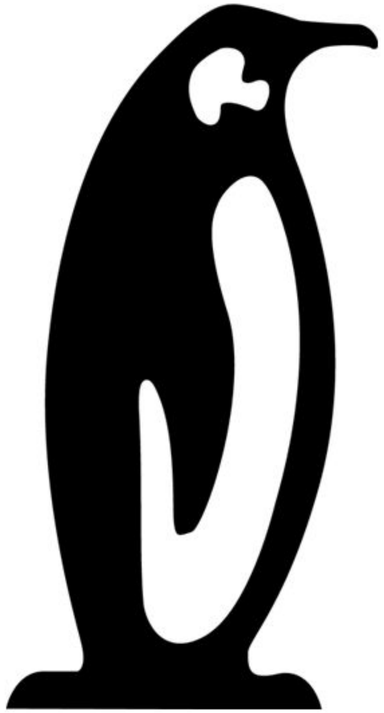


Linux Plumbers Conference

Vienna, Austria | September 18-20, 2024

Netdev CI

What is being validated. What Network subsystems can do.

LINUX PLUMBERS CONFERENCE | Vienna, Austria
Sept. 18-20, 2024

Matthieu Baerts (NGIO)
@matttbe@fosstodon.org

NIPA - Netdev Infrastructure for Patch Automation

Plan:

- What does it do? Why? What is important to know?
- How to extend its coverage?
- How to have this in other (sub)subsystems?
- What is missing?

What does it do?
Why?
What is important to know?

What does NIPA do?

It tests stuff...

What does NIPA do?

It tests stuff... a bit of history first.

- Before NIPA: patches were (hopefully) tested elsewhere:
  - Private tests: builds, static analytic, functional tests
  - Some CI validating different trees: LKP/O-day, Syzbot, LKFT, etc.
  - Late feedback

What does NIPA do?

It tests stuff... a bit of history first.

- In 2020: Jakub brought NIPA to life!
  - Static analytic tests:
    - Basic patch checks, build test with GCC, CLang
    - Executed patch-by-patch
  - Results are publicly available

What does NIPA do?

It tests stuff... a bit of history first.

- In December 2023: Functional tests are appearing
  - Coccicheck
  - Documentation building
  - KUnit
  - Report back to patchwork

What does NIPA do?

It tests stuff... a bit of history first.

- In January 2024: Functional tests are definitively there
  - KSelftests: VMs running networking selftests
  - Web pages to present results on netdev.bots.linux.dev
  - Results still published on Patchwork ; logs are still available

What is NIPA?

What it is:

- A bunch of scripts (Python/Bash)
- Tracking patches from netdev
- Machines to build / launch tests
- Web UI pages

What is NIPA?

What it is:

- A bunch of scripts (Python/Bash)
- Tracking patches from netdev
- Machines to build / launch tests
- Web UI pages

What it is **not**:

- A service to validate non-locally tested patches
- General testing purpose
- Hosting tests

Why is it needed?

- To help maintainers and reviewers:
  - Quick automated feedback
- To reduce the feedback loop, increase trust:
  - Reduce issues seen after merging: tracking, pinging, reverting, ...
- To have some controls:
  - What has been tested, how, integration with Patchwork, etc.

Functional tests: current status

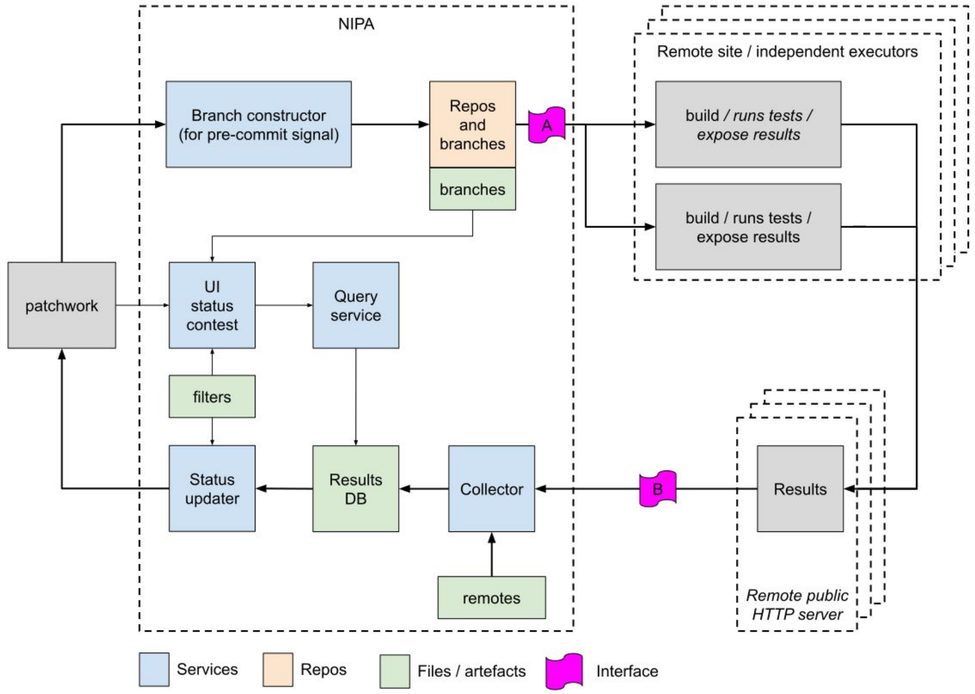
- 750+ tests not counting subtests
- 8 tests are ignored, mostly when using a debug kconfig
- ~2h to run get all results
- 26 VMs running tests in parallel
- **/!** Not coverring all cases **/!**

How does it work?

