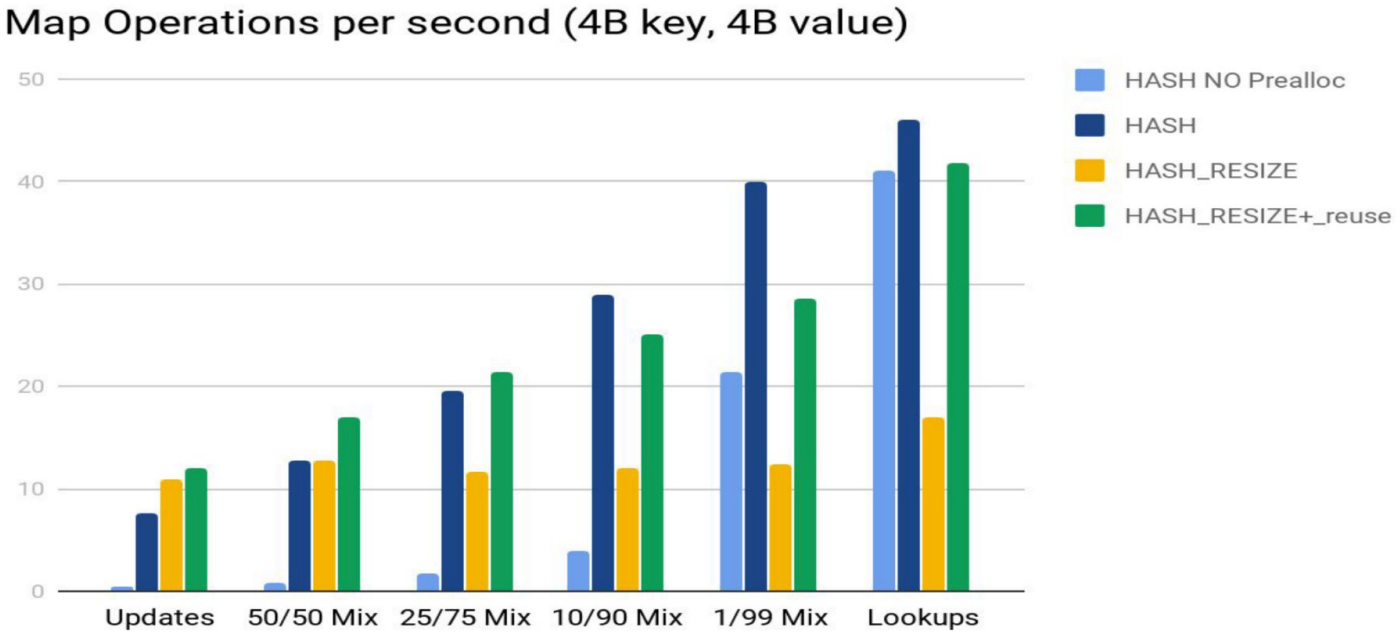
Better algorithms for BPF

Resizable HASHMAP

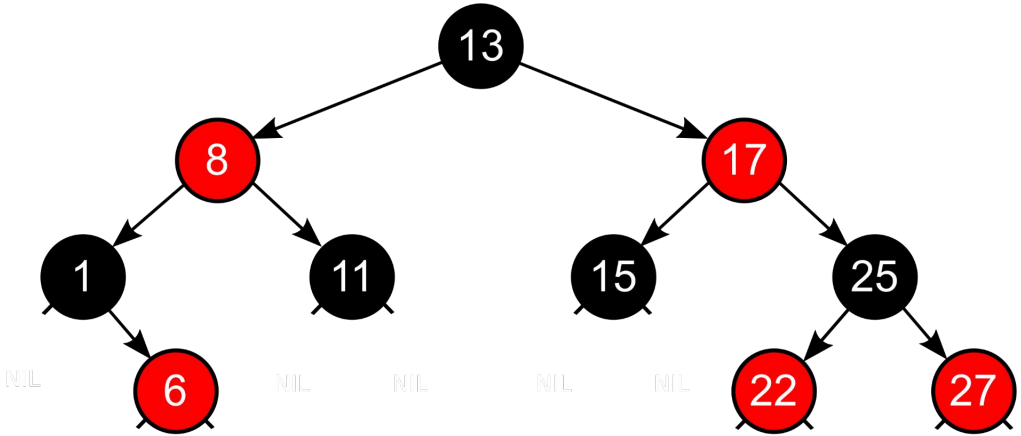
- Rhashtable-based map
  - John Fastabend’s “Right-sizing is hard, Resizable maps for optimal map size”
  - https://lpc.events/event/7/contributions/681/

