Hyatt Regency Santa Clara, CA
September 15-17, 2025

Always-On Diagnostics

eBPF-Powered Insights for Linux SMB and NFS Clients

Meetakshi Setiya
Software Engineer, Microsoft

www.sniadeveloper.org

Introduction

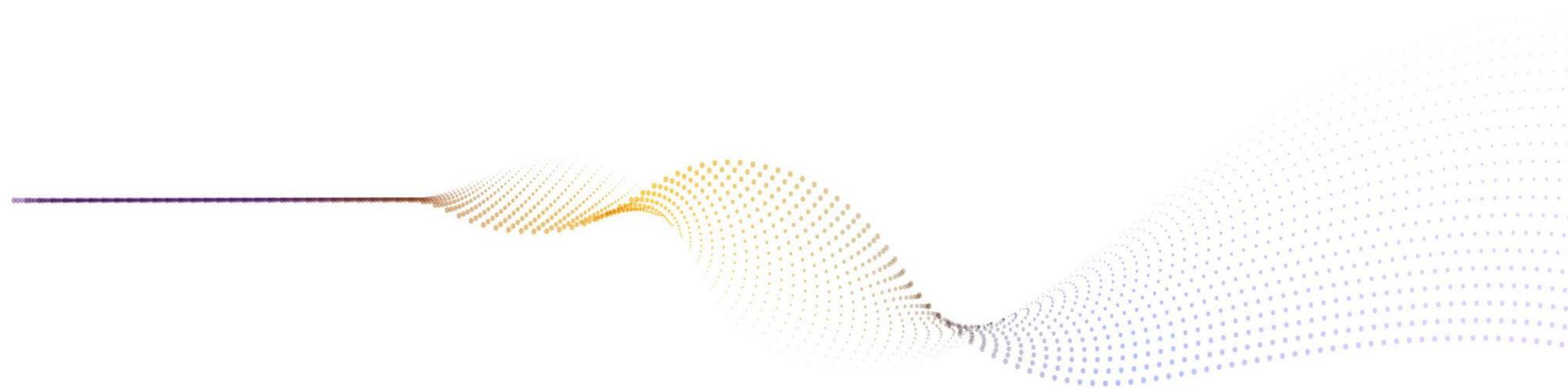
- Developer in the Azure Files team at Microsoft
- Improving Linux customers' experience with Azure Files
- Contributing to the development and enhancement of the Linux SMB client.

©2025 SNIA Developer Conference. All Rights Reserved.

Agenda

- Background
- Introduction to AOD
- Demo
- AOD Architecture : Deep Dive
- Performance Benchmarks
- Future Improvements

 ©2025 SNIA Developer Conference. All Rights Reserved.



Background

©2025 SNIA Developer Conference. All Rights Reserved.

SDC 25

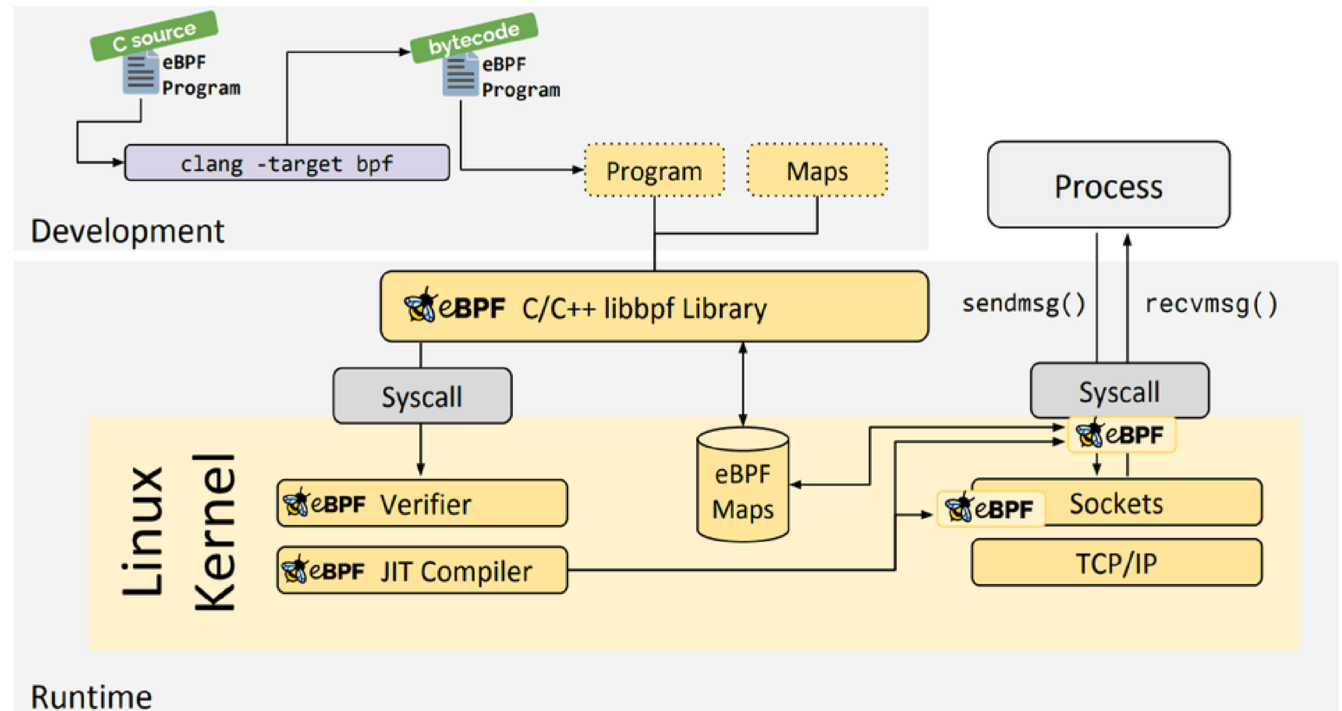
Existing Debugging Tools

- procfs
- Tracepoints
- strace
- Other commands/utilities – cifsiostat, lsof, ss, tshark

 ©2025 SNIA Developer Conference. All Rights Reserved.