- Monitor patches from Patchwork (web service for maintainers)
- Add new patches to the build queue
  - Build + small tests
  - Send results on Patchwork
  - ETA: 1 to 12h (or more when CLang builds get stuck)
- Every 3h: periodic tests

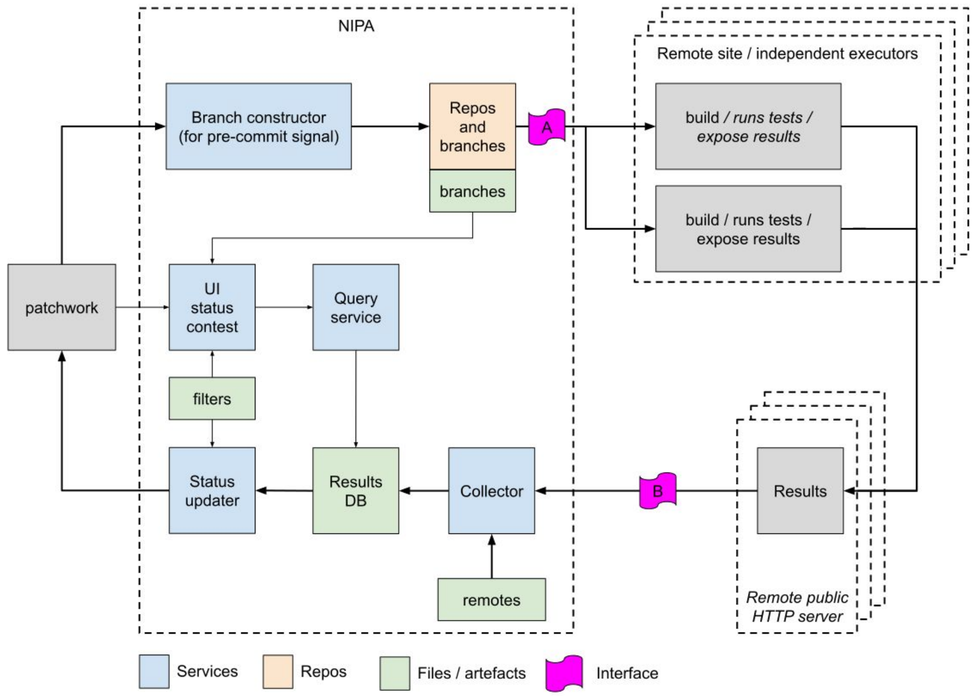
Periodic tests: every 3 hours: branch

- Create a new branch:
  - On top of net-next, merged with -net if possible
  - With all patches that passed the build and are still in review
- Why?
  - Tests can be long, and be retried
  - To cope with future HW tests



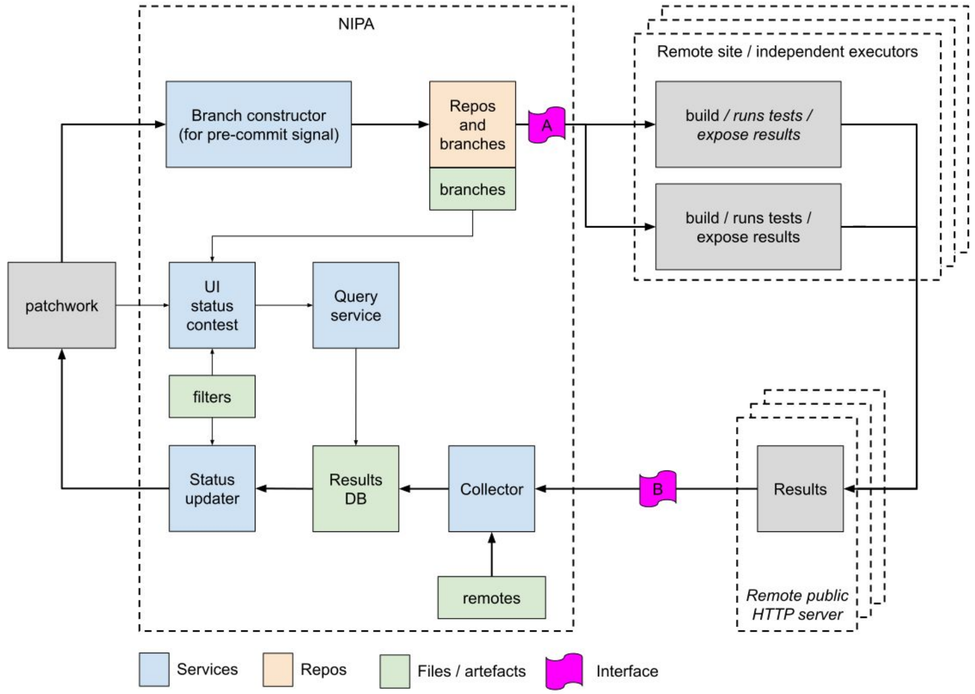
Periodic tests: every 3 hours: tests

- Functional tests in parallel:
  - KUnit, KSelftests, BPF selftests
  - With/Without debug kconfig
  - Multiple VMs in parallel



Periodic tests: every 3 hours: tests

- Report results:
  - For the maintainers & reviewers:
    - Not every failures are problematic
    - People might send too often, fixing unimportant issues.
  - Visible on Patchwork and WebUI

