Extension Framework for File Systems in User Space

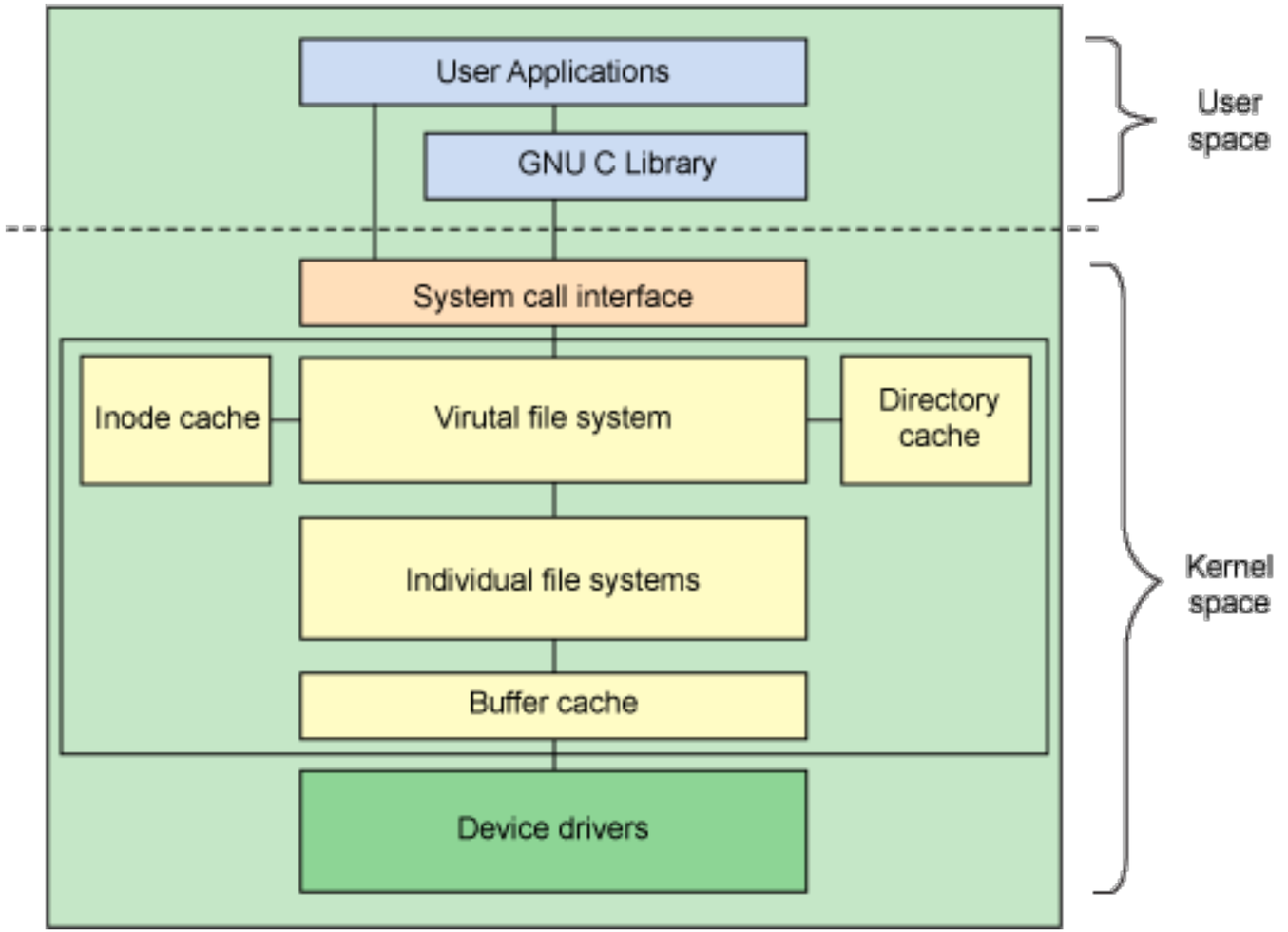
2019 USENIX Annual Technical Conference (ATC)

