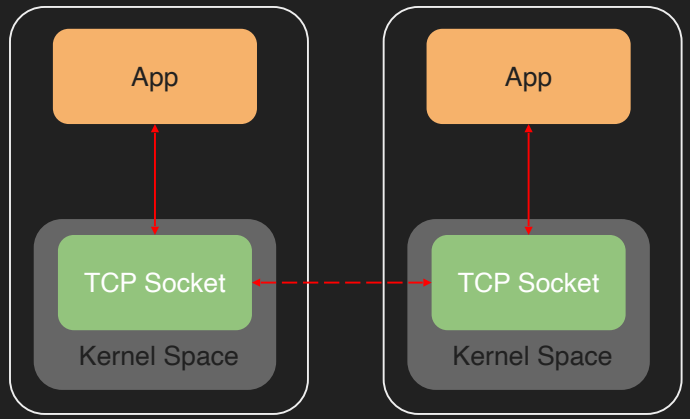
Abstraction provided by the kernel to enable communication between two entities. Sockets are categorized by Address Families.

Examples:




Kernel ⇔ Userspace IPC

(AF_NETLINK)




Abstract network I/O

(AF_INET)

# BP Socket in a Nutshell

## Overview

Linux socket family AF_BP for DTN applications

BP-compliant addressing (e.g. `ipn:20.3` )

Bridge between apps and ION DTN

## System Calls

Core Operations:

- `socket, bind, close`

I/O Operations:

- `sendmsg, recvmsg, poll`

Configuration:

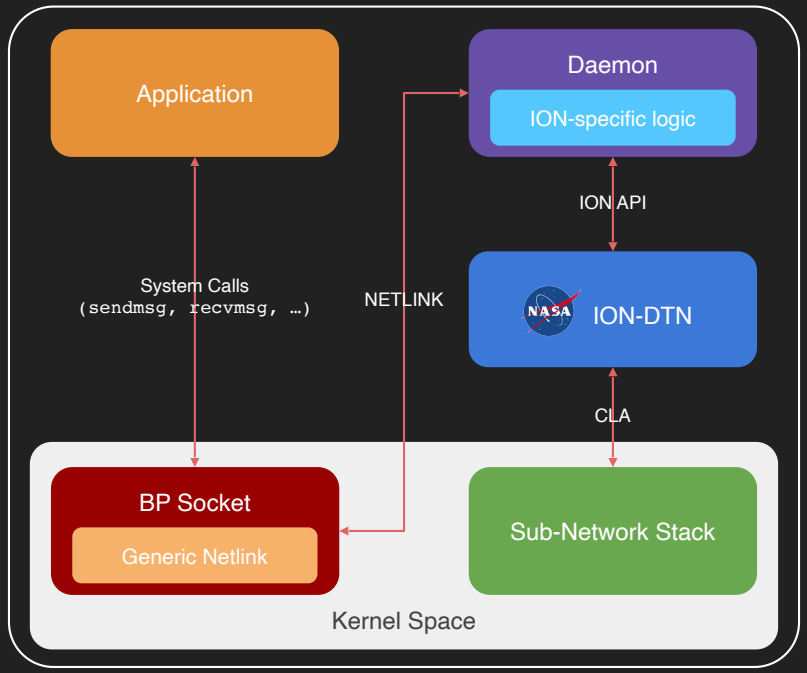
- `setsockopt, getsockopt`

## Example

```
int fd = socket(AF_BP, SOCK_DGRAM, 1);
```

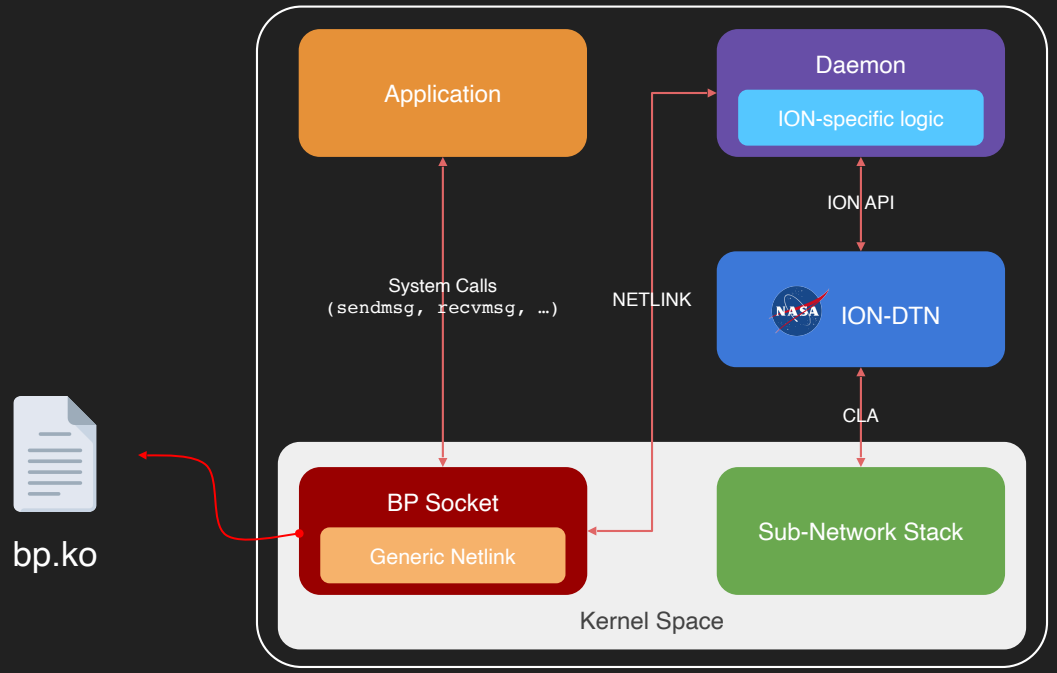
# Architecture

## Bridging applications and ION (Interplanetary Overlay Network)



# Architecture

## Bridging applications and ION (Interplanetary Overlay Network)



# Socket Addressing: Internet vs Bundle Protocol

| | Internet socket (AF_INET) | BP socket (AF_BP) |
|---|---|---|
| Address family | AF_INET | AF_BP |
| Address struct | `struct sockaddr_in {`<br>`  sa_family_t  sin_family;`<br>`  in_port_t  sin_port;`<br>`  struct in_addr  sin_addr;`<br>`};` | `struct sockaddr_bp {`<br>`  sa_family_t bp_family;`<br>`  bp_scheme_t bp_scheme; // IPN or DTN`<br>`  union {`<br>`  struct { uint32_t node_id; uint32_t service_id; } ipn;`<br>`  } bp_addr;`<br>`};` |
| URI form | <addr>:<port> | ipn:<node_id>.<service_id> |
| Example | 192.168.2.1:8080 | ipn:10.2 |
| bind() semantics | expose (addr, port) | expose (node_id, service_id) |

# Datagram-Oriented Socket

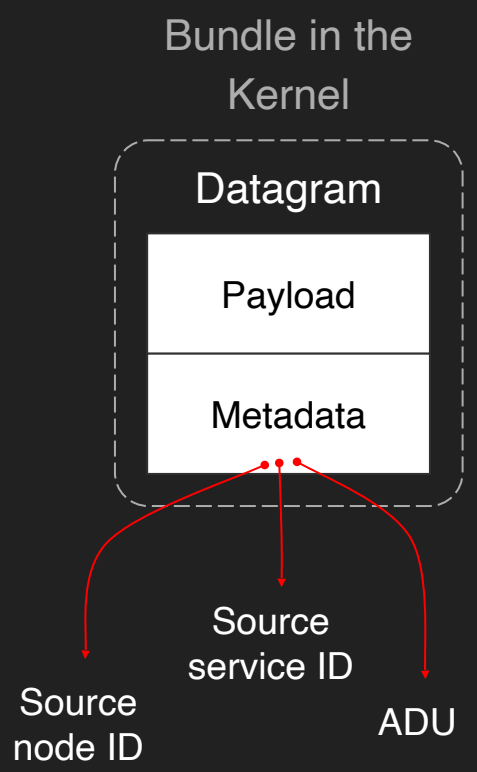
Bundle = kernel datagram (`struct sk_buff`)

Connection-oriented calls are disabled

~~`connect`~~`()`, ~~`listen`~~`()`, ~~`accept`~~`()`, …

Data transmission and reception after `bind()`

Implicit binding not supported



# Available Socket Options

Applications can configure BP sockets using `setsockopt()` / `getsockopt()` with `SOL_SOCKET`

| Option | Description |
| --- | --- |
| SO_RCVTIMEO | Set a **receive timeout** (example: 3 seconds) |

Example

```
struct timeval tv = { .tv_sec = 3, .tv_usec = 0 };
int rc = setsockopt(fd, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv));
```

# Available Receive Message Flags

Applications can use the following flags with `recvmsg()`

| Flag | Description |
|---|---|
| MSG_PEEK | Peeks at the incoming datagram without removing it from the queue. Useful for inspecting message size or content before full reception. If the buffer is too small the operation failed. |
| MSG_TRUNC | Returns the full size of the message, even if the provided buffer is too small. The datagram is consumed from the queue, and only the portion that fits is copied to the buffer. The MSG_TRUNC flag is set in msg_flags. |

Example

```
ssize_t need = recvmsg(fd, &msg, MSG_PEEK | MSG_TRUNC);
```

# Safe Two-Steps Receive Strategy

**Step 1 – Probe the message size:**

Use recvmsg() with MSG_PEEK | MSG_TRUNC
→ This reads the message metadata **without consuming** it,
and tells you **how big your buffer needs to be**.

```
ssize_t need = recvmsg(fd, &msg, MSG_PEEK | MSG_TRUNC);
```

**Step 2 – Allocate & receive fully:**

Once you know the required size (need),
allocate the correct buffer and call recvmsg() again:

```
recvmsg(fd, &msg, 0);
```

