HOWTO interact with BPF subsystem

This document provides information for the BPF subsystem about various workflows related to reporting bugs, submitting patches, and queueing patches for stable kernels.

For general information about submitting patches, please refer to Documentation/process/. This document only describes additional specifics related to BPF.

Reporting bugs

Q: How do I report bugs for BPF kernel code?

A: Since all BPF kernel development as well as bpftool and iproute2 BPF loader development happens through the bpf kernel mailing list, please report any found issues around BPF to the following mailing list:

> bpf@vger.kernel.org

This may also include issues related to XDP, BPF tracing, etc.

Given netdev has a high volume of traffic, please also add the BPF maintainers to Cc (from kernel `MAINTAINERS` file):

- Alexei Starovoitov <ast@kernel.org>
- Daniel Borkmann <daniel@iogearbox.net>

In case a buggy commit has already been identified, make sure to keep the actual commit authors in Cc as well for the report. They can typically be identified through the kernel's git tree.

Please do NOT report BPF issues to bugzilla.kernel.org since it is a guarantee that the reported issue will be overlooked.

Submitting patches

Q: To which mailing list do I need to submit my BPF patches?

A: Please submit your BPF patches to the bpf kernel mailing list:

bpf@vger.kernel.org

In case your patch has changes in various different subsystems (e.g. networking, tracing, security, etc), make sure to Cc the related kernel mailing lists and maintainers from there as well, so they are able to review the changes and provide their Acked-by's to the patches.

Q: Where can I find patches currently under discussion for BPF subsystem?

A: All patches that are Cc'ed to netdev are queued for review under netdev patchwork project:

https://patchwork.kernel.org/project/netdevbpf/list/

Those patches which target BPF, are assigned to a 'bpf' delegate for further processing from BPF maintainers. The current queue with patches under review can be found at:

https://patchwork.kernel.org/project/netdevbpf/list/?delegate=121173

Once the patches have been reviewed by the BPF community as a whole and approved by the BPF maintainers, their status in patchwork will be changed to 'Accepted' and the submitter will be notified by mail. This means that the patches look good from a BPF perspective and have been applied to one of the two BPF kernel trees.

In case feedback from the community requires a respin of the patches, their status in patchwork will be set to 'Changes Requested', and purged from the current review queue. Likewise for cases where patches would get rejected or are not applicable to the BPF trees (but assigned to the 'bpf' delegate).

Q: How do the changes make their way into Linux?

A: There are two BPF kernel trees (git repositories). Once patches have been accepted by the BPF maintainers, they will be applied to one of the two BPF trees:

- https://git.kernel.org/pub/scm/linux/kernel/git/bpf/bpf.git/
- https://git.kernel.org/pub/scm/linux/kernel/git/bpf/bpf-next.git/

The bpf tree itself is for fixes only, whereas bpf-next for features, cleanups or other kind of improvements ("next-like" content). This is analogous to net and net-next trees for networking. Both bpf and bpf-next will only have a master branch in order to simplify against which branch patches should get rebased to.

Accumulated BPF patches in the bpf tree will regularly get pulled into the net kernel tree. Likewise, accumulated BPF patches accepted into the bpf-next tree will make their way into net-next tree. net and net-next are both run by David S. Miller. From there, they will go into the kernel mainline tree run by Linus Torvalds. To read up on the process of net and net-next being merged into the mainline tree, see the :ref:`netdev-FAQ`

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\bpf\(linux-master)(Documentation)(bpf)bpf_devel_QA.rst`, **line 98**); *backlink*
>
> Unknown interpreted text role "ref".

Occasionally, to prevent merge conflicts, we might send pull requests to other trees (e.g. tracing) with a small subset of the patches, but net and net-next are always the main trees targeted for integration.