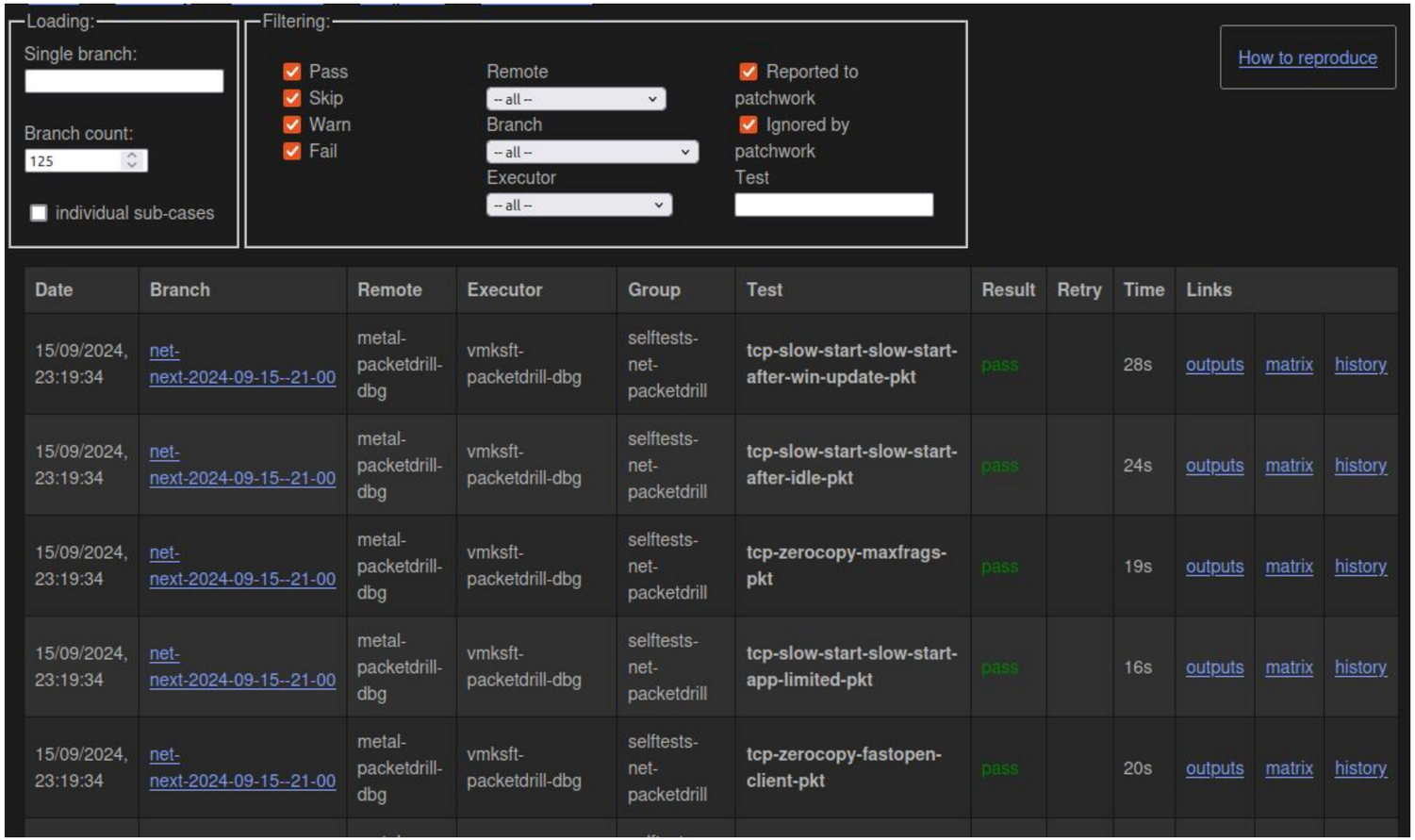


Web UI: Patchwork

netdev/cc_maintainers	success	CCed 13 of 13 maintainers
netdev/build_clang	success	Errors and warnings before: 16 this patch: 16
netdev/verify_signedoff	success	Signed-off-by tag matches author and committer
netdev/deprecated_api	success	None detected
netdev/check_selftest	success	No net selftest shell script
netdev/verify_fixes	success	No Fixes tag
netdev/build_allmodconfig_warn	success	Errors and warnings before: 17 this patch: 17
netdev/checkpatch	success	total: 0 errors, 0 warnings, 0 checks, 17 lines checked
netdev/build_clang_rust	success	No Rust files in patch. Skipping build
netdev/kdoc	success	Errors and warnings before: 0 this patch: 0
netdev/source_inline	success	Was 0 now: 0
netdev/contest	success	net-next-2024-09-16--18-00 (tests: 764)