# Available Send Message Flags

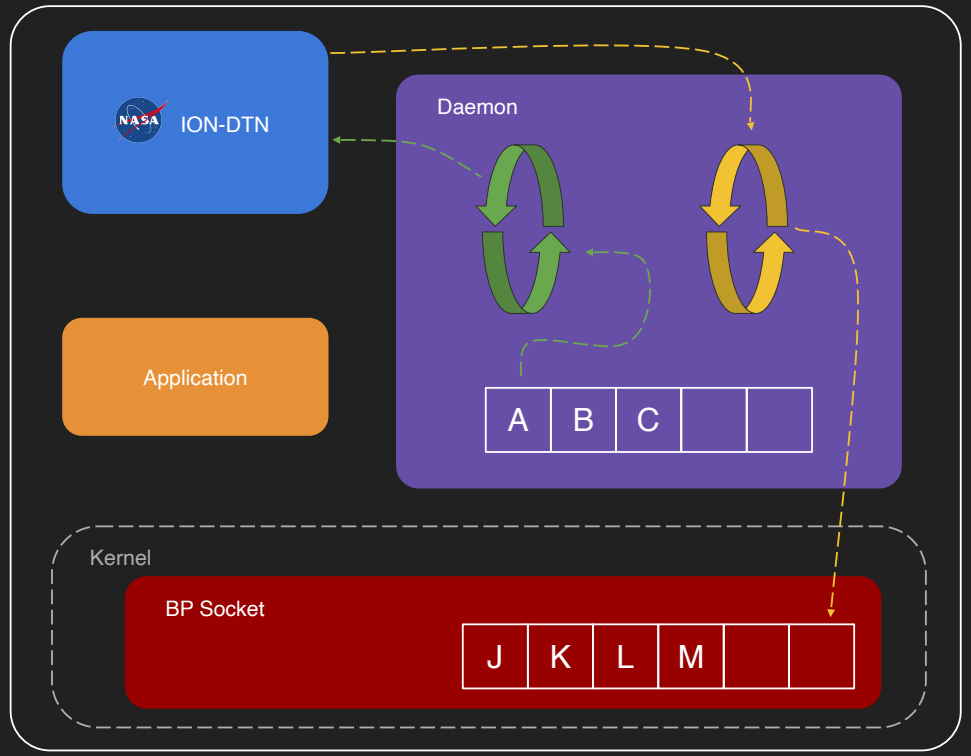
Applications can tune delivery, reporting, priority, and custody using flags

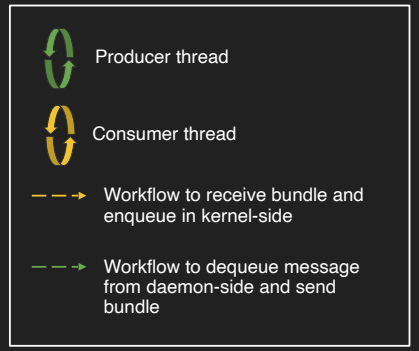
| Category | Flags |
|---|---|
| **Acknowledgment** | MSG_ACK_REQUESTED |
| **Status Reports** *(combinable)* | MSG_RECEIVED_RPT, MSG_CUSTODY_RPT, MSG_FORWARDED_RPT, MSG_DELIVERED_RPT, MSG_DELETED_RPT |
| **Priority** *(mutually exclusive)* | MSG_BP_BULK_PRIORITY, MSG_BP_STD_PRIORITY, MSG_BP_EXPEDITED_PRIORITY |
| **Custody** *(mutually exclusive)* | MSG_SOURCE_CUSTODY_REQUIRED, MSG_SOURCE_CUSTODY_OPTIONAL, MSG_NO_CUSTODY_REQUIRED |

Example

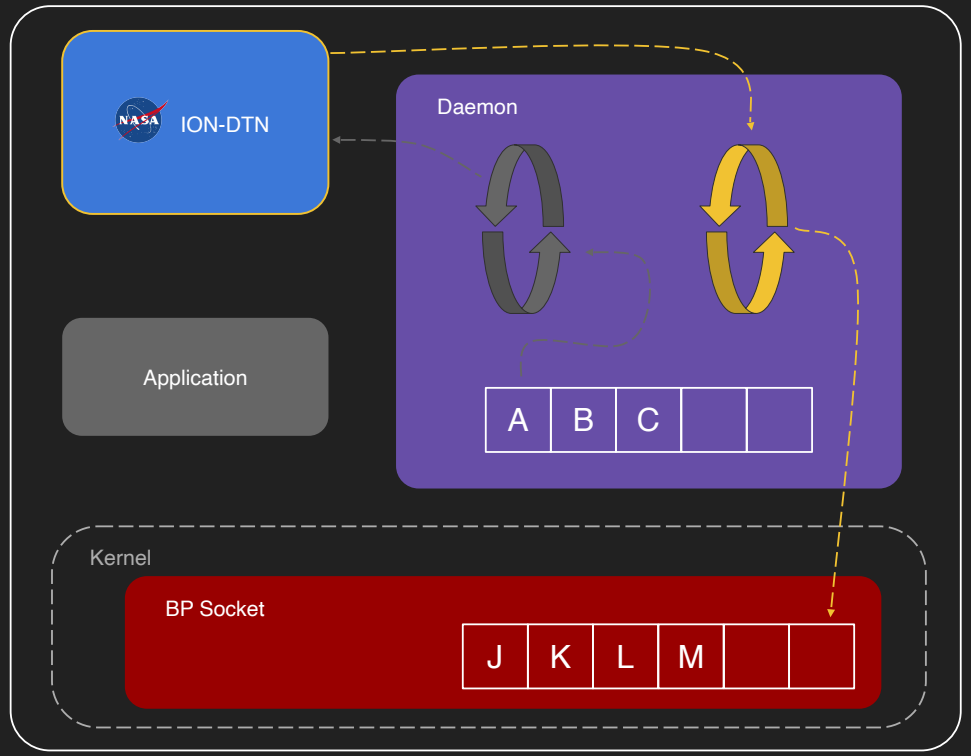
```
ssize_t sent = sendmsg(fd, &msg, MSG_ACK_REQUESTED |
                       MSG_NO_CUSTODY_REQUIRED);
```

