Running BPF programs from userspace

This document describes the `BPF_PROG_RUN` facility for running BPF programs from userspace.

Overview

The `BPF_PROG_RUN` command can be used through the `bpf()` syscall to execute a BPF program in the kernel and return the results to userspace. This can be used to unit test BPF programs against user-supplied context objects, and as way to explicitly execute programs in the kernel for their side effects. The command was previously named `BPF_PROG_TEST_RUN`, and both constants continue to be defined in the UAPI header, aliased to the same value.

The `BPF_PROG_RUN` command can be used to execute BPF programs of the following types:

- `BPF_PROG_TYPE_SOCKET_FILTER`
- `BPF_PROG_TYPE_SCHED_CLS`
- `BPF_PROG_TYPE_SCHED_ACT`
- `BPF_PROG_TYPE_XDP`
- `BPF_PROG_TYPE_SK_LOOKUP`
- `BPF_PROG_TYPE_CGROUP_SKB`
- `BPF_PROG_TYPE_LWT_IN`
- `BPF_PROG_TYPE_LWT_OUT`
- `BPF_PROG_TYPE_LWT_XMIT`
- `BPF_PROG_TYPE_LWT_SEG6LOCAL`
- `BPF_PROG_TYPE_FLOW_DISSECTOR`
- `BPF_PROG_TYPE_STRUCT_OPS`
- `BPF_PROG_TYPE_RAW_TRACEPOINT`
- `BPF_PROG_TYPE_SYSCALL`

When using the `BPF_PROG_RUN` command, userspace supplies an input context object and (for program types operating on network packets) a buffer containing the packet data that the BPF program will operate on. The kernel will then execute the program and return the results to userspace. Note that programs will not have any side effects while being run in this mode; in particular, packets will not actually be redirected or dropped, the program return code will just be returned to userspace. A separate mode for live execution of XDP programs is provided, documented separately below.

Running XDP programs in "live frame mode"

The `BPF_PROG_RUN` command has a separate mode for running live XDP programs, which can be used to execute XDP programs in a way where packets will actually be processed by the kernel after the execution of the XDP program as if they arrived on a physical interface. This mode is activated by setting the `BPF_F_TEST_XDP_LIVE_FRAMES` flag when supplying an XDP program to `BPF_PROG_RUN`.

The live packet mode is optimised for high performance execution of the supplied XDP program many times (suitable for, e.g., running as a traffic generator), which means the semantics are not quite as straight-forward as the regular test run mode. Specifically:

- When executing an XDP program in live frame mode, the result of the execution will not be returned to userspace; instead, the kernel will perform the operation indicated by the program's return code (drop the packet, redirect it, etc). For this reason, setting the `data_out` or `ctx_out` attributes in the syscall parameters when running in this mode will be rejected. In addition, not all failures will be reported back to userspace directly; specifically, only fatal errors in setup or during execution (like memory allocation errors) will halt execution and return an error. If an error occurs in packet processing, like a failure to redirect to a given interface, execution will continue with the next repetition; these errors can be detected via the same trace points as for regular XDP programs.
- Userspace can supply an ifindex as part of the context object, just like in the regular (non-live) mode. The XDP program will be executed as though the packet arrived on this interface; i.e., the `ingress_ifindex` of the context object will point to that interface. Furthermore, if the XDP program returns `XDP_PASS`, the packet will be injected into the kernel networking stack as though it arrived on that ifindex, and if it returns `XDP_TX`, the packet will be transmitted *out* of that same interface. Do note, though, that because the program execution is not happening in driver context, an `XDP_TX` is actually turned into the same action as an `XDP_REDIRECT` to that same interface (i.e., it will only work if the driver has support for the `ndo_xdp_xmit` driver op).
- When running the program with multiple repetitions, the execution will happen in batches. The batch size defaults to 64 packets (which is same as the maximum NAPI receive batch size), but can be specified by userspace through the `batch_size` parameter, up to a maximum of 256 packets. For each batch, the kernel executes the XDP program repeatedly, each invocation getting a separate copy of the packet data. For each repetition, if the program drops the packet, the data page is immediately recycled (see below). Otherwise, the packet is buffered until the end of the batch, at which point all packets

buffered this way during the batch are transmitted at once.

- When setting up the test run, the kernel will initialise a pool of memory pages of the same size as the batch size. Each memory page will be initialised with the initial packet data supplied by userspace at `BPF_PROG_RUN` invocation. When possible, the pages will be recycled on future program invocations, to improve performance. Pages will generally be recycled a full batch at a time, except when a packet is dropped (by return code or because of, say, a redirection error), in which case that page will be recycled immediately. If a packet ends up being passed to the regular networking stack (because the XDP program returns `XDP_PASS`, or because it ends up being redirected to an interface that injects it into the stack), the page will be released and a new one will be allocated when the pool is empty.

  When recycling, the page content is not rewritten; only the packet boundary pointers (`data`, `data_end` and `data_meta`) in the context object will be reset to the original values. This means that if a program rewrites the packet contents, it has to be prepared to see either the original content or the modified version on subsequent invocations.