The pull requests will contain a high-level summary of the accumulated patches and can be searched on netdev kernel mailing list through the following subject lines (`yyyy-mm-dd` is the date of the pull request):

```
pull-request: bpf yyyy-mm-dd
pull-request: bpf-next yyyy-mm-dd
```

Q: How do I indicate which tree (bpf vs. bpf-next) my patch should be applied to?

A: The process is the very same as described in the :ref:`netdev-FAQ`, so please read up on it. The subject line must indicate whether the patch is a fix or rather "next-like" content in order to let the maintainers know whether it is targeted at bpf or bpf-next.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\bpf\(linux-master)(Documentation)(bpf)bpf_devel_QA.rst`, **line 123**); *backlink*
>
> Unknown interpreted text role "ref".

For fixes eventually landing in bpf -> net tree, the subject must look like:

```
git format-patch --subject-prefix='PATCH bpf' start..finish
```

For features/improvements/etc that should eventually land in bpf-next -> net-next, the subject must look like:

```
git format-patch --subject-prefix='PATCH bpf-next' start..finish
```

If unsure whether the patch or patch series should go into bpf or net directly, or bpf-next or net-next directly, it is not a problem either if the subject line says net or net-next as target. It is eventually up to the maintainers to do the delegation of the patches.

If it is clear that patches should go into bpf or bpf-next tree, please make sure to rebase the patches against those trees in order to reduce potential conflicts.

In case the patch or patch series has to be reworked and sent out again in a second or later revision, it is also required to add a version number (`v2`, `v3`, ...) into the subject prefix:

```
git format-patch --subject-prefix='PATCH bpf-next v2' start..finish
```

When changes have been requested to the patch series, always send the whole patch series again with the feedback incorporated (never send individual diffs on top of the old series).

Q: What does it mean when a patch gets applied to bpf or bpf-next tree?

A: It means that the patch looks good for mainline inclusion from a BPF point of view.

Be aware that this is not a final verdict that the patch will automatically get accepted into net or net-next trees eventually:

On the bpf kernel mailing list reviews can come in at any point in time. If discussions around a patch conclude that they cannot get included as-is, we will either apply a follow-up fix or drop them from the trees entirely. Therefore, we also reserve to rebase the trees when deemed necessary. After all, the purpose of the tree is to:

i. accumulate and stage BPF patches for integration into trees like net and net-next, and
ii. run extensive BPF test suite and workloads on the patches before they make their way any further.

Once the BPF pull request was accepted by David S. Miller, then the patches end up in net or net-next tree, respectively, and make their way from there further into mainline. Again, see the :ref:`netdev-FAQ` for additional information e.g. on how often they are merged to mainline.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\bpf\(linux-master)(Documentation)(bpf)bpf_devel_QA.rst`, **line 179**); *backlink*
>
> Unknown interpreted text role "ref".

Q: How long do I need to wait for feedback on my BPF patches?

A: We try to keep the latency low. The usual time to feedback will be around 2 or 3 business days. It may vary depending on the complexity of changes and current patch load.

Q: How often do you send pull requests to major kernel trees like net or net-next?

A: Pull requests will be sent out rather often in order to not accumulate too many patches in bpf or bpf-next.

As a rule of thumb, expect pull requests for each tree regularly at the end of the week. In some cases pull requests could additionally come also in the middle of the week depending on the current patch load or urgency.

Q: Are patches applied to bpf-next when the merge window is open?

A: For the time when the merge window is open, bpf-next will not be processed. This is roughly analogous to net-next patch processing, so feel free to read up on the :ref:`netdev-FAQ` about further details.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\bpf\(linux-master)(Documentation)(bpf)bpf_devel_QA.rst`, **line 204**); *backlink*
>
> Unknown interpreted text role "ref".

During those two weeks of merge window, we might ask you to resend your patch series once bpf-next is open again. Once Linus released a `v*-rc1` after the merge window, we continue processing of bpf-next.