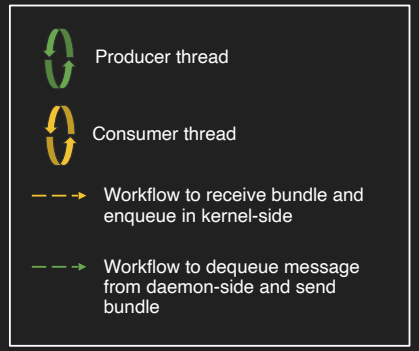
# Data Flow Architecture



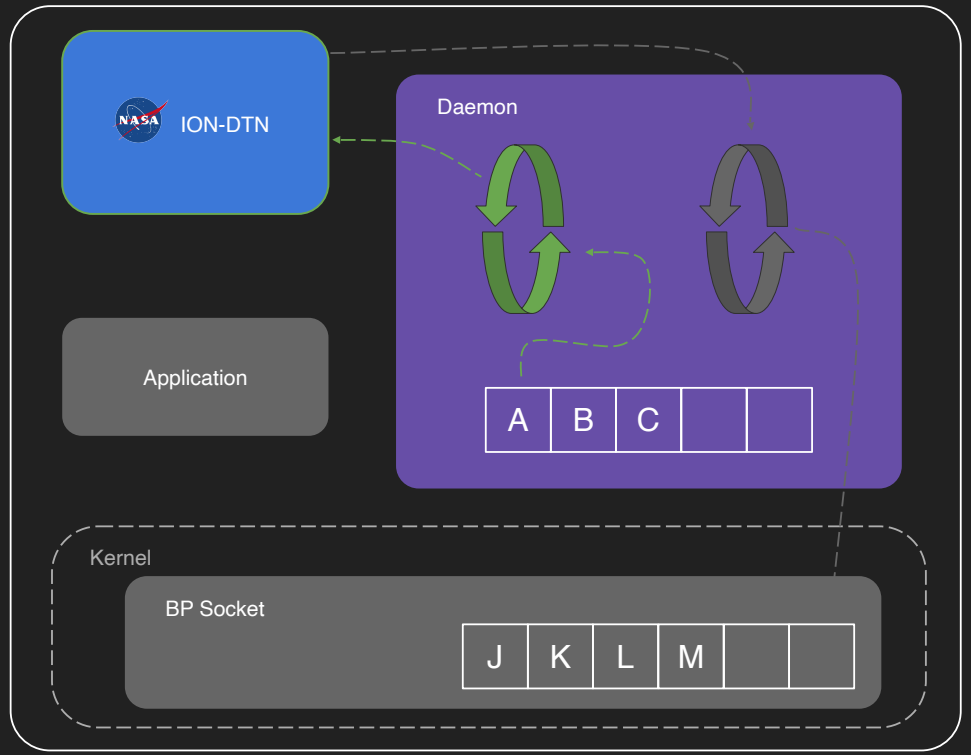



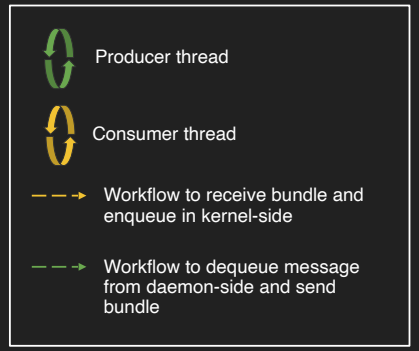
# Data Flow Architecture



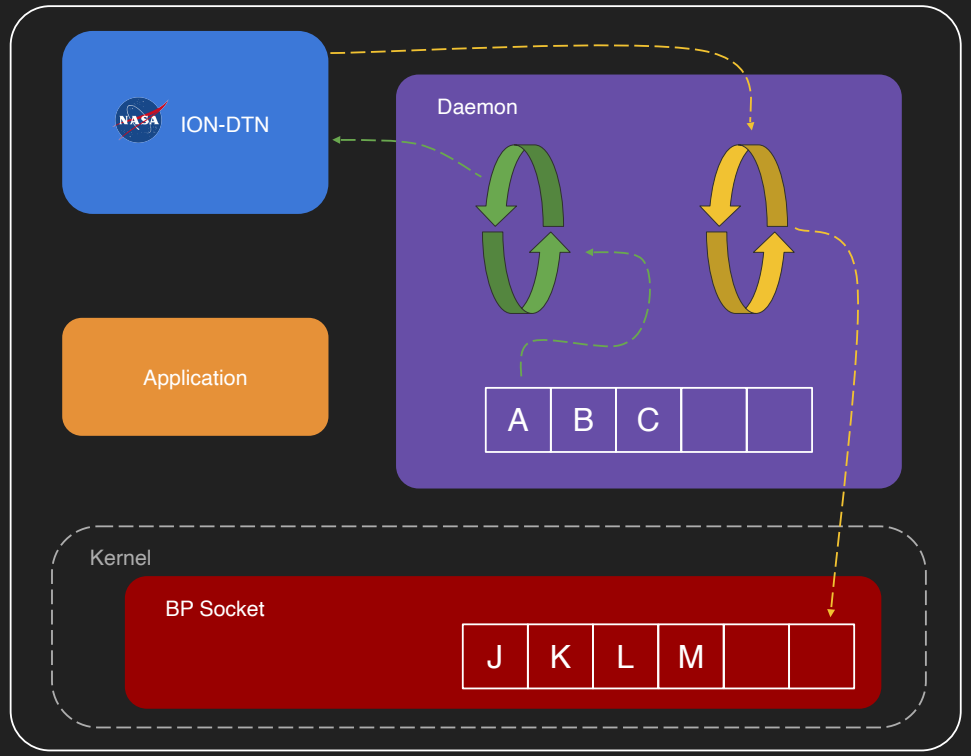



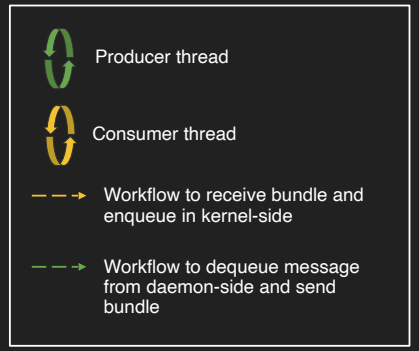
# Data Flow Architecture



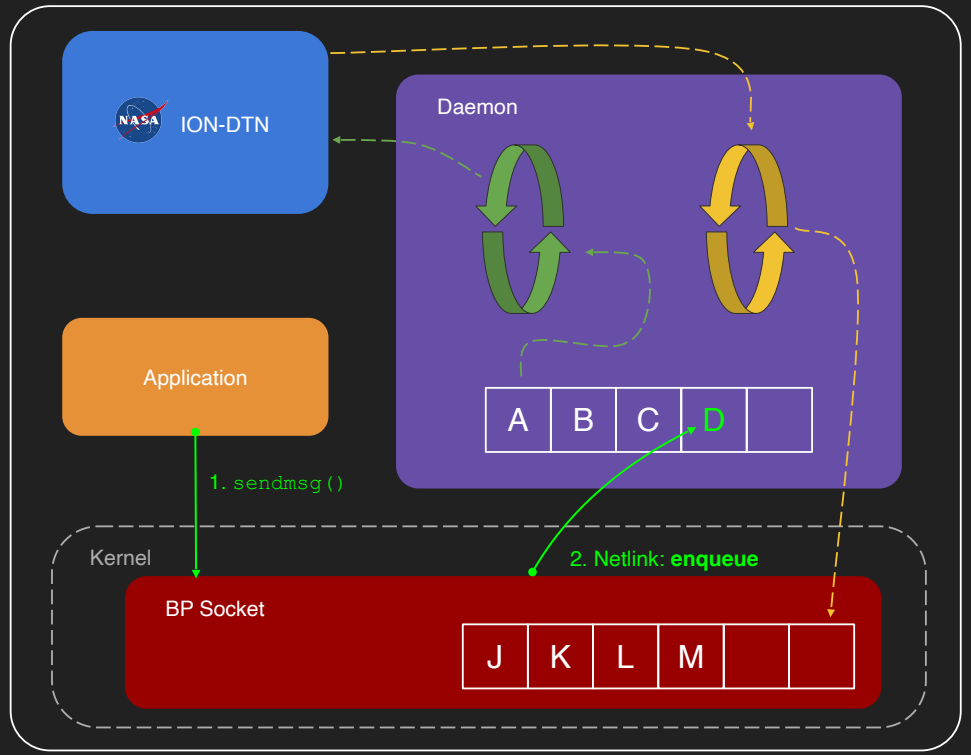



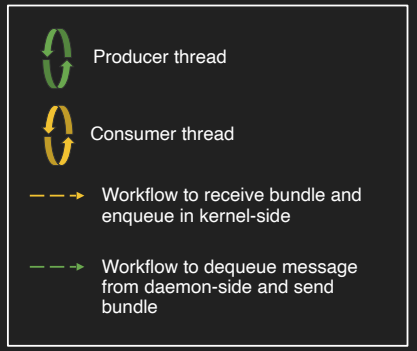
# Data Flow Architecture



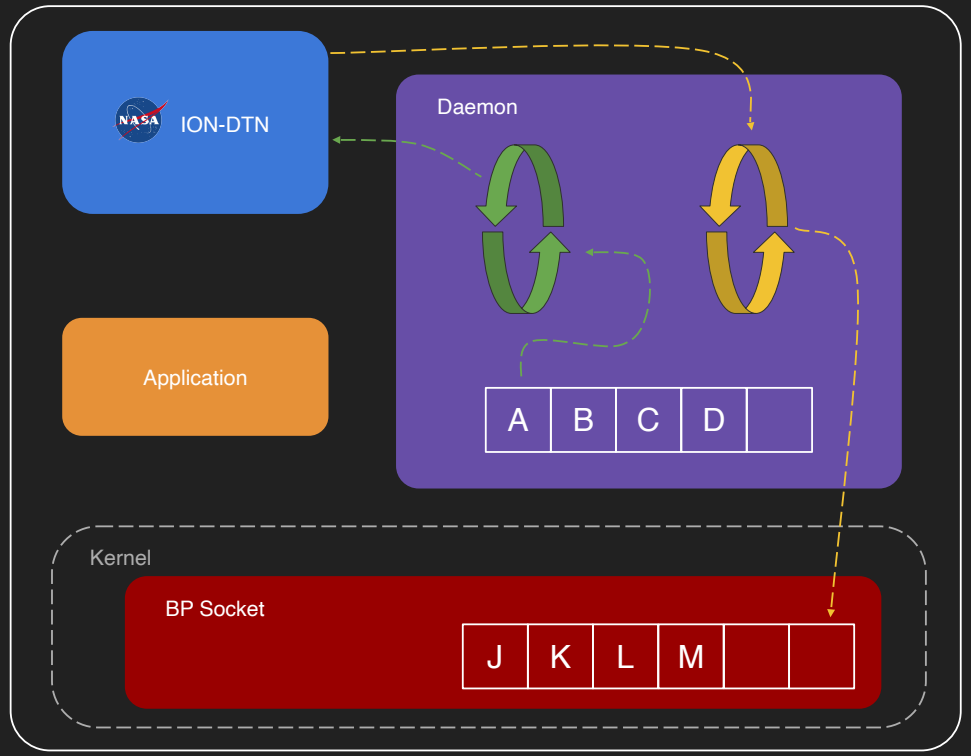



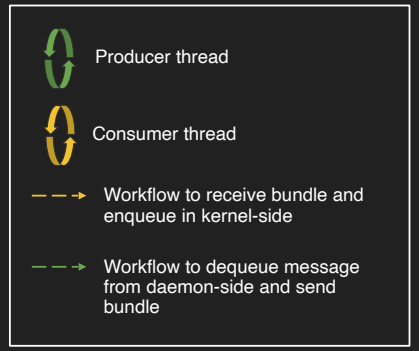
# Data Flow Architecture



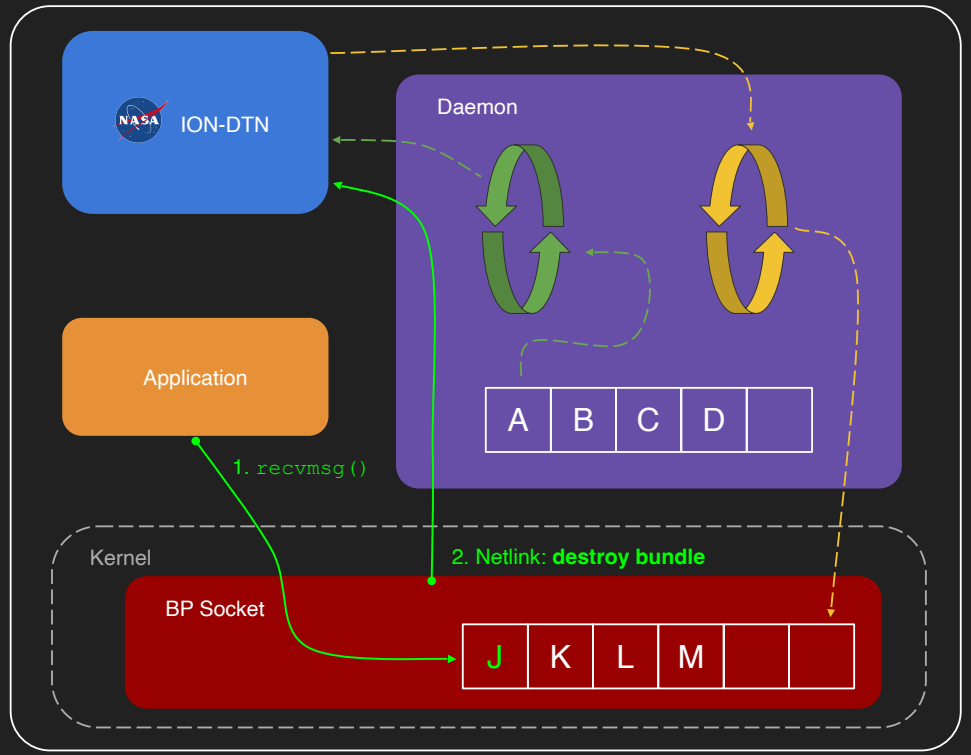



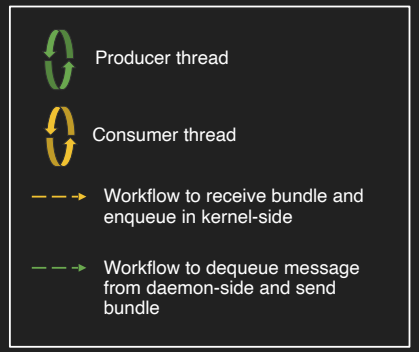
# Data Flow Architecture



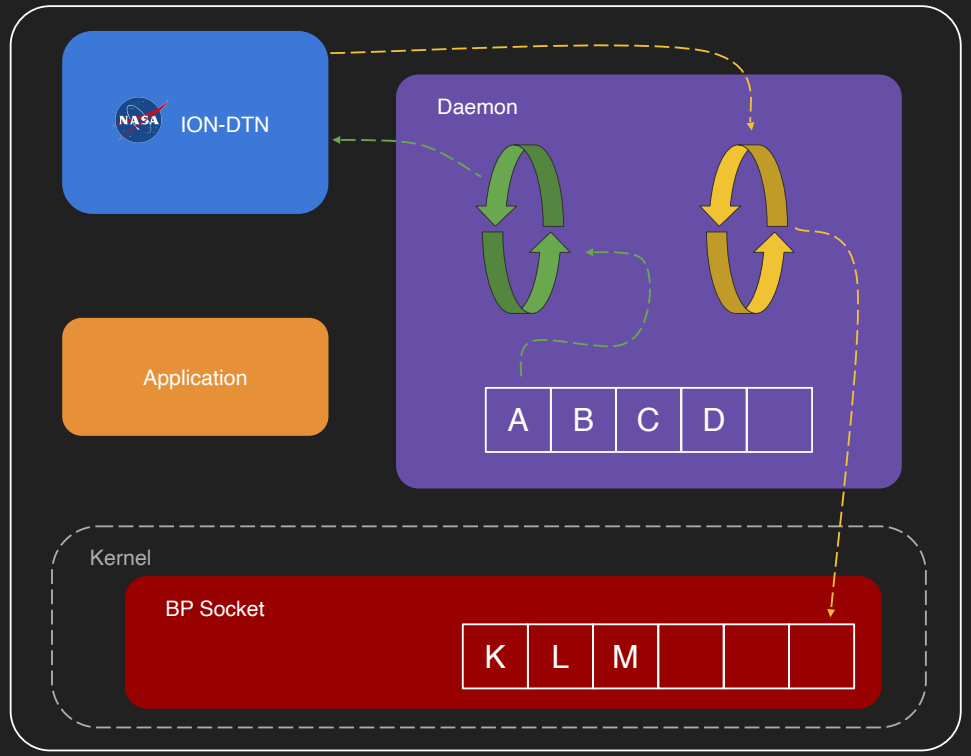



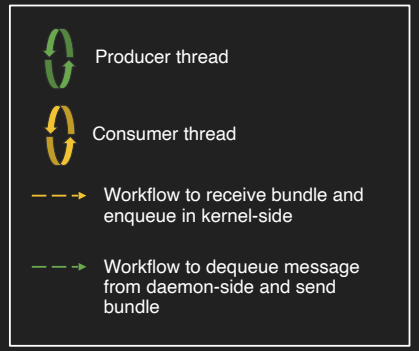
# Data Flow Architecture



# Data Flow Architecture



# 4. Open-ended Question

# Become a “BP Socket” contributor

🧩 *Current behavior:*

When a BP socket is closed, the entire receive queue in the kernel is dropped.

🤔 *Open Question:*

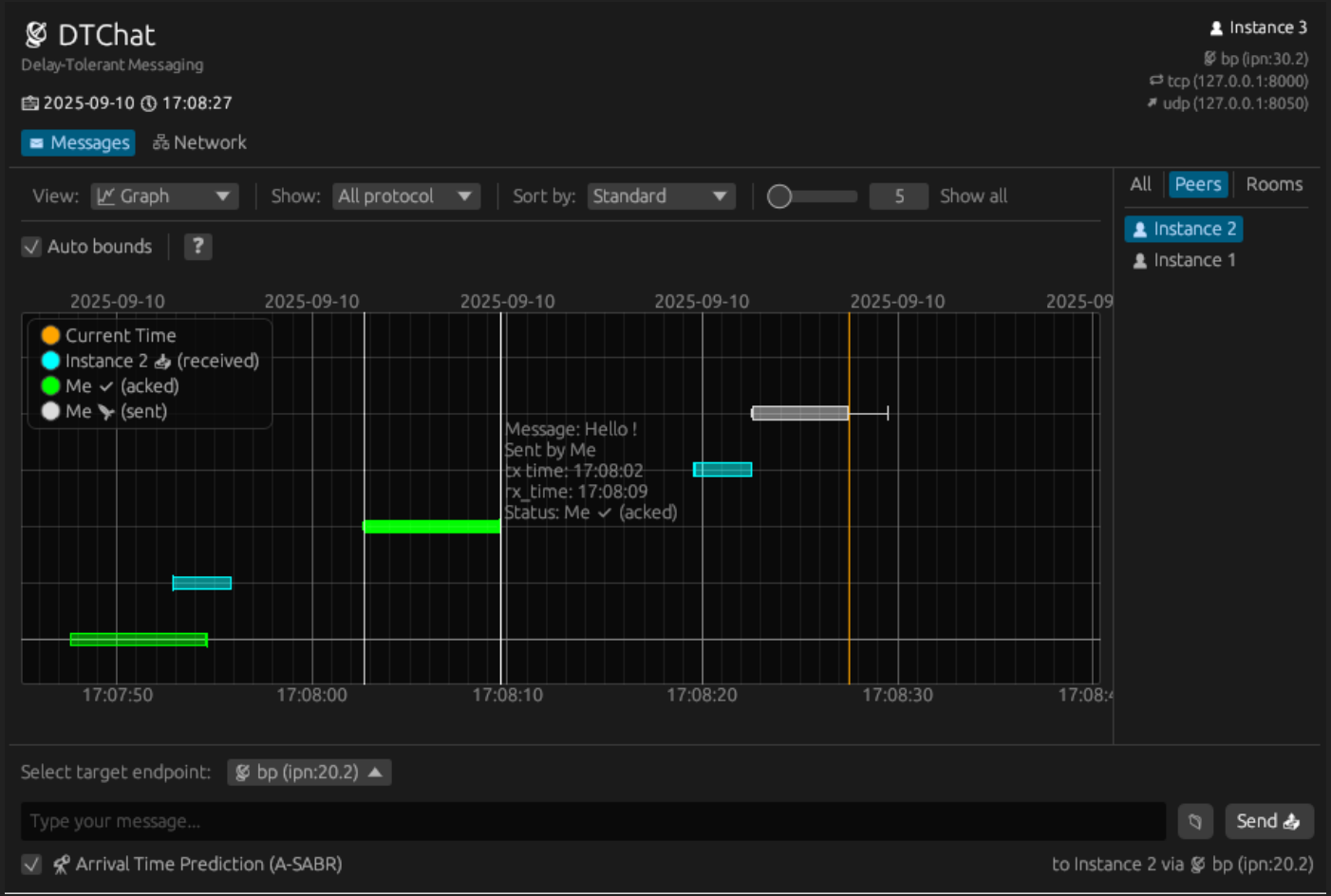
Should pending bundles be flushed on socket close,

or **persisted and re-delivered** when the same endpoint is reopened?

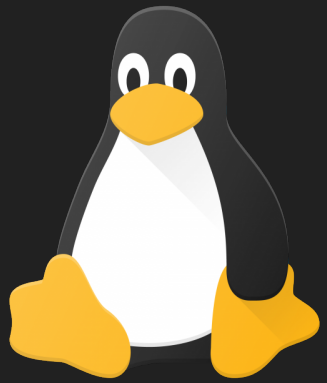
https://github.com/DTN-MTP/bp-socket/issues/39

# 5. First Results

*From proof-of-concept to practical usage*


DTChat
Delay-Tolerant Messaging
2025-09-10 17:08:27
Messages
Network
Instance 3
bp (ipn:30.2)
tcp (127.0.0.1:8000)
udp (127.0.0.1:8050)
View: Graph
Show: All protocol
Sort by: Standard
5
Show all
All
Peers
Rooms
Instance 2
Instance 1
Auto bounds
2025-09-10
Current Time
Instance 2 (received)
Me (acked)
Me (sent)
Message: Hello !
Sent by Me
tx time: 17:08:02
rx_time: 17:08:09
Status: Me (acked)
17:07:50
17:08:00
17:08:10
17:08:20
17:08:30
Select target endpoint:
bp (ipn:20.2)
Type your message...
Send
Arrival Time Prediction (A-SABR)