Ashish Bijlani and Umakishore Ramachandran
Georgia Institute of Technology

Presenter: Yu-Chia Liu
Date: 1/21/2020

File System in Kernel Level

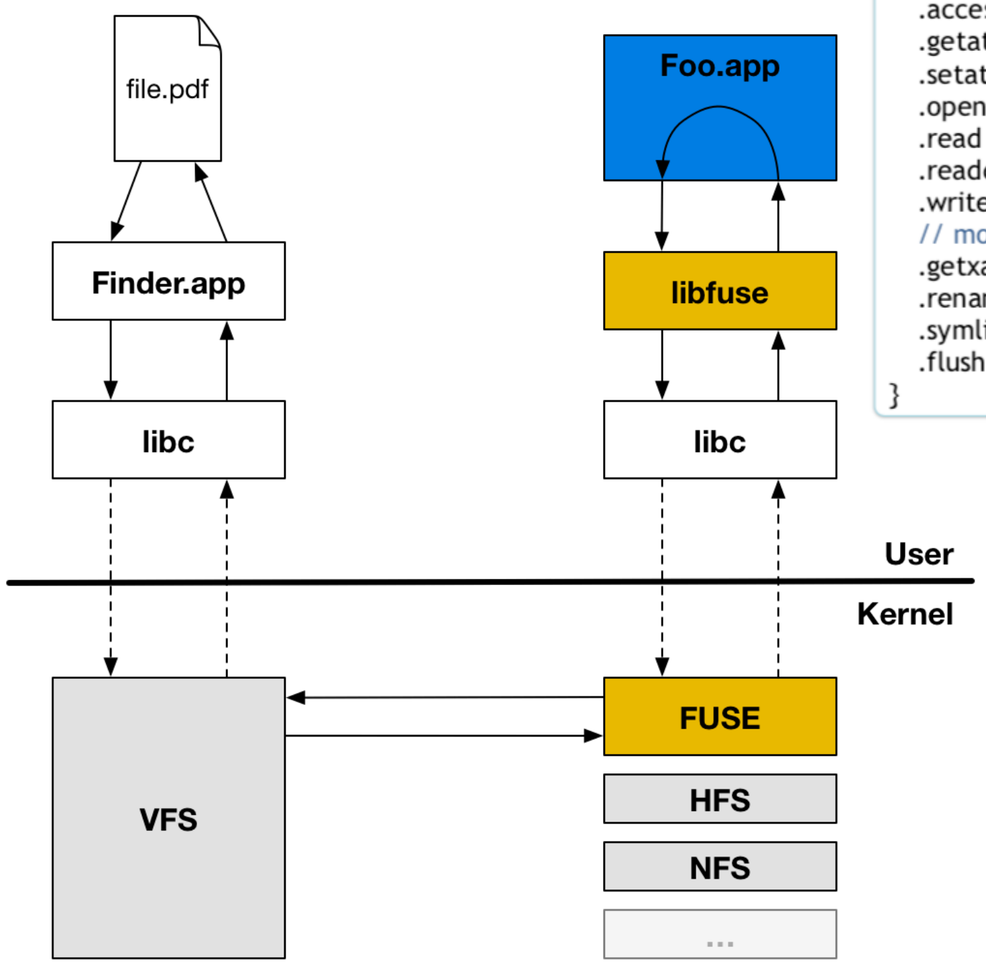
- Pros
  - Native performance
  - General purpose
- Cons
  - Cause kernel panic when system crashed
  - Less flexibility
  - More difficult to develop/debug/ maintain



http://www.cs.montana.edu/~andrew.hamilton/cs560/VFS/Arch.html

File Systems in User Space (FUSE)

- Pros
  - Flexibility
  - Easy to Develop/Debug/Maintain
  - Specialized
- Cons
  - Poor Performance



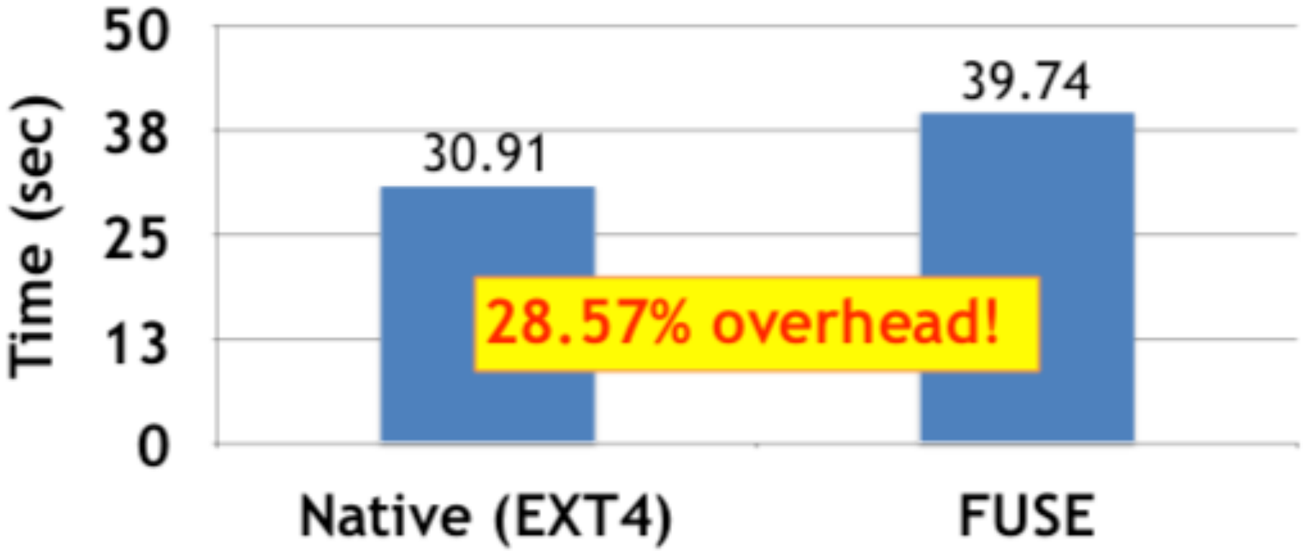

```
struct fuse_lowlevel_ops ops {
    .lookup  = handle_lookup,
    .access  = NULL,
    .getattr = handle_getattr,
    .setattr = handle_setattr,
    .open    = handle_open,
    .read    = handle_read,
    .readdir = handle_readdir,
    .write   = handle_write,
    // more handlers ...
    .getxattr = handle_getxattr,
    .rename = handle_rename,
    .symlink = handle_symlink,
    .flush   = NULL,
}
```

Kernel Switches

https://blogs.dropbox.com/tech/2016/05/going-deeper-with-project-infinite/

FUSE Performance

- "cd linux-4.18; make tinyconfig; make -j4"
  - Intel i5-3350 quad core, Samsung 850 EVO NVMe SSD, 16 GB RAM
  - Linux 4.11.0, Ubuntu 16.04.4 LTS, LibFUSE commit # 386b1b, StackFS w/ EXT4