For non-subscribers to kernel mailing lists, there is also a status page run by David S. Miller on net-next that provides guidance:

http://vger.kernel.org/~davem/net-next.html

Q: Verifier changes and test cases

Q: I made a BPF verifier change, do I need to add test cases for BPF kernel selftests?

A: If the patch has changes to the behavior of the verifier, then yes, it is absolutely necessary to add test cases to the BPF kernel selftests suite. If they are not present and we think they are needed, then we might ask for them before accepting any changes.

In particular, test_verifier.c is tracking a high number of BPF test cases, including a lot of corner cases that LLVM BPF back end may generate out of the restricted C code. Thus, adding test cases is absolutely crucial to make sure future changes do not accidentally affect prior use-cases. Thus, treat those test cases as: verifier behavior that is not tracked in test_verifier.c could potentially be subject to change.

Q: samples/bpf preference vs selftests?

Q: When should I add code to `samples/bpf/` and when to BPF kernel selftests?

A: In general, we prefer additions to BPF kernel selftests rather than `samples/bpf/`. The rationale is very simple: kernel selftests are regularly run by various bots to test for kernel regressions.

The more test cases we add to BPF selftests, the better the coverage and the less likely it is that those could accidentally break. It is not that BPF kernel selftests cannot demo how a specific feature can be used.

That said, `samples/bpf/` may be a good place for people to get started, so it might be advisable that simple demos of features could go into `samples/bpf/`, but advanced functional and corner-case testing rather into kernel selftests.

If your sample looks like a test case, then go for BPF kernel selftests instead!

Q: When should I add code to the bpftool?

A: The main purpose of bpftool (under tools/bpf/bpftool/) is to provide a central user space tool for debugging and introspection of BPF programs and maps that are active in the kernel. If UAPI changes related to BPF enable for dumping additional information of programs or maps, then bpftool should be extended as well to support dumping them.

Q: When should I add code to iproute2's BPF loader?

A: For UAPI changes related to the XDP or tc layer (e.g. `cls_bpf`), the convention is that those control-path related changes are added to iproute2's BPF loader as well from user space side. This is not only useful to have UAPI changes properly designed to be usable, but also to make those changes available to a wider user base of major downstream distributions.

Q: Do you accept patches as well for iproute2's BPF loader?

A: Patches for the iproute2's BPF loader have to be sent to:

> netdev@vger.kernel.org

While those patches are not processed by the BPF kernel maintainers, please keep them in Cc as well, so they can be reviewed.

The official git repository for iproute2 is run by Stephen Hemminger and can be found at:

> https://git.kernel.org/pub/scm/linux/kernel/git/shemminger/iproute2.git/

The patches need to have a subject prefix of '`[PATCH iproute2 master]`' or '`[PATCH iproute2 net-next]`'. '`master`' or '`net-next`' describes the target branch where the patch should be applied to. Meaning, if kernel changes went into the net-next kernel tree, then the related iproute2 changes need to go into the iproute2 net-next branch, otherwise they can be targeted at master branch. The iproute2 net-next branch will get merged into the master branch after the current iproute2 version from master has been released.

Like BPF, the patches end up in patchwork under the netdev project and are delegated to 'shemminger' for further processing:

> http://patchwork.ozlabs.org/project/netdev/list/?delegate=389

Q: What is the minimum requirement before I submit my BPF patches?

A: When submitting patches, always take the time and properly test your patches *prior* to submission. Never rush them! If maintainers find that your patches have not been properly tested, it is a good way to get them grumpy. Testing patch submissions is a hard requirement!

Note, fixes that go to bpf tree *must* have a `Fixes:` tag included. The same applies to fixes that target bpf-next, where the affected commit is in net-next (or in some cases bpf-next). The `Fixes:` tag is crucial in order to identify follow-up commits and tremendously helps for people having to do backporting, so it is a must have!

We also don't accept patches with an empty commit message. Take your time and properly write up a high quality commit message, it is essential!