to Instance 2 via bp (ipn:20.2)


DTChat
Delay-Tolerant Me
2025-10-02
Messages
View:
Auto bound
2025-10-02
Current T
08:59:45
Select a peer/r
Type your me
Arrival Time Prediction (A-SABR)
Instance 1
bp (ipn:10.2)
27.0.0.1:6500)
27.0.0.1:6550)
rs
Rooms
ll messages
Send
Help
Drag to move. Scroll to move up/down. Ctrl & scroll to zoom in/out. Enable the "Auto bounds" option to adapt the view dynamically. Click on the legend to display/hide elements. Hovering over any box displays the corresponding basic message information.
Received messages appear as cyan boxes without whiskers. The sender is indicated in the legend. The box spans from the message creation time to its reception time.
Current Time
Peer (received)
Sent messages (acknowledged by the peer) appear as green boxes. They may include a left whisker, representing the delay between delegating transmission to the socket (asynchronous) and the socket's confirmation that transmission is complete.
Current Time
Me (acked)
Sent messages (not acknowledged) appear as light grey boxes, spanning from the sending time to the current time. If enabled, a right whisker shows the predicted arrival time.
Current Time
Me (sent)
Because of round-trip delays, once the current time surpasses the predicted arrival time, the whisker is replaced by the boxplot median.
Current Time
Me (sent)
Messages that encounter an error upon sending appear as red boxes, spanning from the sending time to the current time.
Current Time
Me (error)



# Thank You!

https://github.com/DTN-MTP/bp-socket

pierrot.sylvain14@gmail.com

# Appendix – Additional Materials

# Flushing Datagram Queue on Close

Problem:

When a datagram socket is closed, the entire receive queue in the kernel is dropped.

👉 For BP Socket, this means losing all pending ADUs (Application Data Units) still buffered.