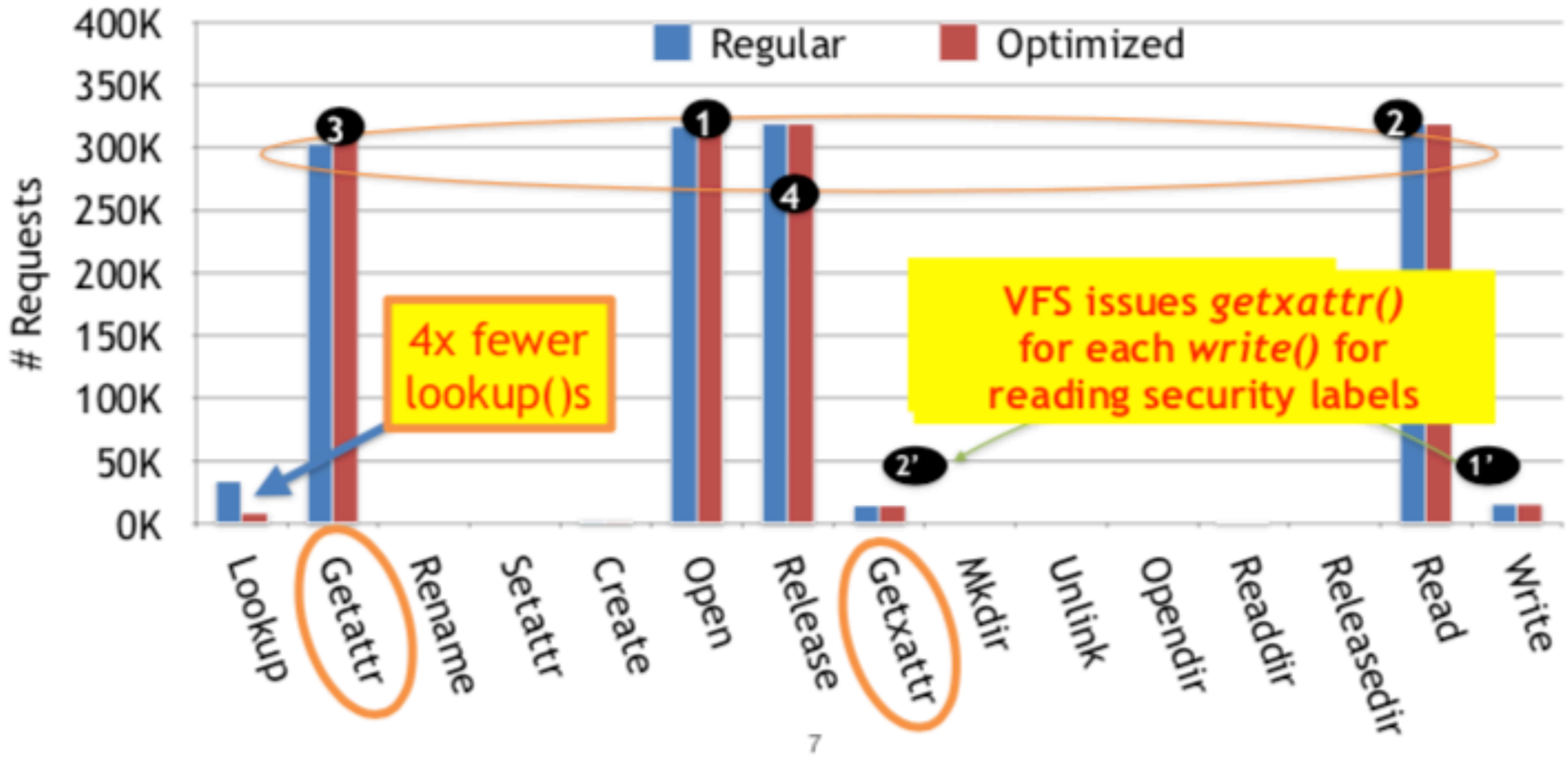


Opts Enabled

```
-o max_write=128K
-o splice_read
-o splice_write
-o splice_move
entry_timeout > 0
attr_timeout > 0
```

Performance Breakdown

- # of Requests Received by FUSE



7

ExtFUSE

- Extension framework for FUSE
  - A thin extension that can handle requests in kernel -> Avoid context switches
- Share data between FUSE daemon and extensions using eBPF maps
  - Cache metadata in the kernel