Think about it this way: other developers looking at your code a month from now need to understand *why* a certain change has been done that way, and whether there have been flaws in the analysis or assumptions that the original author did. Thus providing a proper rationale and describing the use-case for the changes is a must.

Patch submissions with >1 patch must have a cover letter which includes a high level description of the series. This high level summary will then be placed into the merge commit by the BPF maintainers such that it is also accessible from the git log for future reference.

Q: Features changing BPF JIT and/or LLVM

Q: What do I need to consider when adding a new instruction or feature that would require BPF JIT and/or LLVM integration as well?

A: We try hard to keep all BPF JITs up to date such that the same user experience can be guaranteed when running BPF programs on different architectures without having the program punt to the less efficient interpreter in case the in-kernel BPF JIT is enabled.

If you are unable to implement or test the required JIT changes for certain architectures, please work together with the related BPF JIT developers in order to get the feature implemented in a timely manner. Please refer to the git log (`arch/*/net/`) to locate the necessary people for helping out.

Also always make sure to add BPF test cases (e.g. test_bpf.c and test_verifier.c) for new instructions, so that they can receive broad test coverage and help run-time testing the various BPF JITs.

In case of new BPF instructions, once the changes have been accepted into the Linux kernel, please implement support into LLVM's BPF back end. See LLVM section below for further information.

Stable submission

Q: I need a specific BPF commit in stable kernels. What should I do?

A: In case you need a specific fix in stable kernels, first check whether the commit has already been applied in the related `linux-*.y` branches:

> https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git/

If not the case, then drop an email to the BPF maintainers with the netdev kernel mailing list in Cc and ask for the fix to be queued up:

> netdev@vger.kernel.org

The process in general is the same as on netdev itself, see also the :ref:`netdev-FAQ`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\bpf\(linux-master)(Documentation)(bpf)bpf_devel_QA.rst, line 369); *backlink***
>
> Unknown interpreted text role "ref".

Q: Do you also backport to kernels not currently maintained as stable?

A: No. If you need a specific BPF commit in kernels that are currently not maintained by the stable maintainers, then you are on your own.

The current stable and longterm stable kernels are all listed here:

> https://www.kernel.org/

Q: The BPF patch I am about to submit needs to go to stable as well

What should I do?

A: The same rules apply as with netdev patch submissions in general, see the :ref:`netdev-FAQ`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\bpf\(linux-master)(Documentation)(bpf)bpf_devel_QA.rst, line 385); *backlink***
>
> Unknown interpreted text role "ref".

Never add "`Cc: stable@vger.kernel.org`" to the patch description, but ask the BPF maintainers to queue the patches instead. This can be done with a note, for example, under the `---` part of the patch which does not go into the git log. Alternatively, this can be done as a simple request by mail instead.

Q: Queue stable patches

Q: Where do I find currently queued BPF patches that will be submitted to stable?

A: Once patches that fix critical bugs got applied into the bpf tree, they are queued up for stable submission under:

http://patchwork.ozlabs.org/bundle/bpf/stable/?state=*

They will be on hold there at minimum until the related commit made its way into the mainline kernel tree.

After having been under broader exposure, the queued patches will be submitted by the BPF maintainers to the stable maintainers.

Testing patches

Q: How to run BPF selftests

A: After you have booted into the newly compiled kernel, navigate to the BPF selftests suite in order to test BPF functionality (current working directory points to the root of the cloned git tree):

```
$ cd tools/testing/selftests/bpf/
$ make
```

To run the verifier tests:

```
$ sudo ./test_verifier
```

The verifier tests print out all the current checks being performed. The summary at the end of running all tests will dump information of test successes and failures:

```
Summary: 418 PASSED, 0 FAILED
```

In order to run through all BPF selftests, the following command is needed:

```
$ sudo make run_tests
```

See the kernels selftest Documentation/dev-tools/kselftest.rst document for further documentation.

To maximize the number of tests passing, the .config of the kernel under test should match the config file fragment in tools/testing/selftests/bpf as closely as possible.