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

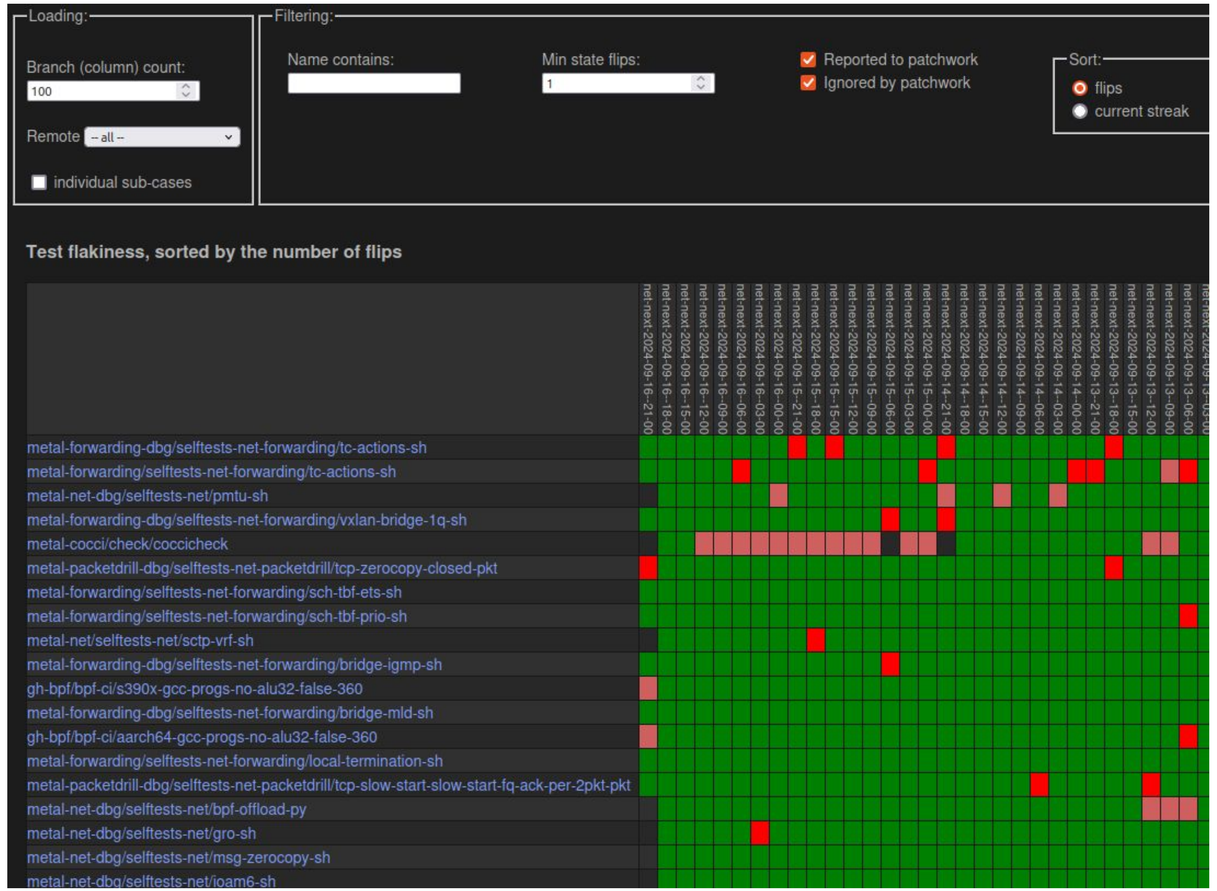
Web UI: Contest



Date	Branch	Remote	Executor	Group	Test	Result	Retry	Time	Links		
15/09/2024, 23:19:34	net-next-2024-09-15--21-00	metal-packetdrill-dbg	vmksft-packetdrill-dbg	selftests-net-packetdrill	tcp-slow-start-slow-start-after-win-update-pkt	pass		28s	outputs	matrix	history
15/09/2024, 23:19:34	net-next-2024-09-15--21-00	metal-packetdrill-dbg	vmksft-packetdrill-dbg	selftests-net-packetdrill	tcp-slow-start-slow-start-after-idle-pkt	pass		24s	outputs	matrix	history
15/09/2024, 23:19:34	net-next-2024-09-15--21-00	metal-packetdrill-dbg	vmksft-packetdrill-dbg	selftests-net-packetdrill	tcp-zerocopy-maxfrags-pkt	pass		19s	outputs	matrix	history
15/09/2024, 23:19:34	net-next-2024-09-15--21-00	metal-packetdrill-dbg	vmksft-packetdrill-dbg	selftests-net-packetdrill	tcp-slow-start-slow-start-app-limited-pkt	pass		16s	outputs	matrix	history
15/09/2024, 23:19:34	net-next-2024-09-15--21-00	metal-packetdrill-dbg	vmksft-packetdrill-dbg	selftests-net-packetdrill	tcp-zerocopy-fastopen-client-pkt	pass		20s	outputs	matrix	history

https://netdev.bots.linux.dev/contest.html

Web UI: Flakes



https://netdev.bots.linux.dev/flakes.html

Web UI: Status

Build processing

Tree	Qlen	Tid	Test	Pid	Patch
bpf					
bpf-next					
net					
net-next					
net-next					

Service	Status	Tasks	CPU cores	Memory Use
nipa-poller.service	active / running	7	20.92	108.41GB
nipa-upload.service	active / running	1	0.02	0.03GB
nipa-mailbot.service	active / running	1	0.00	0.13GB
nipa-brancher.service	active / running	2	0.00	2.67GB
nipa-contest.service	active / running	1	0.09	0.57GB
nipa-collector.service	active / running	1	0.06	3.39GB
net-next.service	success		0.18	
nipa-checks.service	success		0.13	
nipa-flask.service	active / running	9	0.00	0.55GB

Continuous testing results

Hide all-pass runs

Branch	Remote	Time	Tests	Result
net-next-2024-09-15--21-00		15/09/2024, 23:00:19		cidiff
	tdc-dbg	*no result*		
	metal-net-dbg	*pending (expected in 1h 33m)*		
	metal-cocci	*pending (expected in 1h 5m)*		
	metal-mptcp-dbg	*pending (expected in 54m 51s)*		
	metal-net	*pending (expected in 44m 13s)*		
	metal-nf-dbg	*pending (expected in 32m 32s)*		
	metal-forwarding-dbg	*pending (expected in 30m 23s)*		
	metal-netdevsim-dbg	*pending (expected in 10m 19s)*		
	metal-doc-build	*pending (expected in 8m 55s)*		
	metal-bonding-dbg	*pending (expected in 7m 23s)*		
	metal-forwarding	*pending (expected in 4m 24s)*		
	metal-mptcp	*pending (expected in 3m 32s)*		
summary	*14 remotes (all hidden)*	*18m 55s*	*262 / 0 / 0*	**pending**
net-next-2024-09-15--18-00		15/09/2024, 20:00:20		cidiff
	metal-cocci	1h 30m	0 / 0 / 1	fail
	metal-net	1h 5m	109 / 0 / 1	fail
summary	*26 remotes (24 hidden)*	*1h 54m*	*755 / 0 / 2*	**fail**
net-next-2024-09-15--15-00		15/09/2024, 17:00:15		cidiff
	metal-cocci	1h 27m	0 / 0 / 1	fail
summary	*26 remotes (25 hidden)*	*1h 53m*	*755 / 0 / 1*	**fail**
net-next-2024-09-15--12-00		15/09/2024, 14:00:10		cidiff
	metal-cocci	1h 26m	0 / 0 / 1	fail

https://netdev.bots.linux.dev/status.html

Web UI

- Oriented for maintainers and reviewers
- Developers can check: which tests need to be improved?
- Series' author can prepare an eventual future version:
  - But again: test locally first!

Reproduce issues

- Static analytic issues: should be clear and easy to reproduce
- Functional tests: check NIPA's wiki
  - **`virtme-ng`** can help to build and launch a VM
  - Tests can then be launched manually from the VM
  - Some tools might be needed: IPRoute2, NFTables, etc.
  - Maybe there will be a container with all required tools?

https://github.com/linux-netdev/nipa/wiki/

How to extend its coverage?

Extensions

- Increase code coverage with new tests:
  - KUnit: lightweight unit testing framework
  - KSelftests: small tests to exercise individual kernel code paths
  - A new remote: for special infrastructures:
    - Hardware
    - Complex or external dependences

Extensions: KSelftests

- Simple programs in userspace
- Return code: PASS, SKIP, FAIL, XPASS, XFAIL
- Helpers in C, Bash and Python
- TAP format can be used to report subtests
- Should run on any kernel versions...

Extensions: KSelftests: run on any kernel!