# ION SDR Management Issue

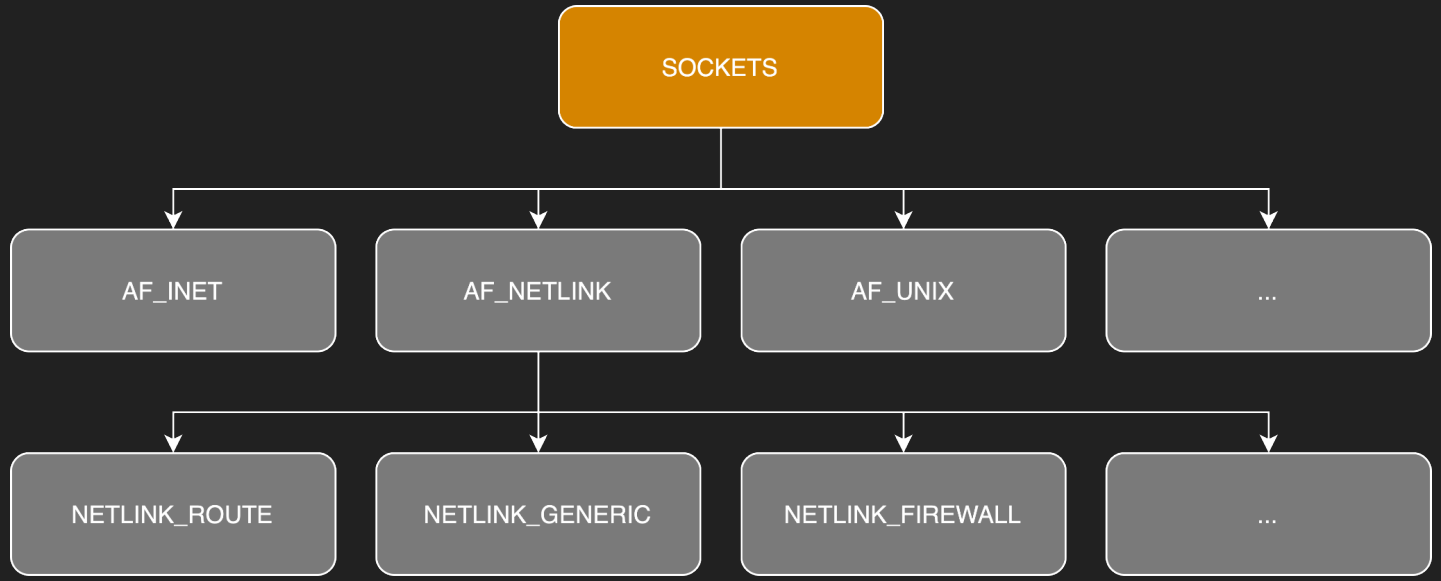
Problem:

When BP Socket sends bundles at high throughput, ION's Space Data Repository (SDR) becomes saturated, causing ION to crash.

👉 Future work will focus on enhancing the daemon's SDR management capabilities to prevent ION crashes under high load.

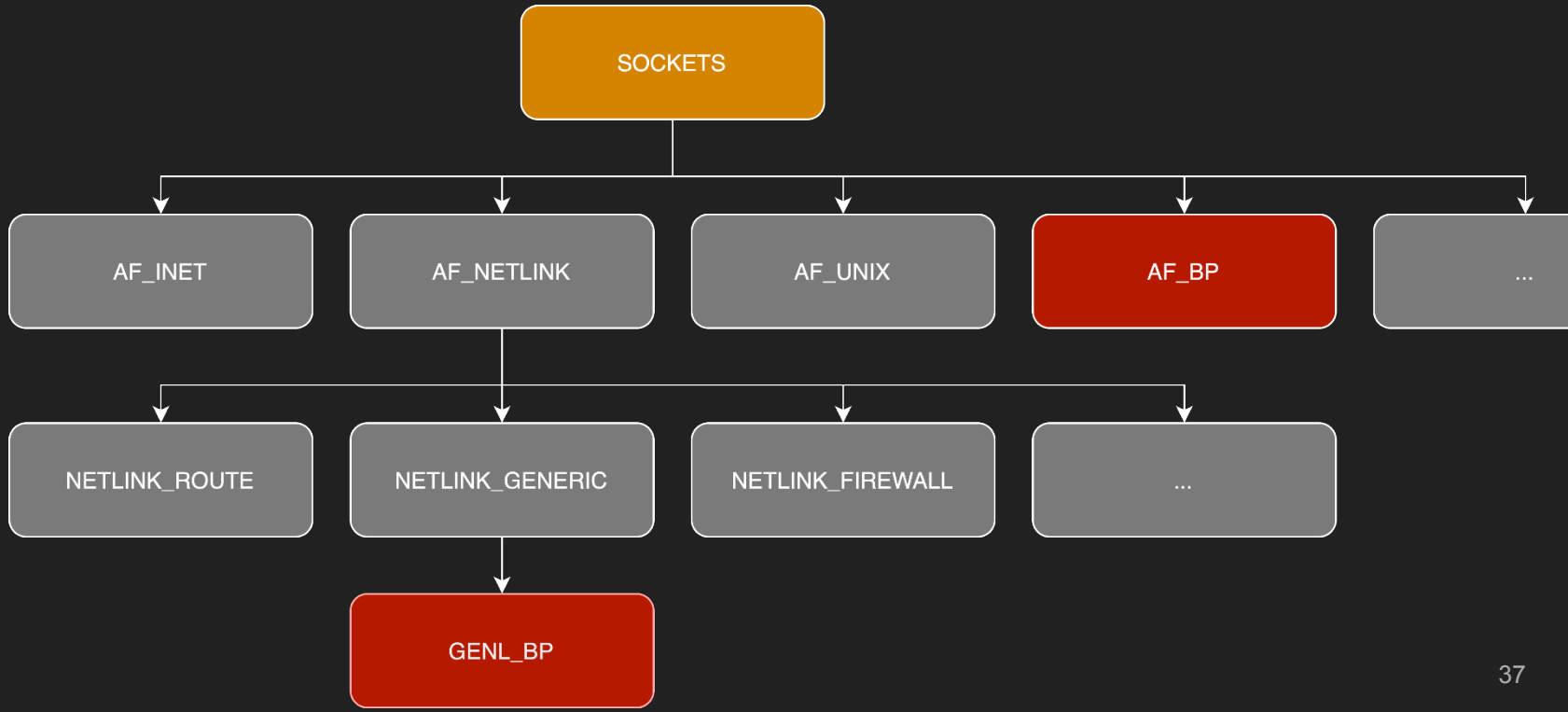
# Sockets Families Hierarchy

## Core socket families



# Sockets Families Hierarchy

## Address Family: *Bundle Protocol*



# Post-Hackathon Codebase

Painful to understand and work with

Large amount of dead code (e.g. unused Unix socket)

Mostly copied from the project **Secure Socket API (SSA)**

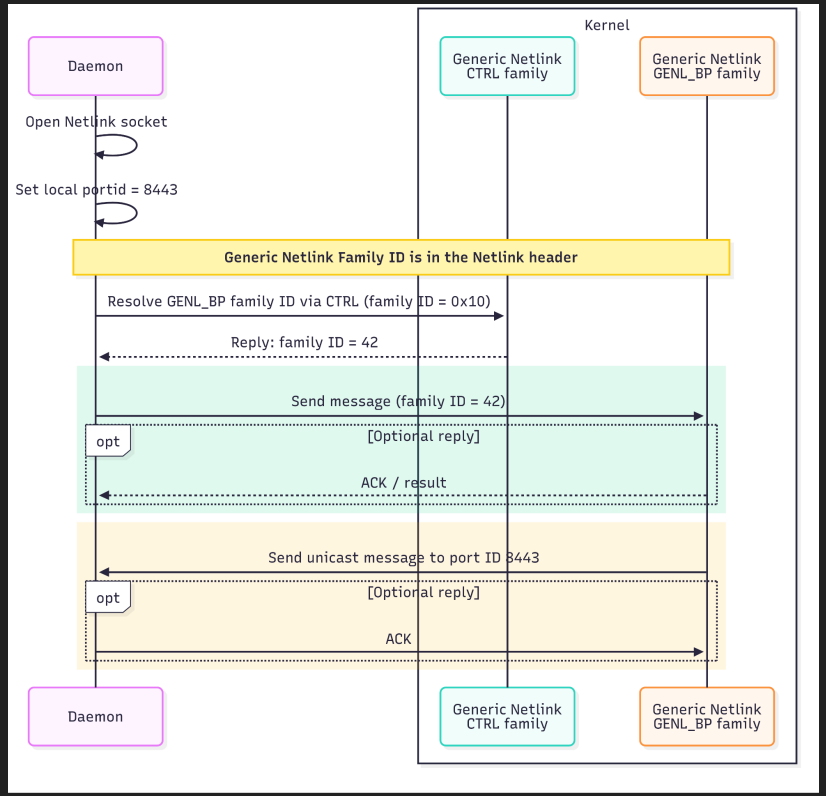
No documentation or setup instructions for development

Only supported sending string message

# Generic Netlink Communication

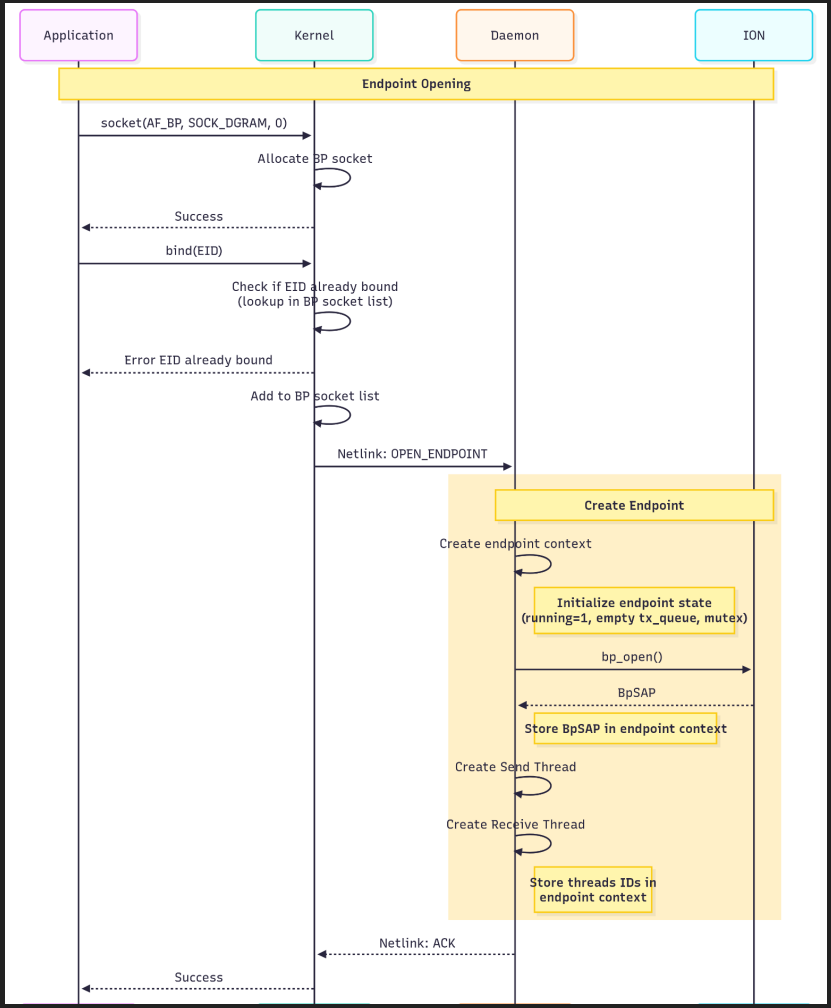
## Kernel ↔ Daemon IPC Coordination

Note. In Generic Netlink, the family ID is carried in the Netlink message header. CTRL is the only family with a fixed ID (**GENL_ID_CTRL = 0x10**). All other families (e.g., **GENL_BP**) IDs are dynamically assigned by the kernel at registration time; you must resolve that ID at runtime via CTRL and reuse it in all subsequent messages.