HASH_RESIZE Benchmark



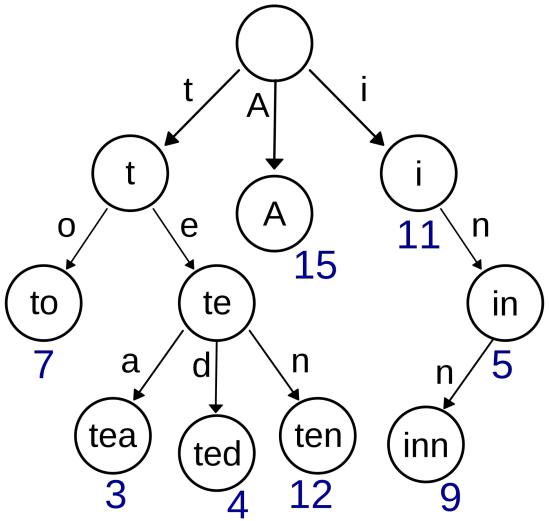
Alternatives to HASHMAP

- Rhashtable-based map
  - John Fastabend's "Right-sizing is hard, Resizable maps for optimal map size"
  - https://lpc.events/event/7/contributions/681/
- Red-Black tree



Alternatives to HASHMAP

- Rhashtable-based map
  - John Fastabend's "Right-sizing is hard, Resizable maps for optimal map size"
  - https://lpc.events/event/7/contributions/681/
- Red-Black tree
- Trie-based implementation