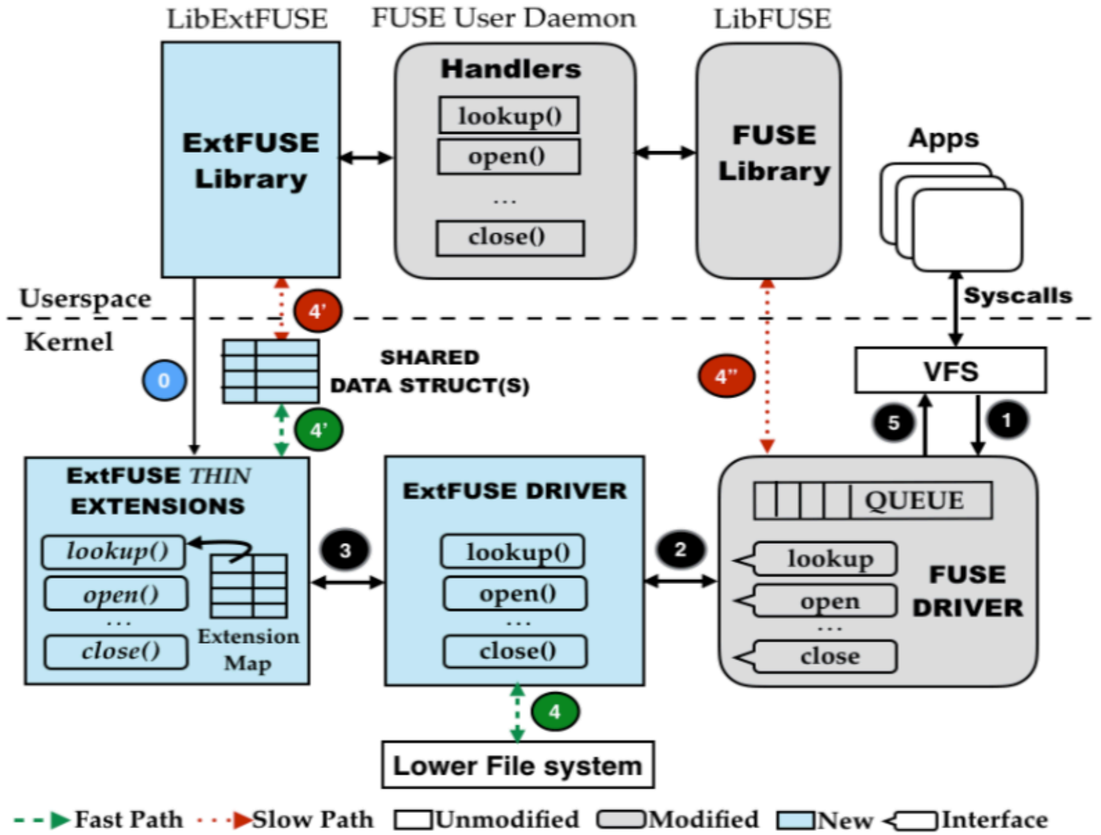
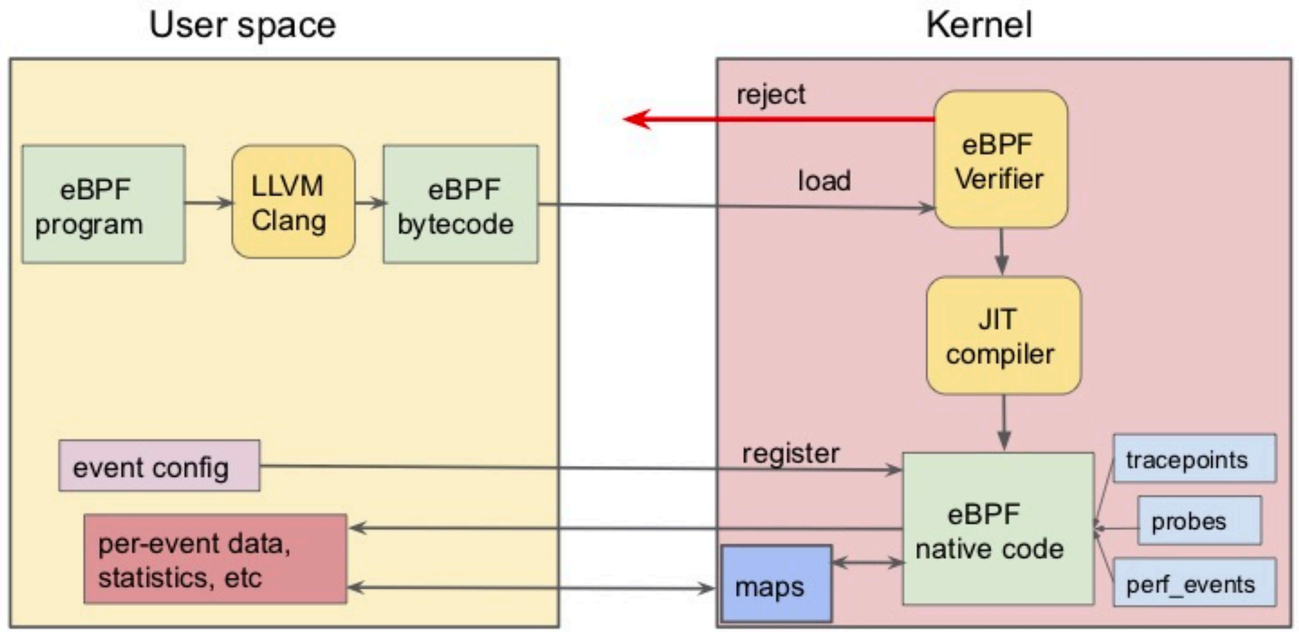


Figure 1: Architectural view of the ExtFUSE framework. The components modified or introduced have been highlighted.

eBPF?