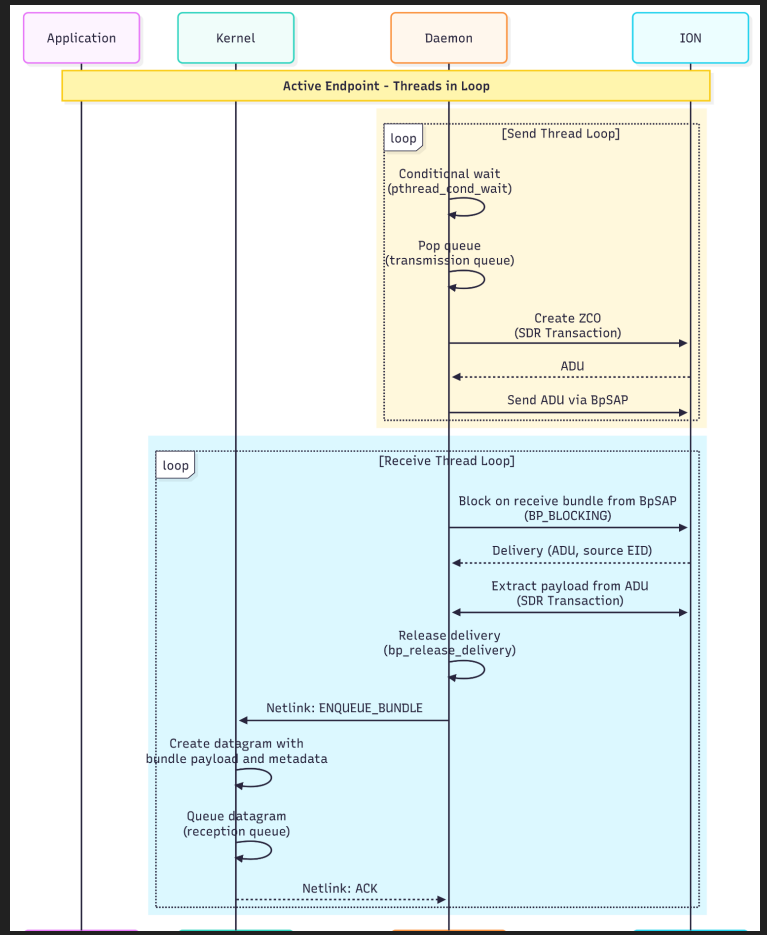
# BP Socket Lifecycle

## Endpoint opening



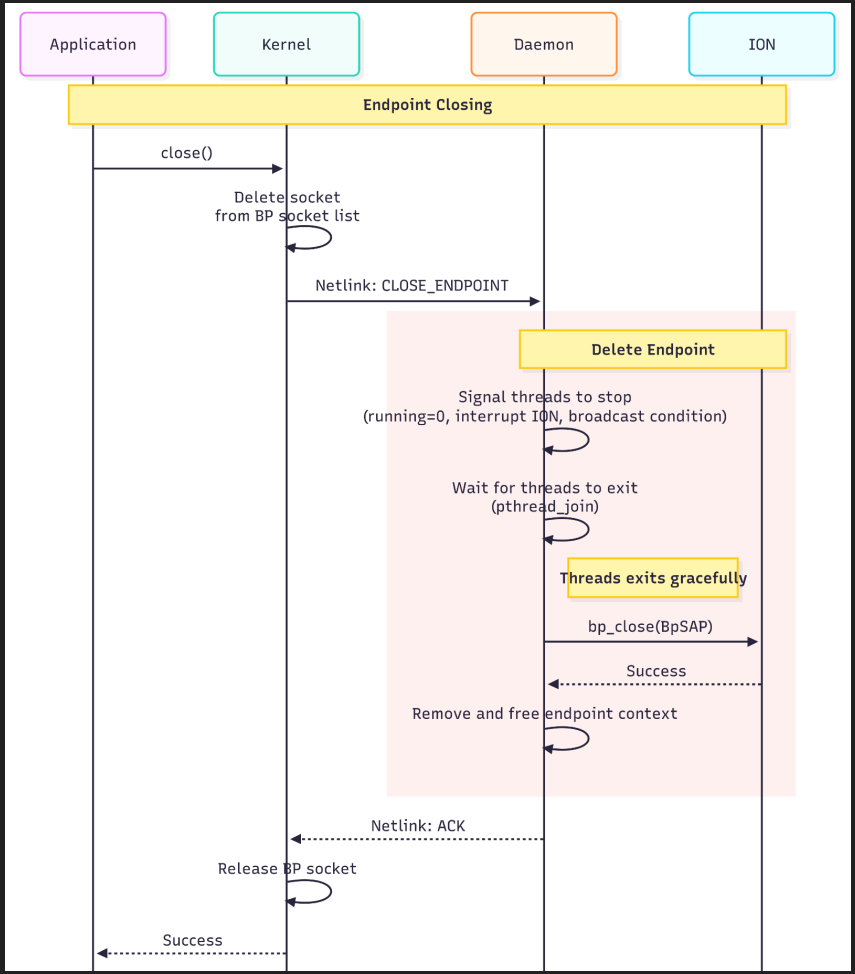
# Endpoint Active State

Two loops per endpoint



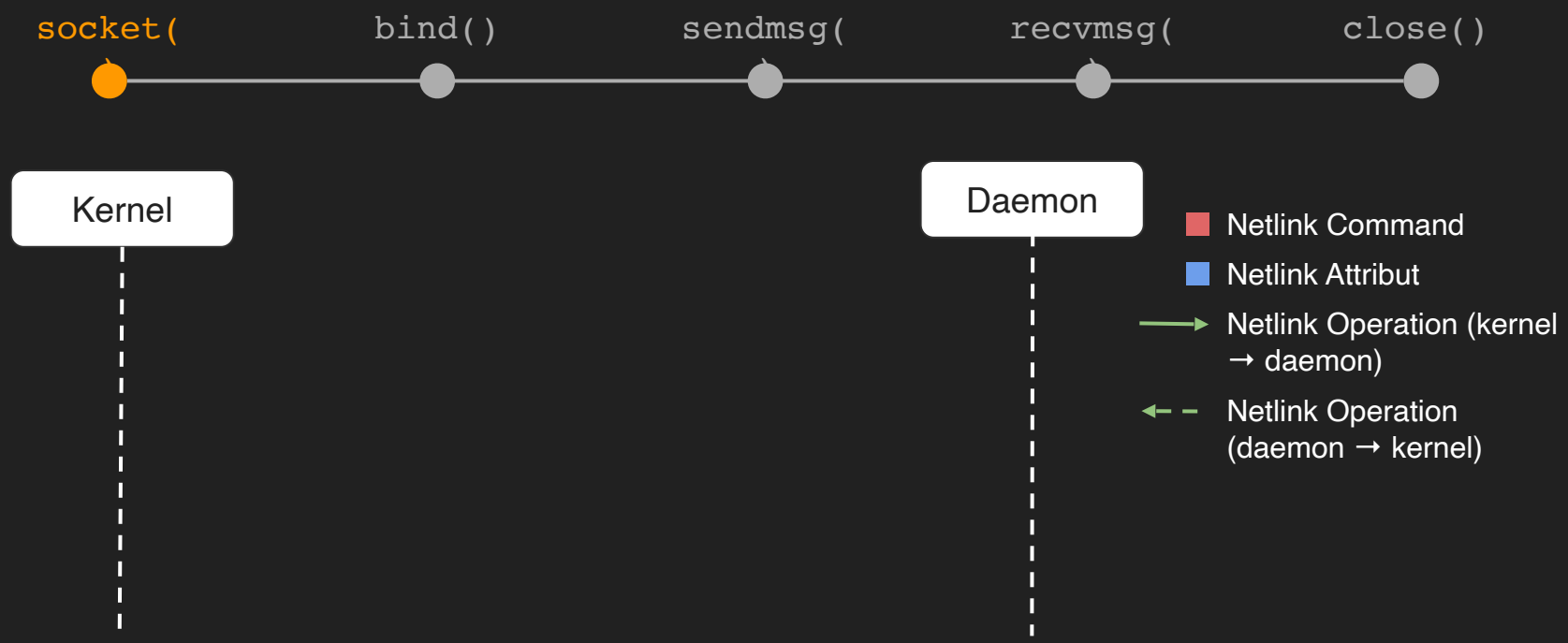
# BP Socket Lifecycle

## Endpoint closing



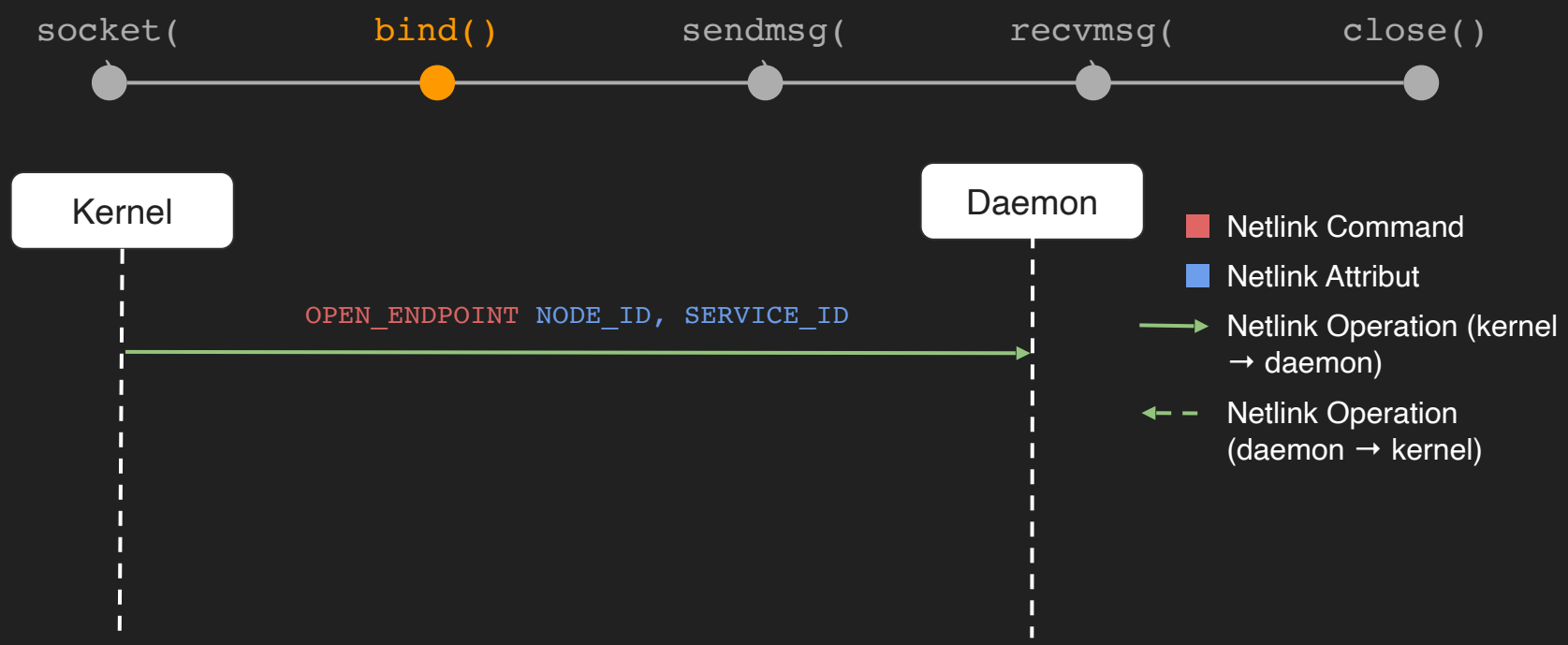