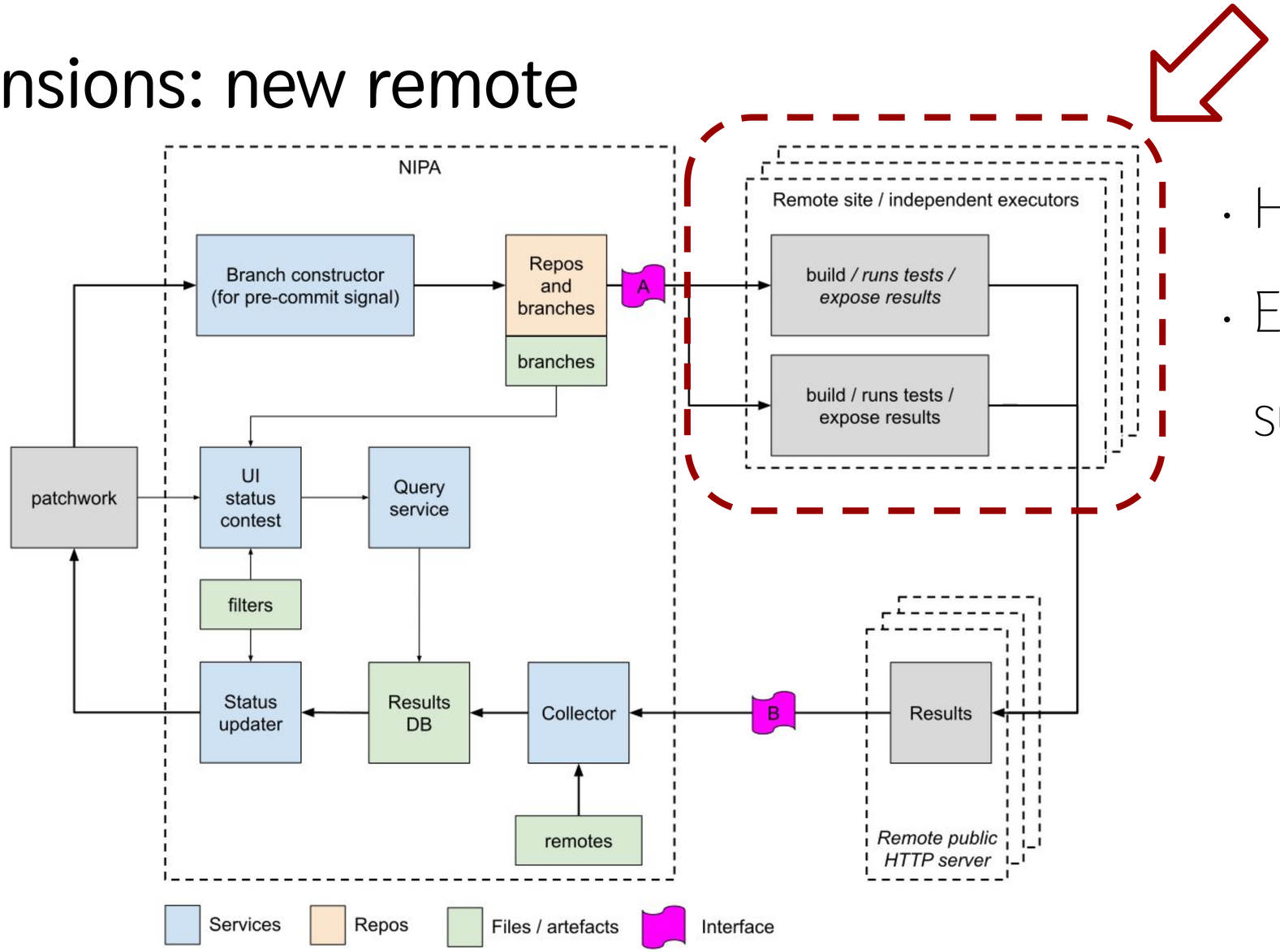
- *"Running tests from mainline offers the best coverage."*
- *"To regression test a bug, we should be able to run that test on an older kernel."*
- Some CIs are doing that when testing stable kernels:
  - Selftests with many subtests can be marked as failed if one cannot work on stable kernels

https://docs.kernel.org/dev-tools/kselftest.html

Extensions: KSelftests: run on any kernel?

- Feasible with Networking tests?
- New syscall, check error: OK
- New feature, still using socket API: NOK
  - MPTCP in v5.6: only one path
  - How to check if the kernel supports multiple paths, >5.6
  - Counters? KAllSyms? Kernel version? (RHEL case)

Extensions: new remote



- HW specific
- External tests suite

KSelftests Drivers

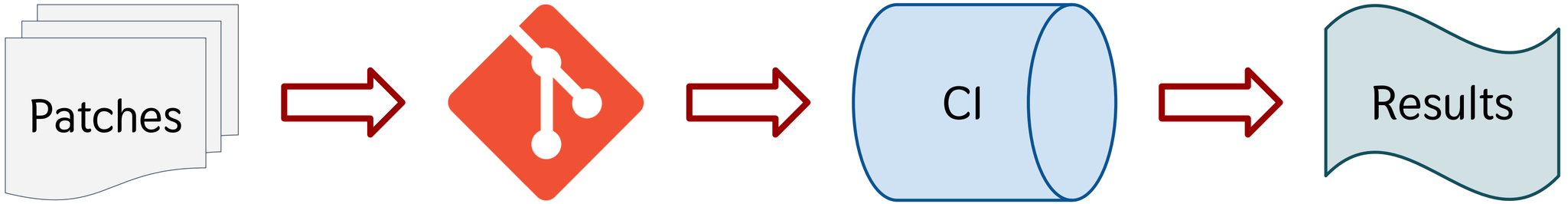
- New requirement from v6.12 for “supported” drivers
- Using a remote runner attached to HW:
  - Helpers for single or dual hosts (ssh) / interfaces (netns)
- Why:
  - Improve feature delivery
  - user and vendor participation

KSelftests Drivers: Why?

