BPF (eBPF) for Packet Filtering and Kernel Tracing

eBPF (Extended Berkeley Packet Filter) is a revolutionary technology in the Linux kernel that allows safe and efficient execution of user-defined programs in kernel space. Initially designed for packet filtering (classic BPF), eBPF has evolved into a powerful sandboxed virtual machine inside the Linux kernel that supports networking, security, tracing, and performance profiling.

🔍 Use Cases of eBPF

Category	Example Use Cases
Packet Filtering	Firewall rules (e.g., XDP), DDoS mitigation
Tracing & Observability	System call tracing, performance monitoring (`bcc`, `bpftrace`)
Security	Syscall filtering, process activity monitoring (`seccomp`, `Tetragon`)
Performance Tuning	CPU/memory profiling, disk I/O latency debugging
Networking	Load balancing (Cilium), deep packet inspection

🔎 eBPF for Kernel Tracing

eBPF can attach probes to various kernel or user-space events:

- **kprobes / kretprobes:** Attach to kernel function entry/exit.
- **uprobes:** Attach to user-space binary functions.
- **tracepoints:** Static instrumentation points in kernel code.
- **perf events:** CPU performance counters.

Tools & Frameworks:

- **BCC (BPF Compiler Collection):** Python-based front end for writing and running eBPF tracing programs.
- **bpftrace:** High-level tracing language similar to `awk`, great for one-liners.
- **Perf / SystemTap:** Older tools, partially superseded by eBPF.

Example: Trace open syscalls with `bpftrace`

```
bpftrace -e 'tracepoint:syscalls:sys_enter_openat { printf("%s opened
```

📦 Real-World Applications

- **Cilium:** Kubernetes CNI plugin that uses eBPF for load balancing, network policy enforcement, and observability.
- **Falco:** Runtime security tool using eBPF to detect suspicious syscalls.
- **Facebook / Netflix:** Use eBPF extensively for performance debugging and latency tracing at scale.

🔧 Development Toolchain

- **LLVM/Clang:** For compiling C code into eBPF bytecode.
- **libbpf:** C API to load and interact with eBPF programs.
- **bpftool:** CLI for inspecting and managing eBPF programs.
- **CO-RE (Compile Once – Run Everywhere):** Mechanism to write portable eBPF programs that adapt to kernel versions.

🧠 Summary

Feature	Description
Packet Filtering	Drop or reroute packets in kernel space (XDP, tc)
Tracing	

Feature	Description
	Kernel/user-space visibility without recompiling kernel
Security	Runtime process and syscall monitoring
Performance	Minimal overhead; safe and fast execution

eBPF represents a paradigm shift in systems programming by safely extending kernel capabilities **without writing kernel modules**, enabling deep observability, fine-grained control, and performance that rivals native code.