- Extended Berkeley Packet Filtering
- Allows to running user space programs inside the kernel
- Avoid unnecessary copies to the user space
- Use the key-value maps as meta-data caches



http://seahorn.github.io/seahorn/crab/static%20analysis/linux%20extensions/ebpf/2019/07/04/seahorn-ebpf.html

Advantages of Using eBPF

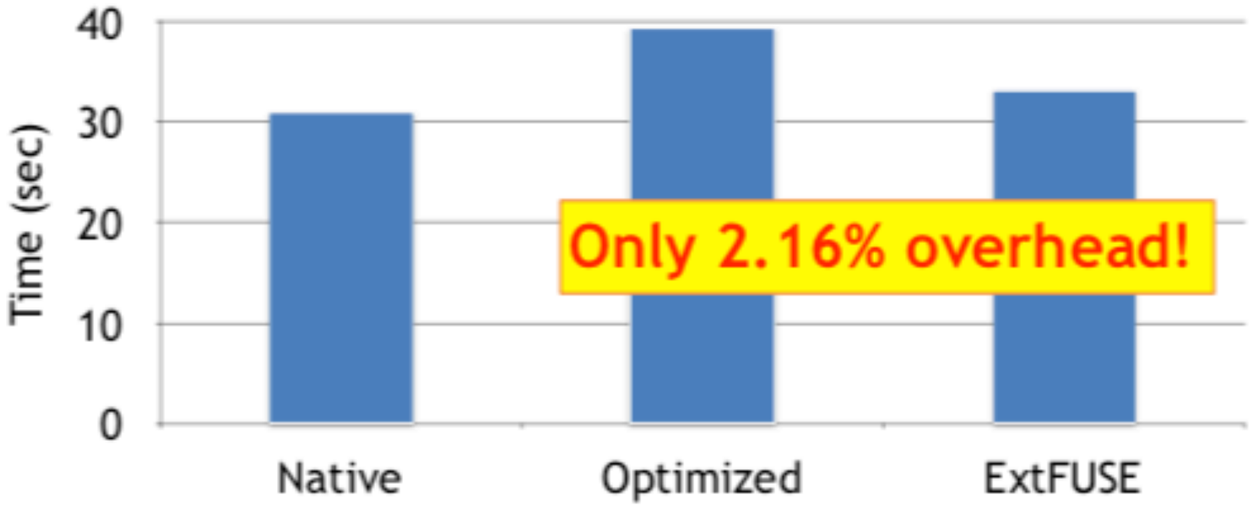
- BPF code can proactively cache/invalidate meta-data in kernel
- BPF code can perform customized filtering or permission checks
- BPF code can directly forward I/O requests to lower FS in kernel

Metadata	Map Key	Map Value	Caching Operations	Serving Extensions	Invalidation Operations
Inode	<nodeID, name>	fuse_entry_param	`lookup, create, mkdir, mknod`	*lookup*	`unlink, rmdir, rename`
Attrs	<nodeID>	fuse_attr_out	`getattr, lookup`	*getattr*	`setattr, unlink, rmdir`
Symlink	<nodeID>	link path	`symlink, readlink`	*readlink*	`unlink`
Dentry	<nodeID>	fuse_dirent	`opendir, readdir`	*readdir*	`releasedir, unlink, rmdir, rename`
XAttrs	<nodeID, label>	xattr value	`open, getxattr, listxattr`	*getattr, listxattr*	`close, setxattr, removexattr`

Table 4: Metadata can be cached in the kernel using eBPF maps by the user-space daemon and served by kernel extensions.

ExtFUSE Performance

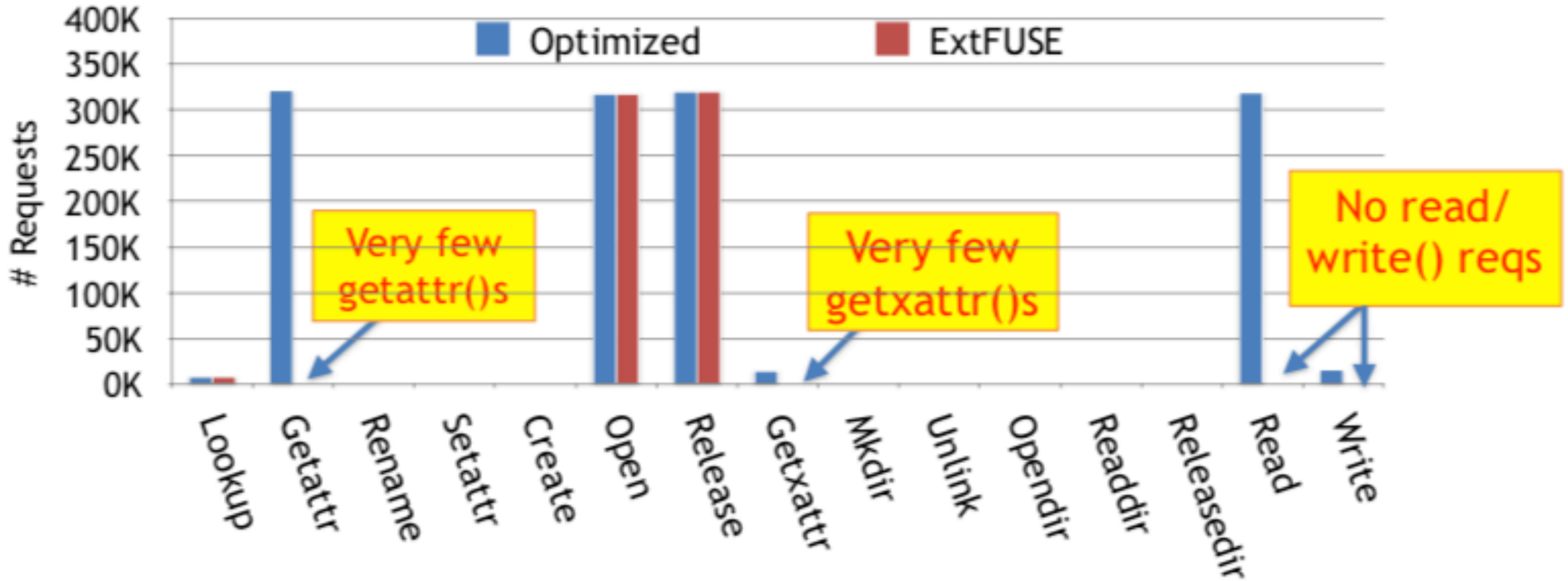
- "cd linux-4.18; make tinyconfig; make -j4"
  - Intel i5-3350 quad core, Samsung 850 EVO NVMe SSD, 16 GB RAM
  - Linux 4.11.0, Ubuntu 16.04.4 LTS, LibFUSE commit # 386b1b, StackFS w/ EXT4