- Requirements / specs defined as tests, reflecting use-cases
- Increase compatibility, share effort, avoid regressions
- **`netdevsim`** can help for prototyping
- Tests can be sent / enabled with implementation
- Make *upstream-first* development model more feasible

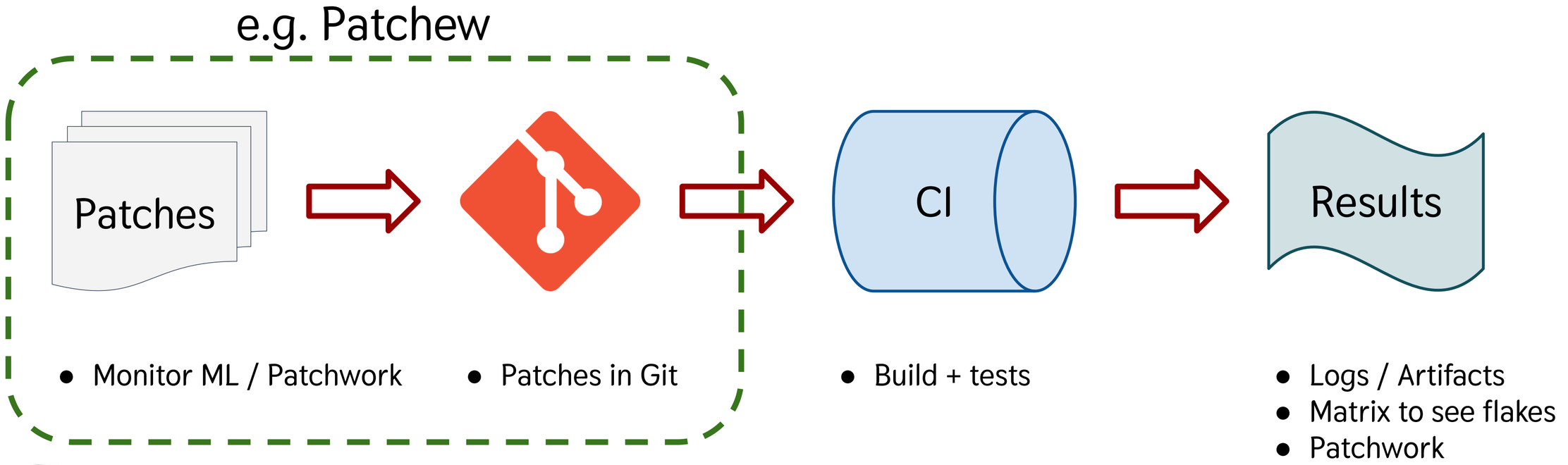
How to have this in other (sub)subsystems?

Replicate this in other subsystems

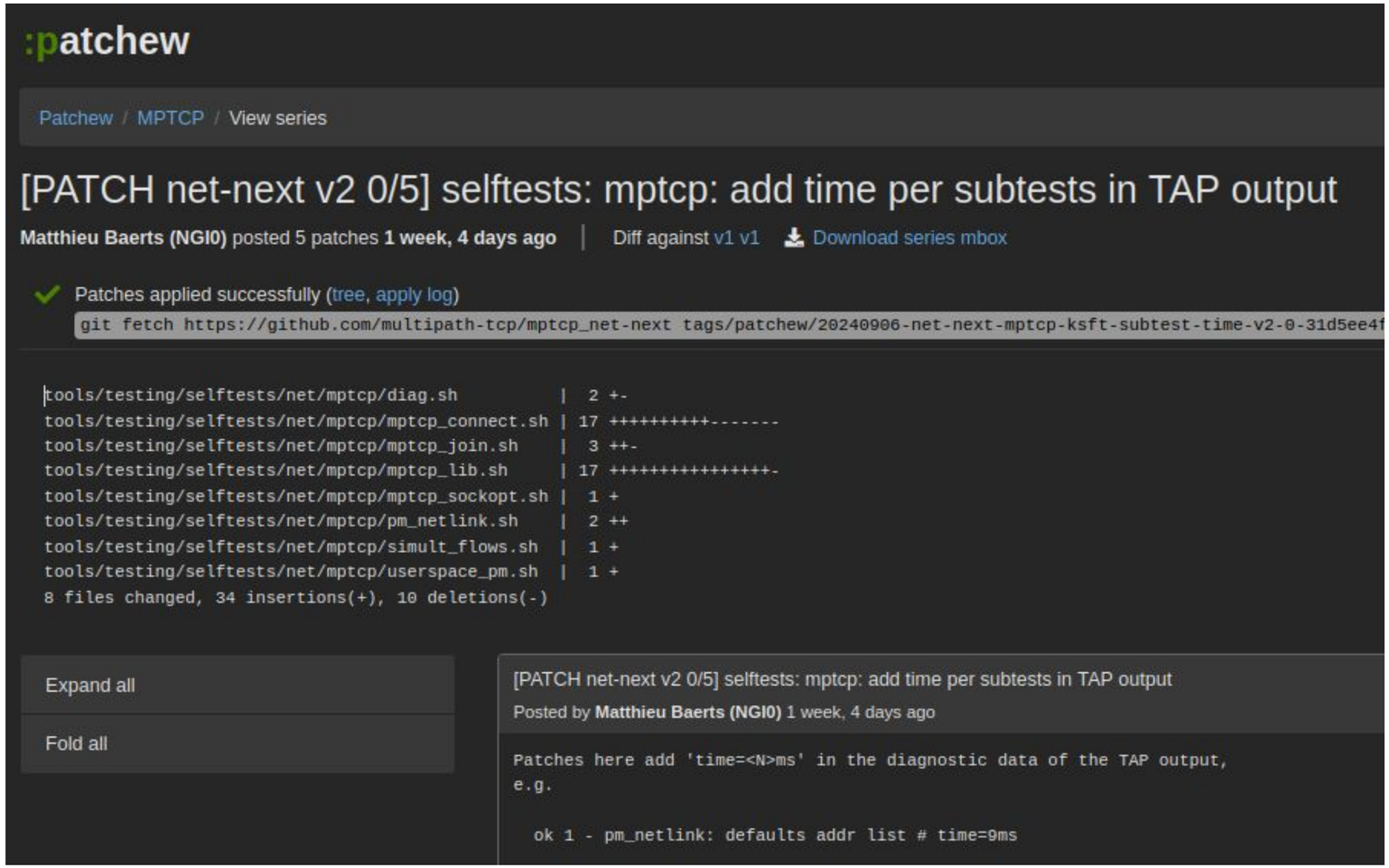


- Monitor ML/Patchwork
- Patches in Git
- Build + tests
- Logs / Artifacts
- Matrix to see flakes
- Patchwork

