“Is Upstream Really Enough?” Practical Realities of Using eBPF in Long-Term Supported Systems

東京 2025 LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DEC. 11-13, 2025

Who am I?

- My name is Kenta Tada
- CNCF End User Technical Advisory Board Member
- Organizer, Cloud Native Community Japan
- Organizer, eBPF Japan Community
- Speaker @ KubeCon Japan 2025, OSSummit 2025, LPC 2025
- The reviewer of the Japanese translation of the book "Learning eBPF" published by O'Reilly Japan

eBPF Japan Community

- We are active as a CNCJ Special Interest Group (SIG)
- We are introducing the internal implementation of the technology supporting the core of eBPF, through to its use cases.

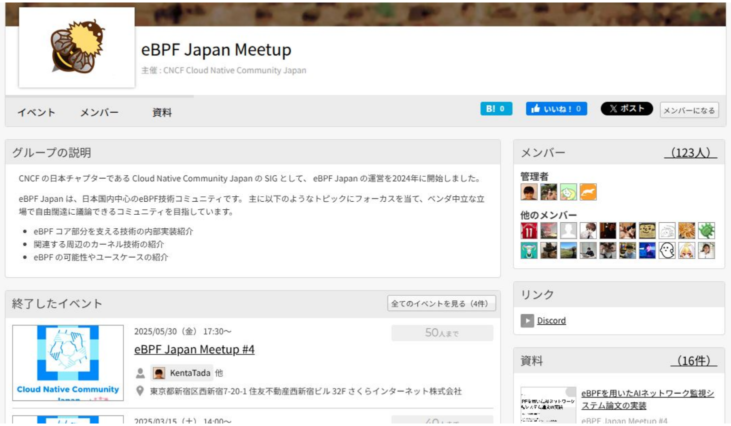
KubeCon + CloudNativeCon North America · Atlanta · Nov 10-13 · Register Now

Cloud Native Community Japan - eBPF Japan Meetup #4

Cloud Native Community Japan

May 30, 5:30 – 8:30 PM (GMT+9)

In-person event



eBPF Japan Meetup

主催 : CNCF Cloud Native Community Japan

イベント　メンバー　資料

グループの説明

CNCF の日本チャプターである Cloud Native Community Japan の SIG として、 eBPF Japan の運営を2024年に開始しました。

eBPF Japan は、日本国内中心のeBPF技術コミュニティです。 主に以下のようなトピックにフォーカスを当て、ベンダ中立な立場で自由闊達に議論できるコミュニティを目指しています。

- eBPF コア部分を支える技術の内部実装紹介
- 関連する周辺のカーネル技術の紹介
- eBPF の可能性やユースケースの紹介

メンバー　(123人)

管理者

他のメンバー

リンク

Discord

終了したイベント　全てのイベントを見る（4件）

2025/05/30（金） 17:30〜　50人まで

eBPF Japan Meetup #4

KentaTada 他

東京都新宿区西新宿7-20-1 住友不動産西新宿ビル 32F さくらインターネット株式会社

2025/03/15（土） 14:00〜

資料　(16件)

eBPFを用いたAIネットワーク監視システム論文の実装

eBPF Japan Meetup #4

Objective

- As eBPF becomes more widely deployed in production and embedded devices, vendors face inconsistent kernel capabilities and increasing maintenance costs. This talk aims to surface the real pains and explore structured solutions such as an **‘eBPF Embedded Profile’**.

Motivation (1/2)

- While eBPF has become a powerful foundation for observability, networking, and security tooling, its usability in long-lived systems (e.g., automotive, industrial-grade embedded) is severely hampered due to:
  - Kernel version divergence
  - Verifier regressions
  - User-space tooling vs. legacy kernel gaps
  - Security policies(Ex. banning bpf() syscall)

Motivation (2/2)

- Furthermore, as eBPF evolves to enable deeper kernel extensibility—evidenced by the recent integration of features like sched_ext into the mainline kernel
- The imperative to address its practical deployment challenges in LTS and embedded environments becomes even more critical.
- This marks a new phase where eBPF is not just for tooling, but a core mechanism for extending the kernel itself.

Requirements for embedded systems

- Maximizing Resource Efficiency
- Deterministic Real-time Communication
- Low Power Consumption
- High Security
- Longevity and Maintainability
- High Availability

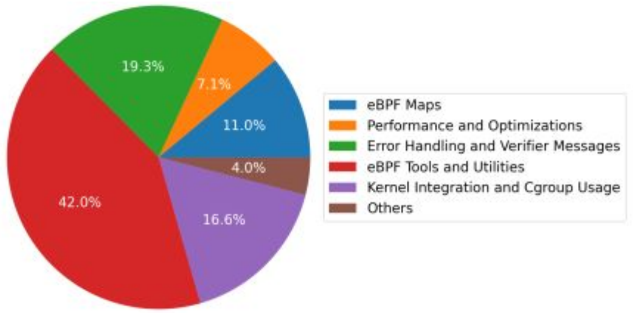
and so on...