Finally to ensure support for latest BPF Type Format features - discussed in Documentation/bpf/btf.rst - pahole version 1.16 is required for kernels built with CONFIG_DEBUG_INFO_BTF=y. pahole is delivered in the dwarves package or can be built from source at

https://github.com/acmel/dwarves

pahole starts to use libbpf definitions and APIs since v1.13 after the commit 21507cd3e97b ("pahole: add libbpf as submodule under lib/bpf"). It works well with the git repository because the libbpf submodule will use "git submodule update --init --recursive" to update.

Unfortunately, the default github release source code does not contain libbpf submodule source code and this will cause build issues, the tarball from https://git.kernel.org/pub/scm/devel/pahole/pahole.git/ is same with github, you can get the source tarball with corresponding libbpf submodule codes from

https://fedorapeople.org/~acme/dwarves

Some distros have pahole version 1.16 packaged already, e.g. Fedora, Gentoo.

Q: Which BPF kernel selftests version should I run my kernel against?

A: If you run a kernel `xyz`, then always run the BPF kernel selftests from that kernel `xyz` as well. Do not expect that the BPF selftest from the latest mainline tree will pass all the time.

In particular, test_bpf.c and test_verifier.c have a large number of test cases and are constantly updated with new BPF test sequences, or existing ones are adapted to verifier changes e.g. due to verifier becoming smarter and being able to better track certain things.

LLVM

Q: Where do I find LLVM with BPF support?

A: The BPF back end for LLVM is upstream in LLVM since version 3.7.1.

All major distributions these days ship LLVM with BPF back end enabled, so for the majority of use-cases it is not required to compile LLVM by hand anymore, just install the distribution provided package.

LLVM's static compiler lists the supported targets through `llc --version`, make sure BPF targets are listed. Example:

```
$ llc --version
LLVM (http://llvm.org/):
  LLVM version 10.0.0
  Optimized build.
  Default target: x86_64-unknown-linux-gnu
```

```
  Host CPU: skylake

  Registered Targets:
    aarch64    - AArch64 (little endian)
    bpf        - BPF (host endian)
    bpfeb      - BPF (big endian)
    bpfel      - BPF (little endian)
    x86        - 32-bit X86: Pentium-Pro and above
    x86-64     - 64-bit X86: EM64T and AMD64
```

For developers in order to utilize the latest features added to LLVM's BPF back end, it is advisable to run the latest LLVM releases. Support for new BPF kernel features such as additions to the BPF instruction set are often developed together.

All LLVM releases can be found at: http://releases.llvm.org/

Q: Got it, so how do I build LLVM manually anyway?

A: We recommend that developers who want the fastest incremental builds use the Ninja build system, you can find it in your system's package manager, usually the package is ninja or ninja-build.

You need ninja, cmake and gcc-c++ as build requisites for LLVM. Once you have that set up, proceed with building the latest LLVM and clang version from the git repositories:

```
$ git clone https://github.com/llvm/llvm-project.git
$ mkdir -p llvm-project/llvm/build
$ cd llvm-project/llvm/build
$ cmake .. -G "Ninja" -DLLVM_TARGETS_TO_BUILD="BPF;X86" \
           -DLLVM_ENABLE_PROJECTS="clang"    \
           -DCMAKE_BUILD_TYPE=Release        \
           -DLLVM_BUILD_RUNTIME=OFF
$ ninja
```

The built binaries can then be found in the build/bin/ directory, where you can point the PATH variable to.

Set `-DLLVM_TARGETS_TO_BUILD` equal to the target you wish to build, you will find a full list of targets within the llvm-project/llvm/lib/Target directory.

Q: Reporting LLVM BPF issues

Q: Should I notify BPF kernel maintainers about issues in LLVM's BPF code generation back end or about LLVM generated code that the verifier refuses to accept?

A: Yes, please do!