Overhead
Optimized Latency: 28.57%
ExtFUSE Latency: 2.16%
ExtFUSE Memory: 50MB
(worst case)
Cached: lookup, attr, xattr
Passthrough: read, write

Performance Breakdown

- # of Requests Received by FUSE



Filebench microbenchmarks

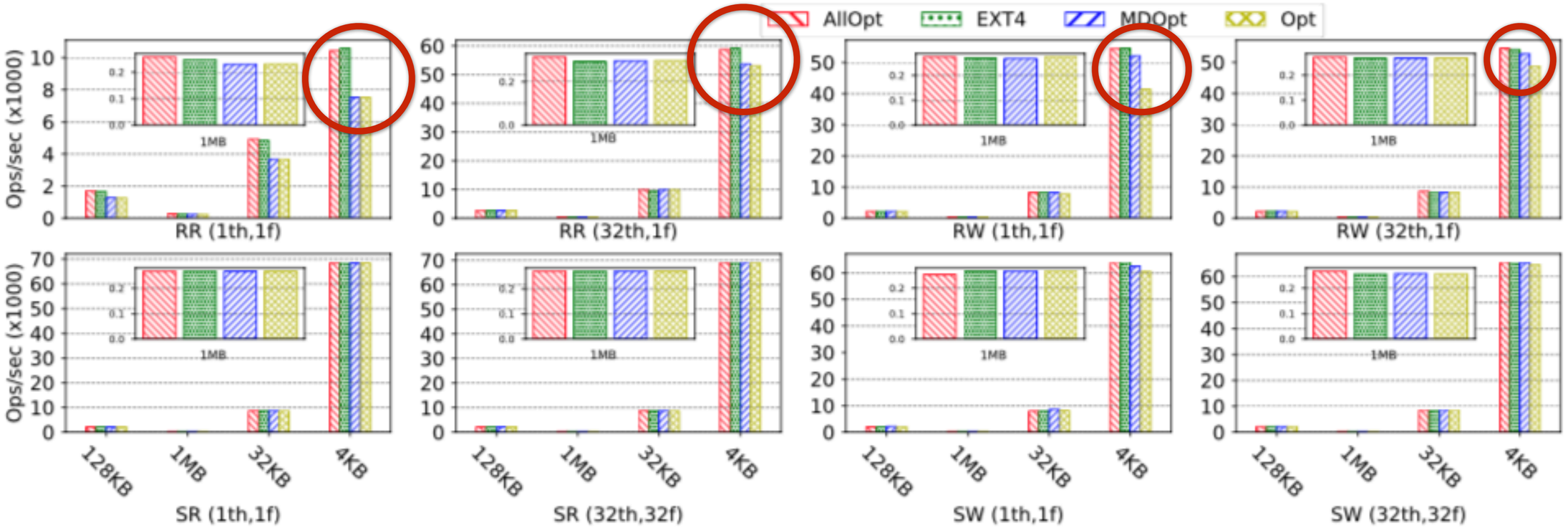


Figure 6: Throughput(ops/sec) for EXT4 and FUSE/ExtFUSE Stackfs (w/ xattr) file systems under different configs (Table 5) as measured by Random Read(RR)/Write(RW), Sequential Read(SR)/Write(SW) Filebench [48] data micro-workloads with IO Sizes between 4KB-1MB and settings *N*th: *N* threads, *N*f: *N* files. We use the same workloads as in [50].

Config	File System	Optimizations
`Opt` [50]	FUSE	128K Writes, Splice, WBCache, MltThrd
`MDOpt`	ExtFUSE	`Opt` + Caches `lookup`, `attrs`, `xattrs`
`AllOpt`	ExtFUSE	`MDOpt` + Pass `R/W` reqs through host FS

Table 5: Different Stackfs configs evaluated.

Real-world Applications

App Stats		CPU (%)		Latency (ms)	
Name	**OBB Size**	**D**	**P**	**D**	**P**
Disney Palace Pets 5.1	374MB	20	2.9	2235	1766
Dead Effect 4	1.1GB	20.5	3.2	8895	4579

Table 7: App launch latency and peak CPU consumption of `sdcard` daemon under default (D), and passthrough (P) settings on Android for two popular games. In passthrough mode, the FUSE driver never forwards `read`/`write` requests to user space, but always passes them through the host (EXT4) file system. See Table 5 for config details.