The system requirements for **embedded devices vary widely**, ranging from industrial equipments to automotive applications.

In addition to those requirements

How to lower the barrier of entry

- The eBPF ecosystem is rapidly expanding, yet documentation has not kept pace with that.
- This mismatch has led to increasing developer confusion, **reflected in the rising number of eBPF Tools-related questions on Stack Overflow.**
- As a result, eBPF development remains challenging for newcomers, highlighting the need for structured education and up-to-date learning resources.



An Empirical Study on the Challenges of eBPF Application Development

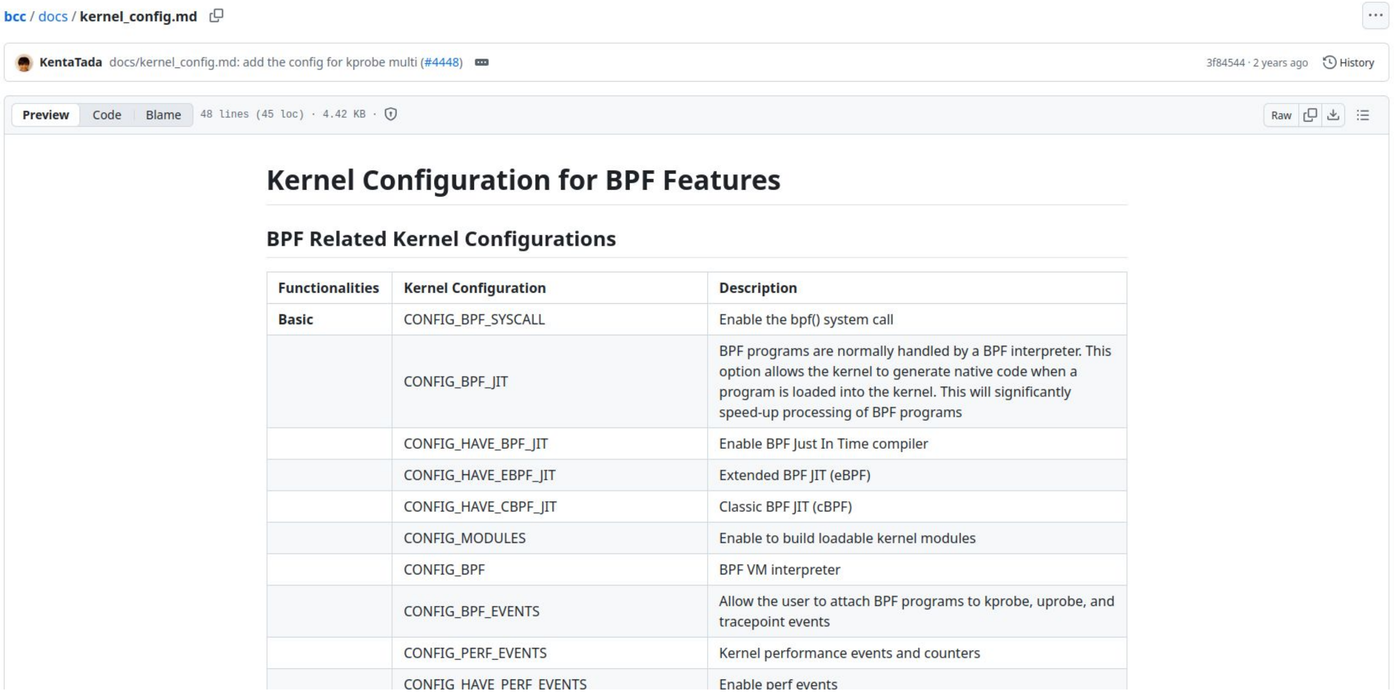
Challenges of eBPF

Items

- Configurations for embedded systems
- Stable ABIs
- Verifier Compatibility
- Security for embedded systems

Configurations for embedded systems

Configurations are difficult for end users...


bcc / docs / kernel_config.md

KentaTada docs/kernel_config.md: add the config for kprobe multi (#4448) 3f84544 · 2 years ago History

Kernel Configuration for BPF Features

BPF Related Kernel Configurations

Functionalities	Kernel Configuration	Description
Basic	CONFIG_BPF_SYSCALL	Enable the bpf() system call
	CONFIG_BPF_JIT	BPF programs are normally handled by a BPF interpreter. This option allows the kernel to generate native code when a program is loaded into the kernel. This will significantly speed-up processing of BPF programs
	CONFIG_HAVE_BPF_JIT	Enable BPF Just In Time compiler
	CONFIG_HAVE_EBPF_JIT	Extended BPF JIT (eBPF)
	CONFIG_HAVE_CBPF_JIT	Classic BPF JIT (cBPF)
	CONFIG_MODULES	Enable to build loadable kernel modules
	CONFIG_BPF	BPF VM interpreter
	CONFIG_BPF_EVENTS	Allow the user to attach BPF programs to kprobe, uprobe, and tracepoint events
	CONFIG_PERF_EVENTS	Kernel performance events and counters
	CONFIG_HAVE_PERF_EVENTS	Enable perf events

東京 2025 LINUX PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

Real world experiences

- When it comes to some selective eBPF features like CONFIG_BPFILTER, we can disable that.
- But most of end users don't know what is actually needed or not.
- Ex1: Enable CONFIG_BPF_KPROBE_OVERRIDE to use bpf_override_return() for the use case of chaos engineering tools
- Ex2: Restrict bpf_probe_write_user() using LSM Lockdown when you want to make your system secure.