LLVM's BPF back end is a key piece of the whole BPF infrastructure and it ties deeply into verification of programs from the kernel side. Therefore, any issues on either side need to be investigated and fixed whenever necessary.

Therefore, please make sure to bring them up at netdev kernel mailing list and Cc BPF maintainers for LLVM and kernel bits:

- Yonghong Song <yhs@fb.com>
- Alexei Starovoitov <ast@kernel.org>
- Daniel Borkmann <daniel@iogearbox.net>

LLVM also has an issue tracker where BPF related bugs can be found:

https://bugs.llvm.org/buglist.cgi?quicksearch=bpf

However, it is better to reach out through mailing lists with having maintainers in Cc.

Q: New BPF instruction for kernel and LLVM

Q: I have added a new BPF instruction to the kernel, how can I integrate it into LLVM?

A: LLVM has a `-mcpu` selector for the BPF back end in order to allow the selection of BPF instruction set extensions. By default the `generic` processor target is used, which is the base instruction set (v1) of BPF.

LLVM has an option to select `-mcpu=probe` where it will probe the host kernel for supported BPF instruction set extensions and selects the optimal set automatically.

For cross-compilation, a specific version can be select manually as well

```
$ llc -march bpf -mcpu=help
Available CPUs for this target:

  generic - Select the generic processor.
  probe   - Select the probe processor.
  v1      - Select the v1 processor.
  v2      - Select the v2 processor.
[...]
```

Newly added BPF instructions to the Linux kernel need to follow the same scheme, bump the instruction set version and implement probing for the extensions such that `-mcpu=probe` users can benefit from the optimization transparently when upgrading their kernels.

If you are unable to implement support for the newly added BPF instruction please reach out to BPF developers for help.

By the way, the BPF kernel selftests run with `-mcpu=probe` for better test coverage.

Q: clang flag for target bpf?

Q: In some cases clang flag `-target bpf` is used but in other cases the default clang target, which matches the underlying architecture, is used. What is the difference and when I should use which?

A: Although LLVM IR generation and optimization try to stay architecture independent, `-target <arch>` still has some impact on generated code:

- BPF program may recursively include header file(s) with file scope inline assembly codes. The default target can handle this well, while `bpf` target may fail if bpf backend assembler does not understand these assembly codes, which is true in most cases.
- When compiled without `-g`, additional elf sections, e.g., .eh_frame and .rela.eh_frame, may be present in the object file with default target, but not with `bpf` target.
- The default target may turn a C switch statement into a switch table lookup and jump operation. Since the switch table is placed in the global readonly section, the bpf program will fail to load. The bpf target does not support switch table optimization. The clang option `-fno-jump-tables` can be used to disable switch table generation.
- For clang `-target bpf`, it is guaranteed that pointer or long / unsigned long types will always have a width of 64 bit, no matter whether underlying clang binary or default target (or kernel) is 32 bit. However, when native clang target is used, then it will compile these types based on the underlying architecture's conventions, meaning in case of 32 bit architecture, pointer or long / unsigned long types e.g. in BPF context structure will have width of 32 bit while the BPF LLVM back end still operates in 64 bit. The native target is mostly needed in tracing for the case of walking `pt_regs` or other kernel structures where CPU's register width matters. Otherwise, `clang -target bpf` is generally recommended.

You should use default target when:

- Your program includes a header file, e.g., ptrace.h, which eventually pulls in some header files containing file scope host assembly codes.
- You can add `-fno-jump-tables` to work around the switch table issue.

Otherwise, you can use `bpf` target. Additionally, you *must* use bpf target when:

- Your program uses data structures with pointer or long / unsigned long types that interface with BPF helpers or context data structures. Access into these structures is verified by the BPF verifier and may result in verification failures if the native architecture is not aligned with the BPF architecture, e.g. 64-bit. An example of this is BPF_PROG_TYPE_SK_MSG require `-target bpf`

Happy BPF hacking!