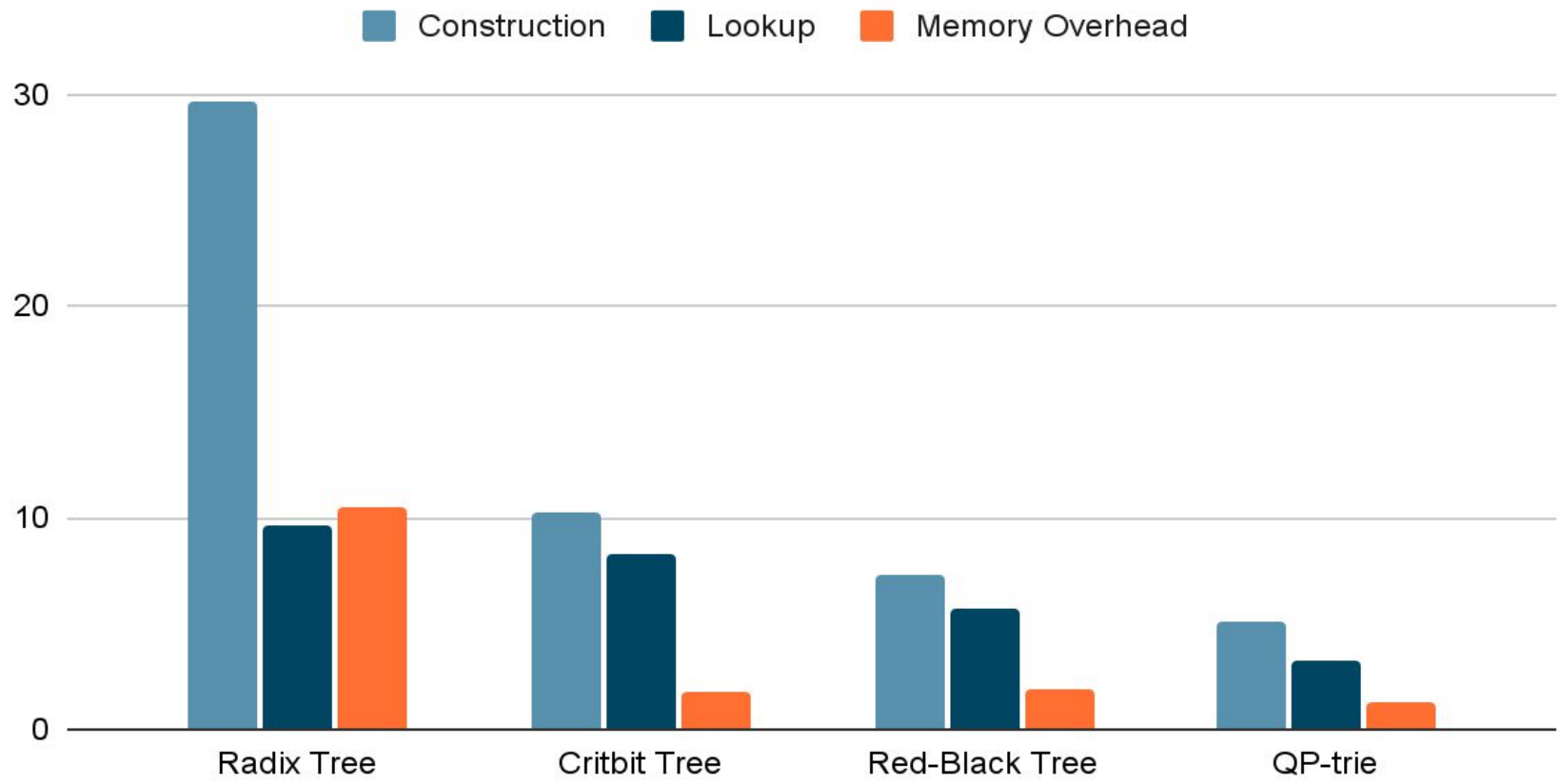


QP-trie

- https://dotat.at/prog/qp/README.html
- https://github.com/fanf2/qp
- Some *encouraging* upstream results from Hou Tao:
  - https://lore.kernel.org/bpf/8b4c1ad2-d6ba-a100-5438-a025ceb7f5e1@huawei.com/

  “... after adding -march=native, both the lookup and update performance of qp-trie are improved. And the lookup performance of qp-trie is always better than tst, but the update performance of qp-trie is still worse than tst.”

Sorted lookup table benchmark



https://9vx.org/post/qp-tries/

NMI curse

- NMI requires everything preallocated
- Which is hard (impossible?) for RB tree, QP trie and such
- Possible solution: allow offloading work out of NMI
  - Kernel-to-kernel BPF ringbuf
  - irq_work_queue() for BPF programs

How do we decide?

Benchmarks! Benchmarks! Benchmarks!

Improved hashing algorithms

- Currently we use jhash (aka lookup3.c)
- 15 years of improvements for hashing algorithms since then
- BPF_MAP_STACK_TRACE suffers from collision
- HASH, BLOOM_FILTER, etc will benefit from **better** and **faster** algo

Improved hashing algorithms

- Currently we use jhash (aka lookup3.c)
- 15 years of improvements for hashing algorithms since then
- BPF_MAP_STACK_TRACE suffers from collision
- HASH, BLOOM_FILTER, etc will benefit from **better** and **faster** algo

Is xxHash by Yann Collet the way to go?..