Configurations for embedded systems

- We already have a kernel config guide for BPF in bcc, which solves a real-world pain: "Engineers who configure kernels often don't fully understand BPF features."
- For the Embedded Profile, we can extend this idea by mapping embedded requirements and security constraints onto the BPF feature set.
- This provides a practical checklist for vendors and dramatically reduces integration cost across the industry.

Stable ABIs

Stable ABIs are needed

- Tracing facilities
  - Kernel internals depending on eBPF programs are not stable ABIs even if tracepoints.
  - Regressions occasionally occur in libbpf-tools.
- kfuncs
  - kfuncs are explicitly known to be subject to changes.
  - However, it is difficult for eBPF program developers in the field to keep up with the small changes.
  - Example: developers must track capability changes per kernel version to use specific kfuncs.

Verifier Compatibility

Verifier Incompatibility

- **The Verifier is great!** However, as the kernel version increases, the implementation changes, which causes compatibility issues for eBPF programs.
- Backporting is so difficult
  - *"It is pretty much impossible to keep verifier spec unchanged during the lifetime of a kernel product. Verifier spec stability goes against bug/CVE fixing, so at some-point you have to choose between one."* by Shung-Hsi Yu
  - Should we modularize the verifier?

Security for embedded systems

Security concerns

- Some organizations disable bpf() entirely due to policy.
  - **Signed eBPF** could significantly help address organizational security policies.
- Software supply chain security and compliance is important
  - Every part of it is compliant with licensing policies and traceable in a bill of materials.
  - See eBPF Security Threat Model
- When it comes to the automotive-graded systems, **the debug information like BTF should be removed.**
  - **Use min_core_btf to reduce both the attack surface and size**

System Hardening from AGL Documentation

- Kernel debug symbols
  - Debug symbols should always be removed from production kernels as they provide a lot of information to attackers.
- Disable Kprobes
- Disable BPF JIT

and so on

https://docs.automotivelinux.org/

While security is essential, these policies directly conflict with eBPF requirements.

Proposed Concept: eBPF Embedded Profile

eBPF Embedded Profile

- A defined subset of eBPF features, APIs, and tooling practices tailored for long-term use in embedded and LTS systems.
- This profile may include:
  - Appropriate Kernel Configs for each use case
  - Versioned and stable kfunc sets
  - Verifier constraints
  - Toolchain recommendations
    - Ex. compiler flags, libbpf versions, etc.
  - Optional certification or conformance tests like LTP

Expected Benefits

- **Significantly** reduced backporting burden
- **Greatly** improved predictable behavior across LTS kernels
- **Accelerated** cross-vendor collaboration
- **Substantially enhanced** developer confidence for long-term use, leading to wider adoption

Ex. Assessing Production-Grade Suitability

Aspect	Description	API explanation
✅ **Runtime Stability**	Whether the API operates reliably under long uptime and constrained resources	`bpf_ringbuf_reserve()` may fail
✅ **Real-Time Compatibility**	Sleepable BPF programs may sleep, and if this is not carefully considered, they can severely impact the latency.	`bpf_copy_from_user()` requires sleepable BPF
✅ **Power & Thermal Efficiency**	Whether the design avoids high-frequency polling	`perf_event_output` can cause busy loops
✅ **Security & Vulnerability History**	Whether known CVEs or past misuse exist	`bpf_probe_write_user()` has been abused in real-world exploits

Collaboration

To gather the real-world requirements

eBPF FOUNDATION

CLOUD NATIVE COMPUTING FOUNDATION

AUTOMOTIVE GRADE LINUX

CIVIL INFRASTRUCTURE PLATFORM
INDUSTRIAL GRADE LINUX

東京 2025 LINUX PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

Next actions

- Collect real-world production requirements
- Talk with related foundations
  - eBPF Foundation : eBPF Embedded Profile
  - CNCF : Edge Native Application
  - AGL : Real world use cases and security requirements
  - CIP : Long-Term Support and upstream contribution
- Make my proposed concept great!!

Key takeaways

- Upstream innovation is amazing, but not enough for long-lived embedded systems.
- To ensure eBPF becomes a stable foundation for the next decade, we need structured collaboration
- A standardized Embedded Profile could be one approach.