# Kernel ↔ Daemon IPC Coordination (Generic Netlink)

## BP Socket Lifecycle



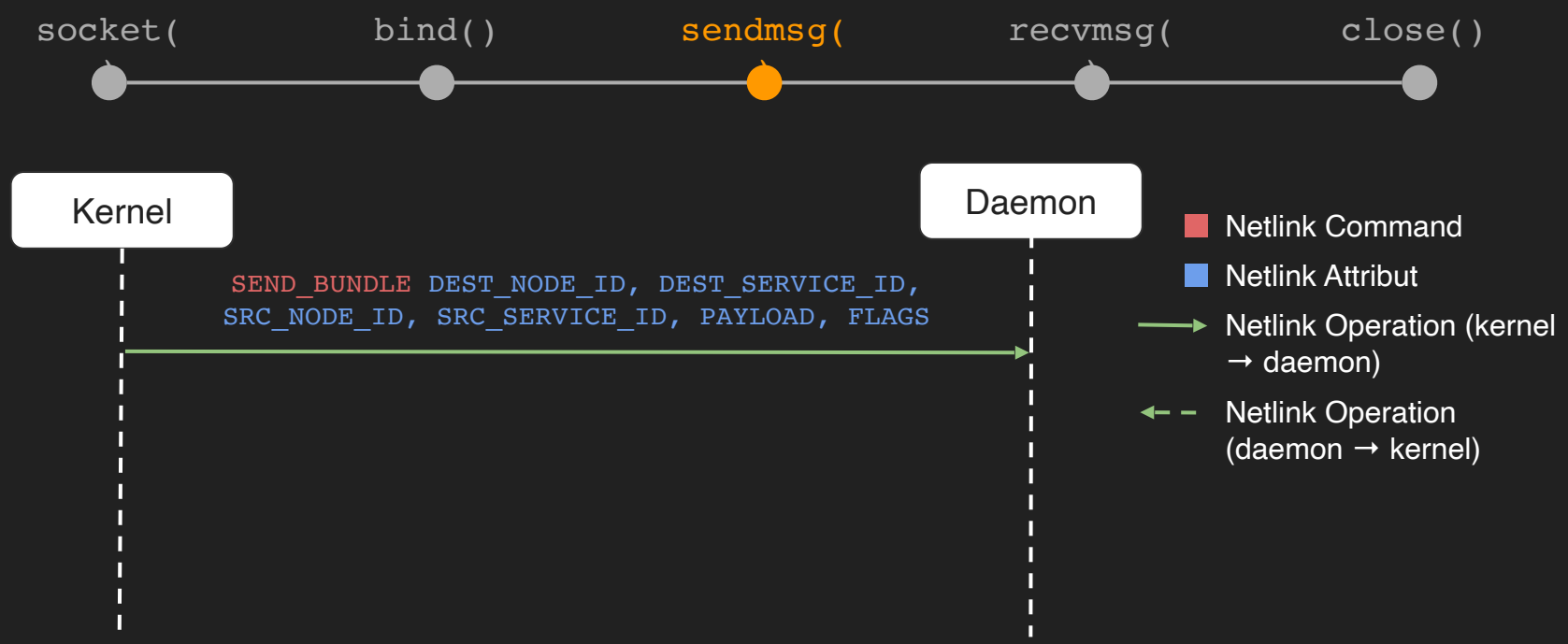
# Kernel ↔ Daemon IPC Coordination (Generic Netlink)

## BP Socket Lifecycle



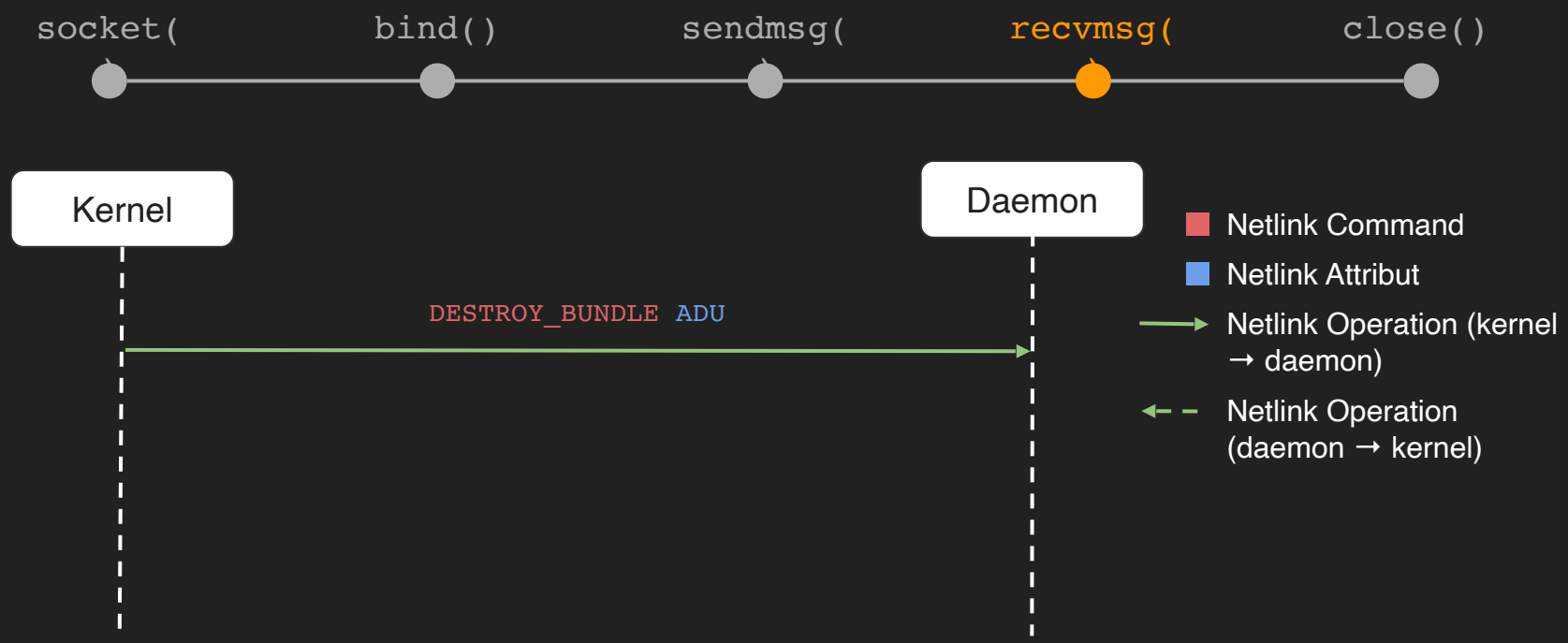
# Kernel ↔ Daemon IPC Coordination (Generic Netlink)