Conclusion

- ExtFUSE framework safely executes “thin” file system handlers in the kernel

- Developers can use ExtFUSE to
  - Cache metadata requests
  - Directly pass I/O requests to lower FS
  - Insert custom security checks in the kernel

Personal Opinions

- The things I do not like:
  - Too many new/modified lines of code for FUSE/ExtFUSE/User space file systems

Component	Version	Loc Modified	Loc New
FUSE kernel driver	4.11.0	312	874
FUSE user-space library	3.2.0	23	84
ExtFUSE user-space library	-	-	581

Table 3: Changes made to the existing Linux FUSE framework to support ExtFUSE functionality.

File System	Functionality	Ext Loc
StackFS [50]	No-ops File System	664
BindFS [35]	Mirroring File System	792
Android sdcard [24]	Perm checks & FAT Emu	928
MergerFS [20]	Union File System	686
LoggedFS [16]	Logging File System	748

Table 6: Lines of code (Loc) of kernel extensions required to adopt ExtFUSE for existing FUSE file systems. We added support for metadata caching as well as R/W passthrough.

5.7 K lines of code changes!

Personal Opinions

- The things I do not like:
  - Too many new/modified lines of code for FUSE/ExtFUSE/User space file systems
  - Absence of comparison to other novel user space file systems

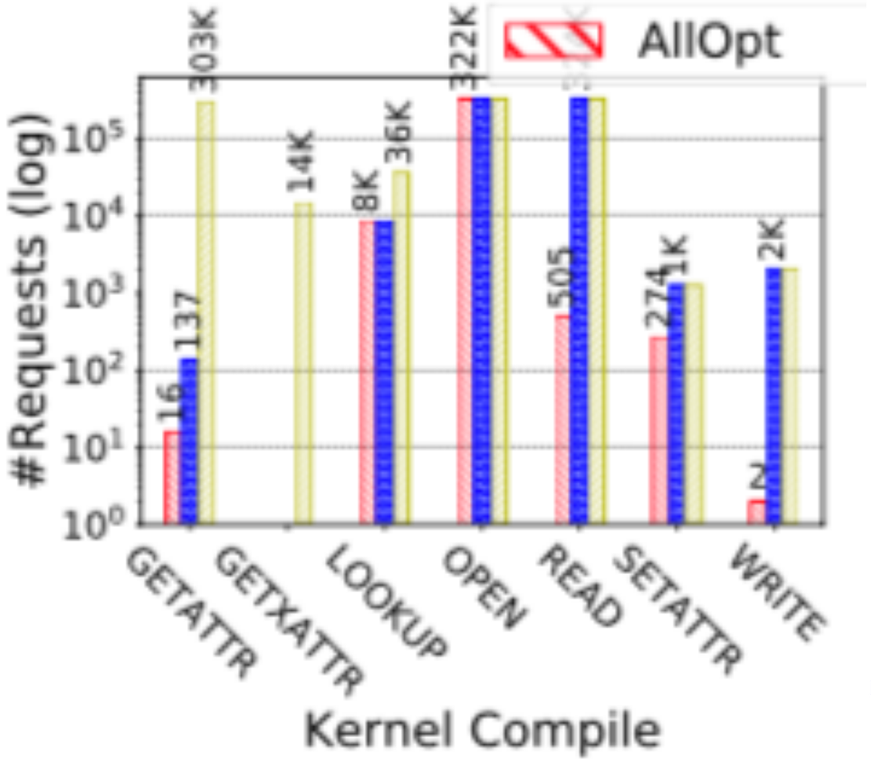
eBPF. ExtFUSE is not the first system to use eBPF for safe extensibility. eXpress DataPath (XDP) [27] allows apps to insert eBPF hooks in the kernel for faster packet processing and filtering. Amit et al. proposed Hyperupcalls [4] as eBPF helper functions for guest VMs that are executed by the hypervisor. More recently, SandFS [7] uses eBPF to provide an extensible file system sandboxing framework. Like ExtFUSE, it also allows unprivileged apps to insert custom security checks into the kernel.

FUSE. File System Translator (FiST) [55] is a tool for simplifying the development of stackable file system. It provides *boilerplate* template code and allows developers to only implement the core functionality of the file system. FiST does not offer safety and reliability as offered by user space file system implementation. Additionally, it requires learning a slightly simplified file system language that describes the operation of the stackable file system. Furthermore, it only applies to stackable file systems.