Replicate this in other subsystems

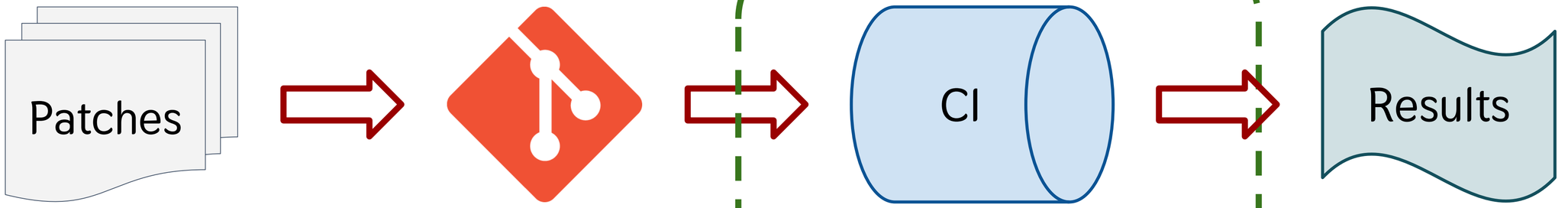


Patches ⇒ Git: Patchew can help



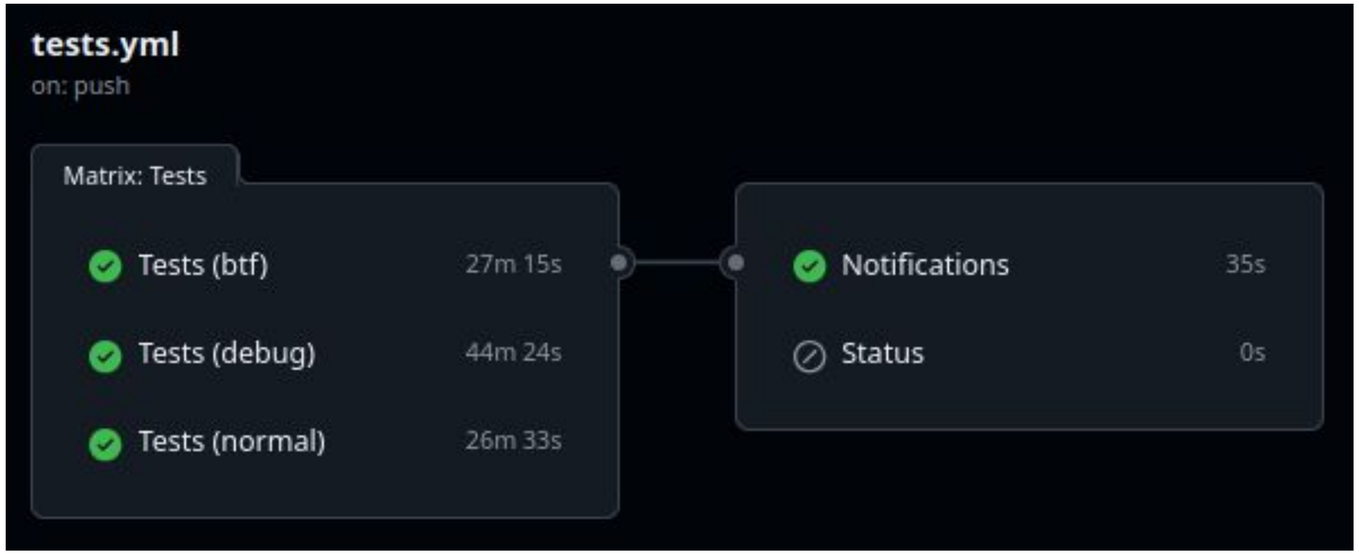
https://patchew.org

Replicate this in other subsystems



- Monitor ML / Patchwork
- Patches in Git
- Build + tests
- Logs / Artifacts
- Matrix to see flakes
- Patchwork

CI: GitHub Actions can help



CI: requirements

- Service or dedicated servers, e.g. HW dependences
- Run the tests:
  - Setup environment
  - Build kernel + run in a VM or dedicated HW
  - KVM support
  - Build / git cache
  - Catch errors: call trace, warning messages, kmemleak, etc.

CI: example

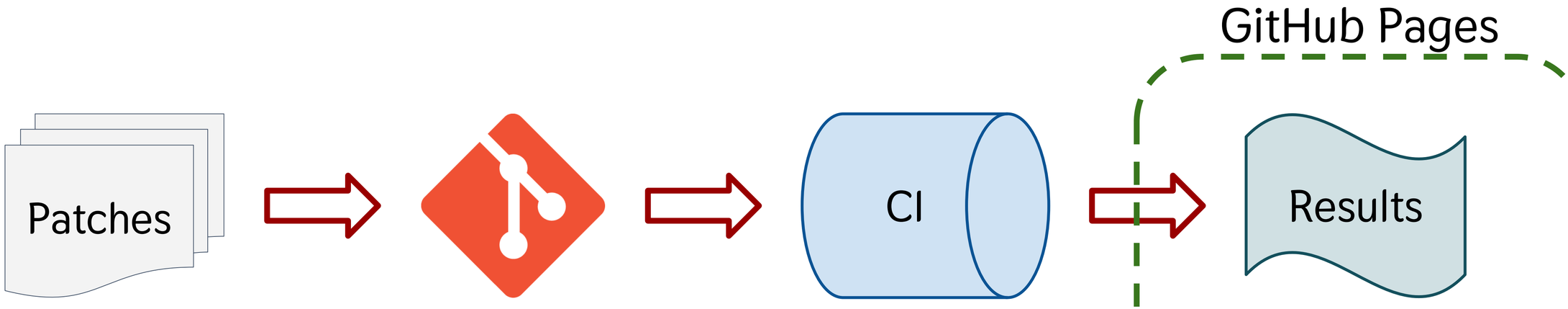
- Environment: containers can help

```
docker run (...) --privileged mptcp/mptcp-upstream-virtme-docker:latest
```

- VM: `virtme-ng` can help
- KVM support: Github Actions **can** support it → opt-in
- Cache: **`ccache`** can help
- Catching errors: shared resources?