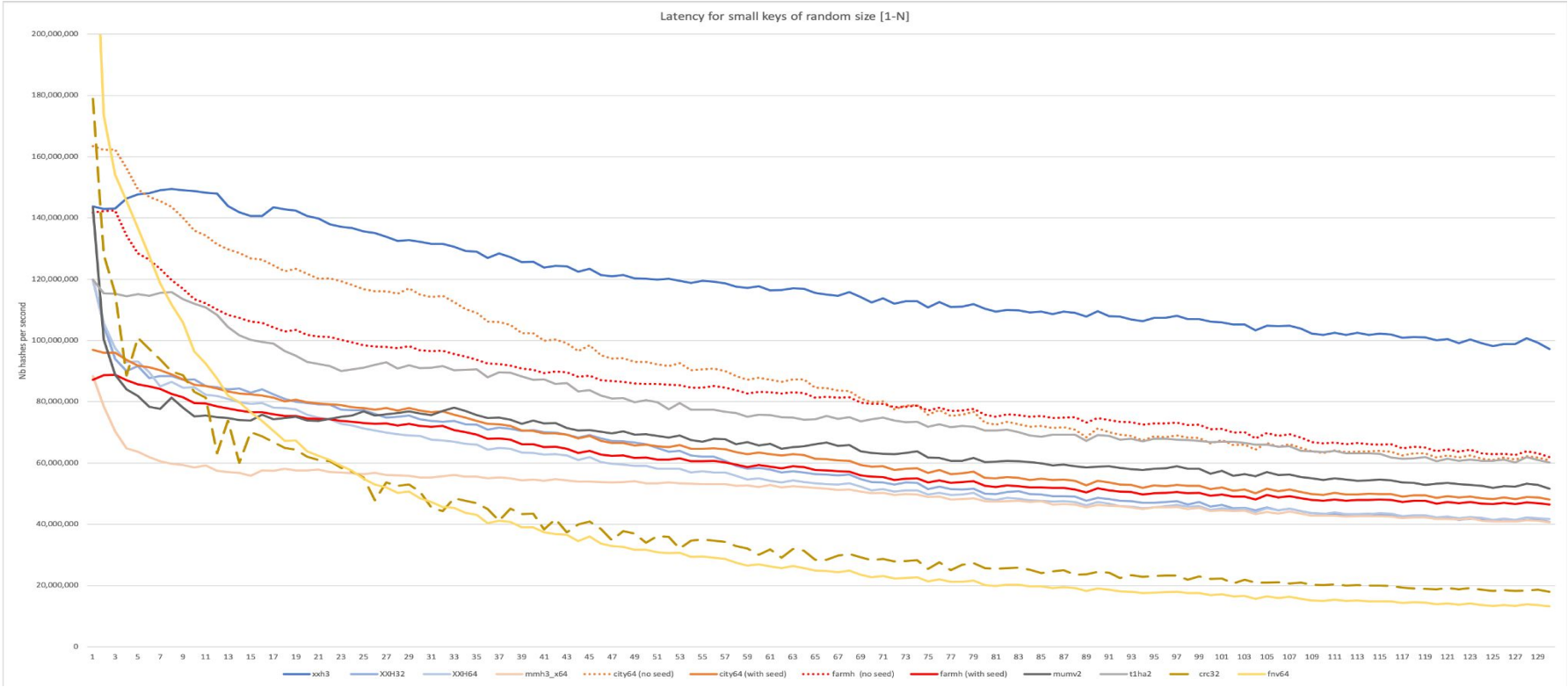
xxHash vs others

Hash Name	Width	Bandwidth (GB/s)	Small Data Velocity	Quality	Comment
XXH3 (SSE2)	64	31.5 GB/s	133.1	10	
XXH128 (SSE2)	128	29.6 GB/s	118.1	10	
RAM sequential read	N/A	28.0 GB/s	N/A	N/A	*for reference*
City64	64	22.0 GB/s	76.6	10	
T1ha2	64	22.0 GB/s	99.0	9	Slightly worse collisions
City128	128	21.7 GB/s	57.7	10	
XXH64	64	19.4 GB/s	71.0	10	
SpookyHash	64	19.3 GB/s	53.2	10	
Mum	64	18.0 GB/s	67.0	9	Slightly worse collisions
XXH32	32	9.7 GB/s	71.9	10	
City32	32	9.1 GB/s	66.0	10	
Murmur3	32	3.9 GB/s	56.1	10	
SipHash	64	3.0 GB/s	43.2	10	
FNV64	64	1.2 GB/s	62.7	5	Poor avalanche properties
Blake2	256	1.1 GB/s	5.1	10	Cryptographic
SHA1	160	0.8 GB/s	5.6	10	Cryptographic but broken
MD5	128	0.6 GB/s	7.8	10	Cryptographic but broken

https://github.com/Cyan4973/xxHash

xxHash: long vs small inputs dichotomy

XXH3 has been designed for excellent performance **on both long and small inputs**



xxHash

- We have xxh32 and xxh64 contributed back in 2017 for zstd
- Let’s jump to 2022 and have xxh3 in Linux!

Maintainers **would be happy** to review contributions (w/ benchmarks)!

BPF ringbuf evolution

- User-to-kernel ringbuf
- Kernel-to-kernel ringbuf
- BPF program called for each record (PROG_TYPE_SYSCALL equivalent)
  - BPF_RINGBUF_SUBMIT command runs BPF prog on each sample?
  - Kthread to run each BPF program?
- bpf_dynptr is an interface to a memory
- Need to work through guarding against malicious user-space