## BP Socket Lifecycle



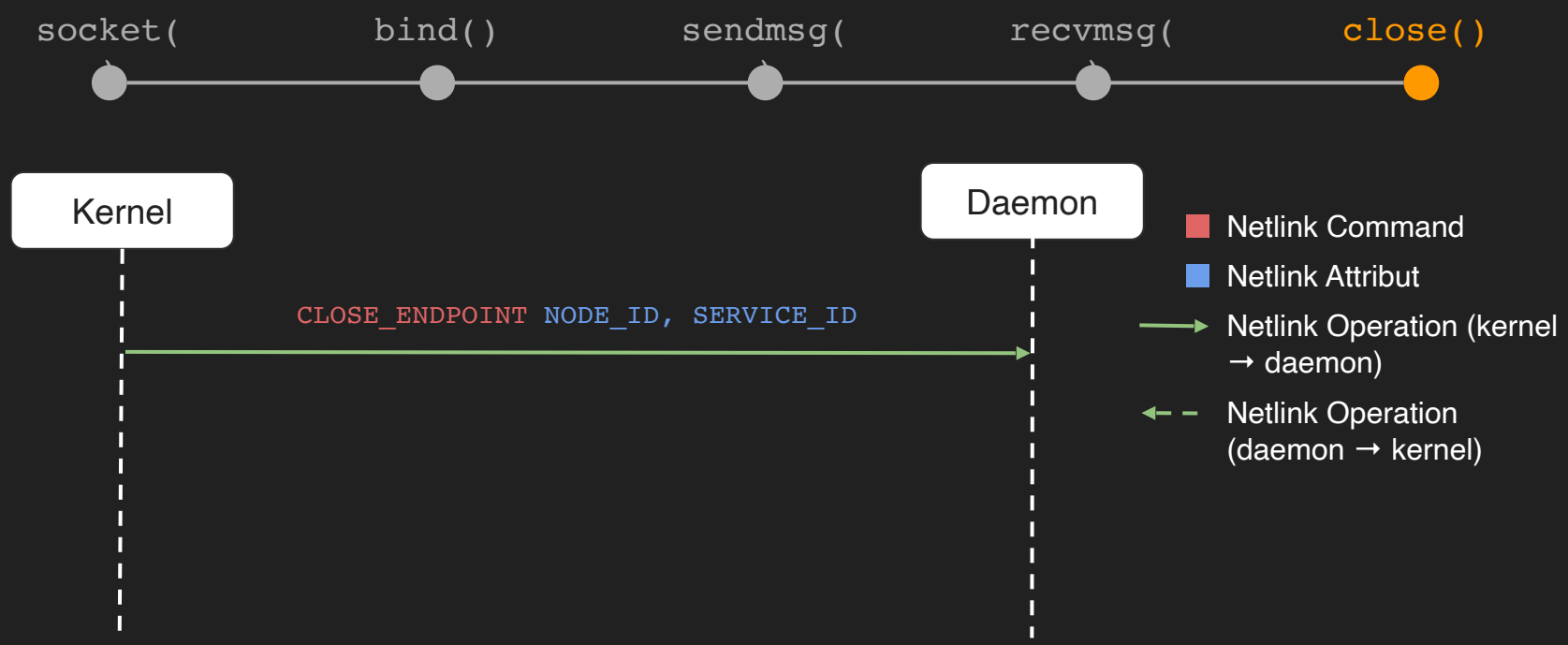
# Kernel ↔ Daemon IPC Coordination (Generic Netlink)

## BP Socket Lifecycle



# Kernel ↔ Daemon IPC Coordination (Generic Netlink)

## BP Socket Lifecycle



# Development Environments

Automated provisioning on **Linux** and **MacOS**

- Provisions ready-to-use DTN nodes: ION & $\mu$D3TN
- Auto-syncs your host project into in **`/bp-socket`** on the ION node
- Installs required packages, build tools, and dependencies
- Creates an isolated private network: 192.168.50.10 & 192.168.50.20
- DTN endpoints: ipn:10.0 (ION) & ipn:20.0 ($\mu$D3TN)

HashiCorp Vagrant **OR** Canonical Multipass + cloud-init

# Continuous Integration

Automated PR checks with GitHub Actions

- Runs on pull requests touching kernel, daemon, or shared headers
- Enforces style: clang-format v20
- Kernel job: installs deps, builds kernel module & runs Sparse static analysis
- Daemon job: installs deps, builds & installs ION 4.1.3, builds daemon
- Early failure before merge — keeps main buildable & consistent

# **BP Socket**: Kernel-side Structure (*struct bp_sock*)

af_bp.h

```
struct bp_sock {
    struct sock         sk;              /* embedded base sock */
    u32                 bp_node_id;      /* bound via bind() */
    u32                 bp_service_id;   /* bound via bind() */
    struct sk_buff_head rx_queue;      /* inbound bundles from
daemon */
    wait_queue_head_t   rx_waitq;      /* block/wake recvmsg() */
};
```

# **BP Socket**: Allocate and Access (*struct bp_sock*)

Idiomatic way to access `struct bp_sock`

```
int bp_release(struct socket* sock)
{
        struct sock* sk = sock->sk;
        struct bp_sock* bp =
bp_sk(sk);

    ...
}
```

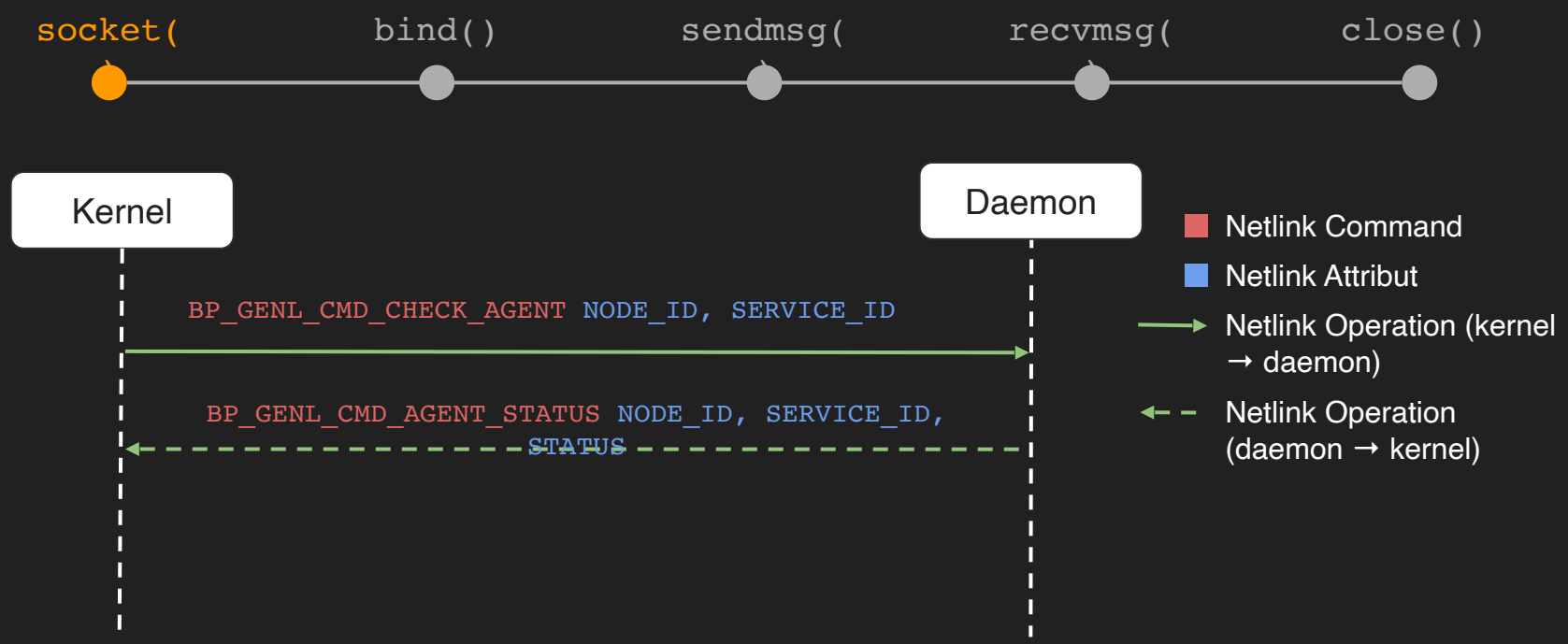
```
struct proto bp_proto = {
        .name = "BP",
        .owner = THIS_MODULE,
        .obj_size = sizeof(struct
bp_sock),
};
```

```
sk = sk_alloc(net, AF_BP, GFP_KERNEL, &bp_proto,
                    1);
```

```
#define bp_sk(ptr) container_of(ptr, struct bp_sock,
                        sk)
```

# ION Healthcheck Before Accepting `socket()`

## BP Socket Lifecycle

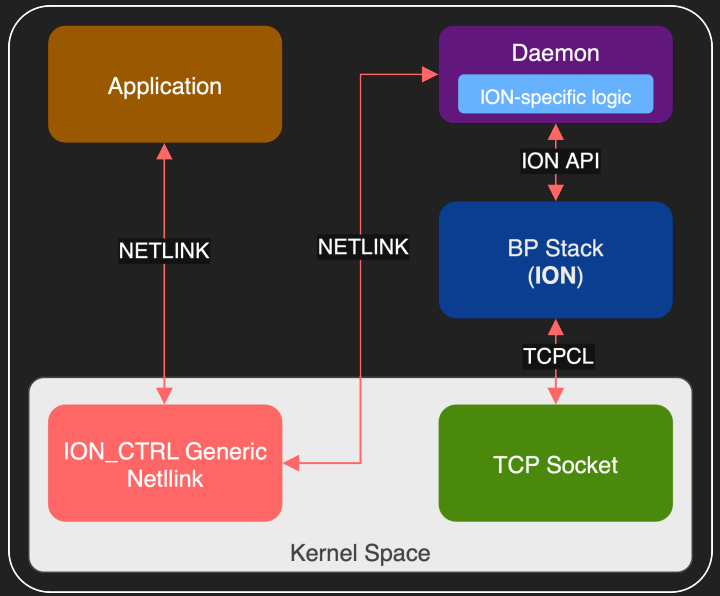


# Generic Netlink Family for ION configuration

Inspired by Netlink-based control interfaces like: WireGuard, nftables...

Example commands:

- ION_CMD_A_ENDPOINT
- ION_CMD_A_CONTACT
- ION_CMD_A_RANGE
- ION_CMD_A_PLAN
- ION_CMD_A_SHEME
- ....



# GitHub Runner with DTN Nodes

Enable real **BP bundle transmission** in GitHub Actions pipelines

Use a **custom self-hosted runner** with ION preinstalled

**Run end-to-end tests** using BP Socket inside the CI

# How to inject code into the kernel

There are a few options to extend kernel capabilities:

- Change kernel source code
- **Load kernel modules**
- Run eBPF programs

**Note:** Kernel modules are pieces of code that can be loaded and unloaded into the kernel upon demand.