https://github.com/multipath-tcp/mptcp-upstream-virtme-docker

Replicate this in other subsystems

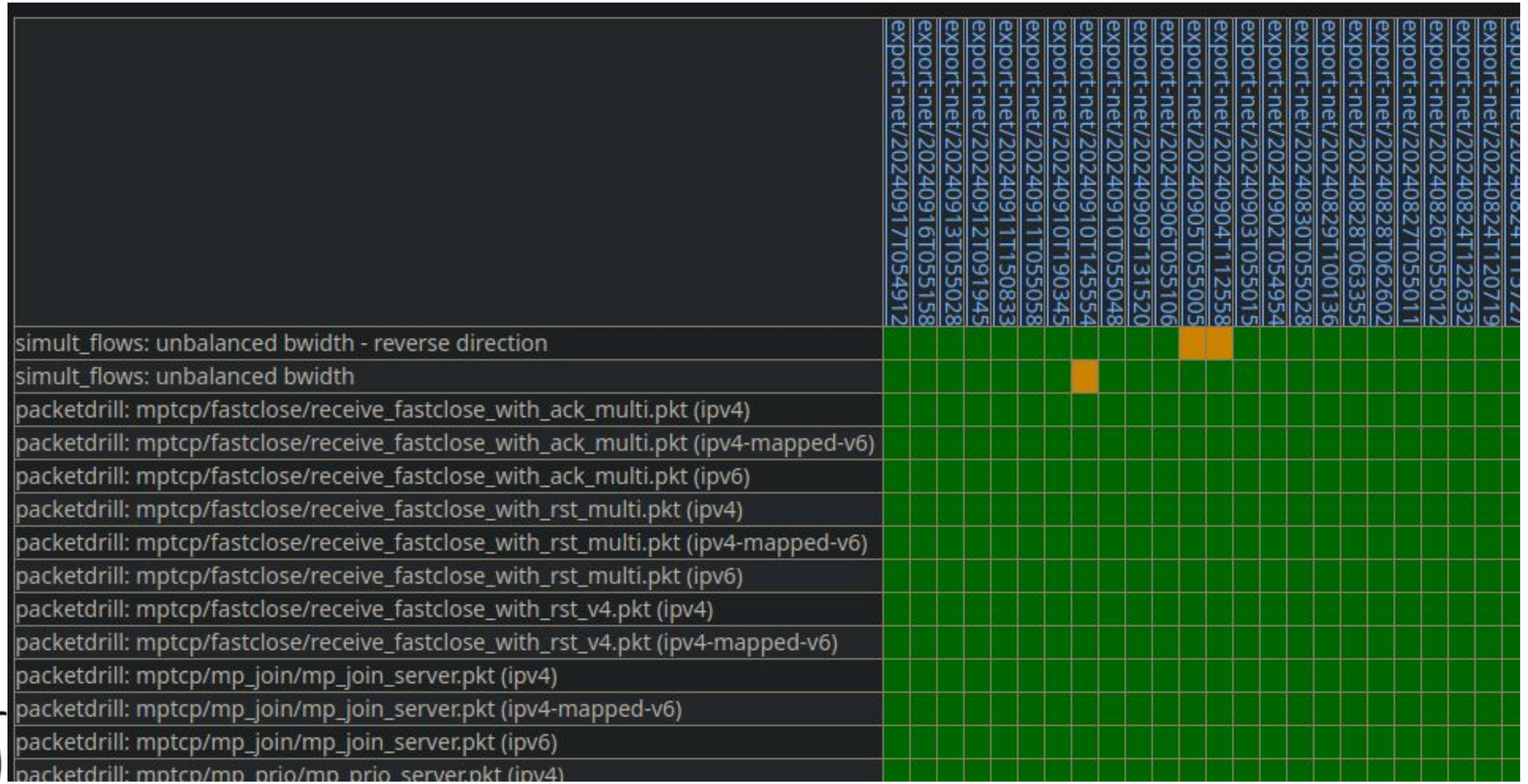


- Monitor ML / Patchwork
- Patches in Git
- Build + tests
- Logs / Artifacts
- Matrix to see flakes
- Patchwork

Results: example

- Logs / Artifacts: usually easy
- Show the last results: TAP parsers or converters to JUnit, etc.
- Check regressions: home made solution published in HTML

Results: example



Interested by that for your subsystem?

- Contact me
- https://github.com/multipath-tcp/mptcp_net-next/actions
- https://github.com/multipath-tcp/mptcp-upstream-virtme-docker

What is missing in NIPA?

Any suggestions?

Linux Plumbers Conference

Vienna, Austria | September 18-20, 2024

Matthieu Baerts (NGI0)
@matttbe@fosstodon.org

The Netdev CI has been checking patches sent to the Netdev mailing list for a couple of years now. Thanks to that, Netdev maintainers are able to easily check which patches are causing issues despite the high volume of patches that are shared every day. Until this year, the CI was limited to kernel builds, and various static checks, but the good thing is that all results were already available publicly. Kernel developers can then access the logs to understand what went wrong, without too much assistance from the maintainers.

In 2024, the Netdev CI has seen the introduction of functional tests by running many Network kernel selftests and unit tests. Even if some of these tests were certainly executed regularly by some, they are now automatically tested, and their results are available to all. This really helps Netdev maintainers and contributors to catch regressions early, and encourage everybody to have their new features and fixes covered by new test cases.

This talk will present how the Netdev CI is currently working, and the small details that are important to know. But it will also explain how it can be extended, e.g. to run some tests on real hardware to validate some drivers, to execute other specific tests that are not part of the kernel repo, tracking performance regressions in a dedicated environment, etc.

Another topic that will be mentioned is how Network subsystems, can have a similar service on their side. The MPTCP CI will be taken as an example, using GitHub Actions with KVM support to run various tests on development patches without having to maintain custom servers similar to what is in place with the Netdev CI.

Comments: My main goal here is to explain what is being done on the Netdev CI, how Netdev subsystems maintainers and contributors can extend it to cover more cases, and have a similar service on their side to pre-validate patches before upstreaming them to Netdev.

On a related topic, this talk can also initiate some discussions about kselftests that are supposed to support any previous kernel versions, and not only the kernel code they are attached to in the kernel repo. This seems hard to support for the Networking subsystem, and even harder to maintain. But if this is not supported, then CIs validating stable versions will stop reporting useful results.