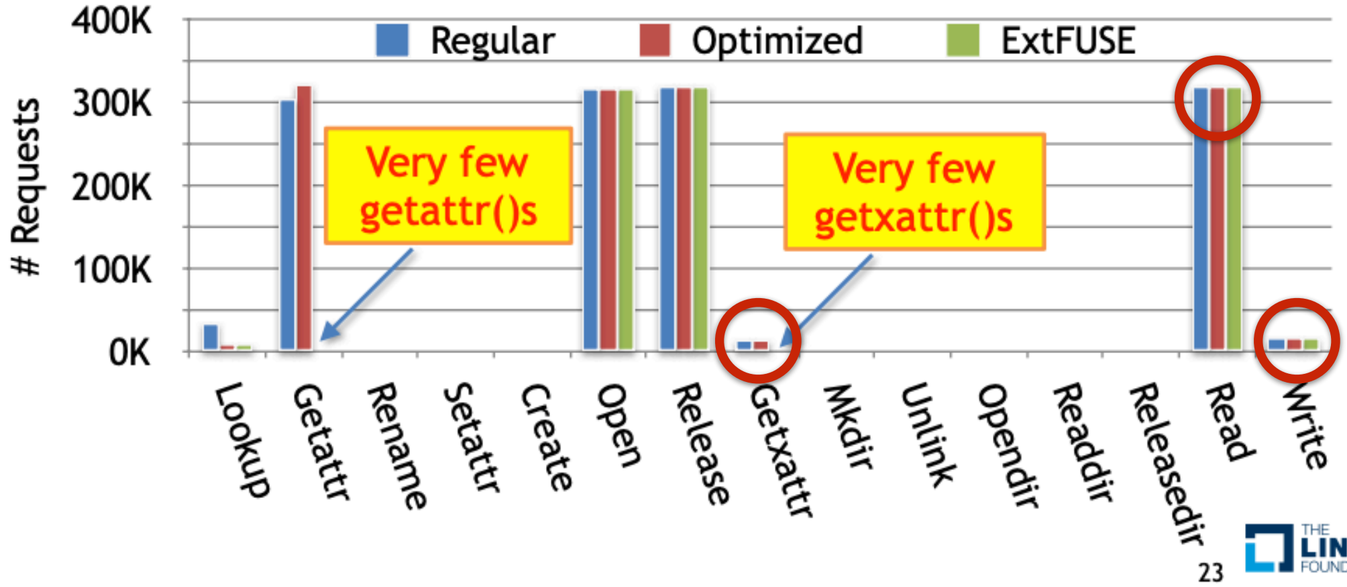
Narayan et al. [34] combined in-kernel stackable FiST driver with FUSE to offload data from I/O requests to user space to apply complex functionality logic and pass processed results to the lower file system. Their approach is only applicable to stackable file systems. They further rely on static per-file policies based on extended attributes labels to enable or disable certain functionality. In contrast, ExtFUSE downloads and safely executes thin extensions from user file systems in the kernel that encapsulate their rich and specialized logic to serve requests in the kernel and skip unnecessary user-kernel switching.

Personal Opinions

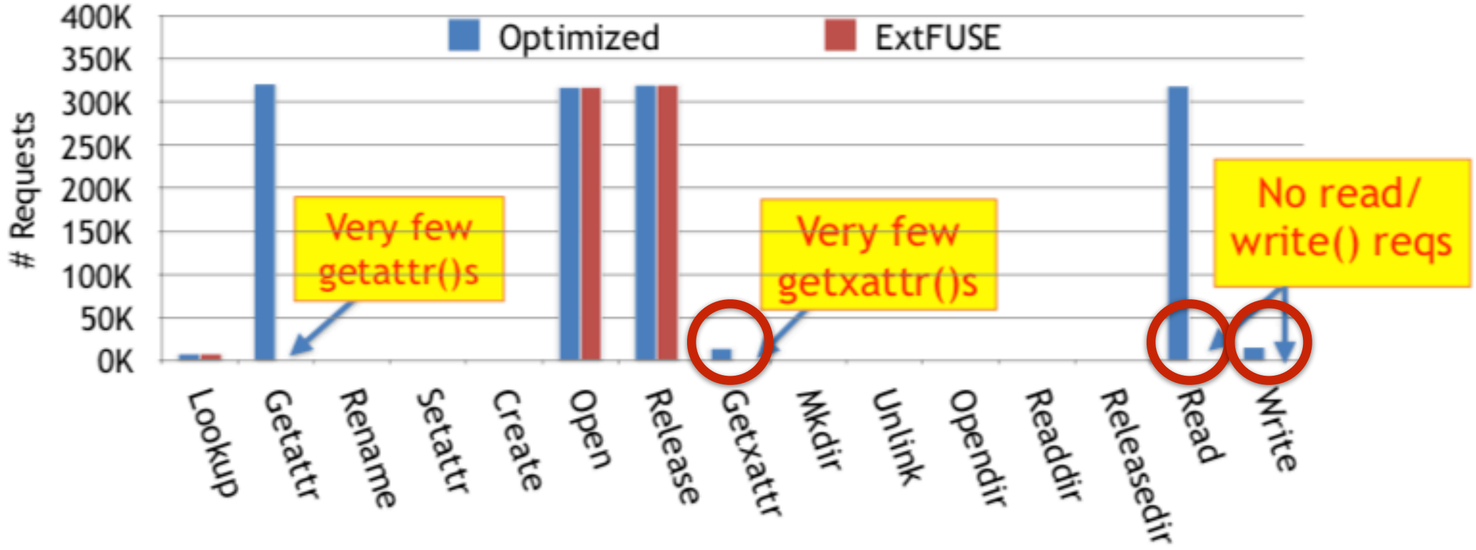
- The things I do not like:
  - Too many new/modified lines of code for FUSE/ExtFUSE/User space file systems
  - Absence of comparison to other novel user space file systems
  - Inconsistent results from the paper, conference slides and slides from other talks



- ***"cd linux-4.17; make tinyconfig; make -j4"***



https://events19.linuxfoundation.org/wp-content/uploads/2017/11/When-eBPF-Meets-FUSE-Improving-Performance-of-User-File-Systems-Ashish-Bijlani-Georgia-Tech.pdf



Personal Opinions

- The things I do not like:
  - Too many new/modified lines of code for FUSE/ExtFUSE/User space file systems
  - Absence of comparison to other novel user space file systems
  - Inconsistent results from the paper, conference slides and slides from other talks

- The things I like:
  - The Design is modular, extendable and compatible with FUSE
  - Identification of Optimization opportunities
  - Consider the balance between safety and performance
  - Built on the top of state-of-the-arts (FUSE and eBPF)

Thanks for Your Attention!

Discussions

- Kernel Space vs. User Space

Discussions

- Generality or Specialization