Why BPF CO-RE?

- eBPF helps run sandboxed, event-driven programs in the kernel context without changing source code
- Hook onto any tracepoint, kernel-level and user-level functions, etc.
- CO-RE: Compile Once – Run Everywhere.
- Kernel BTF info + Clang extensions – BPF loader adjusts compiled BPF program for that kernel.



https://ebpf.io/what-is-ebpf

 ©2025 SNIA Developer Conference. All Rights Reserved.

SDC 25

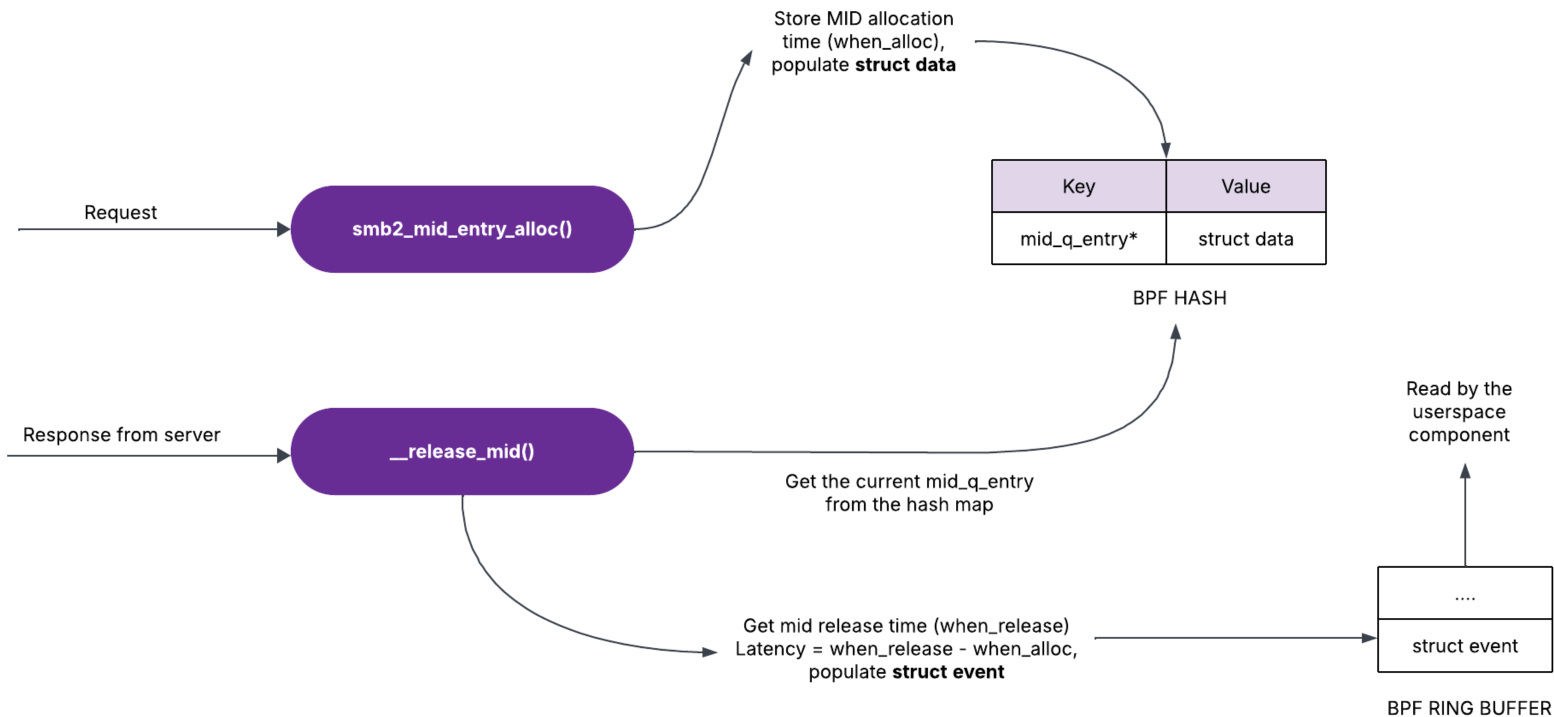
New eBPF Scripts

- **smbslower**: trace all SMB operations slower than a threshold.
- **smbvfsslower**: trace slow VFS callbacks for SMB operations.
- **smbvfssiosnoop**: trace I/O at the VFS layer for SMB operations (function arguments and return values of the callbacks)
- **nfsslower**: trace all NFS operations slower than a threshold.

 ©2025 SNIA Developer Conference. All Rights Reserved.

SDC 25

Smbslower Implementation

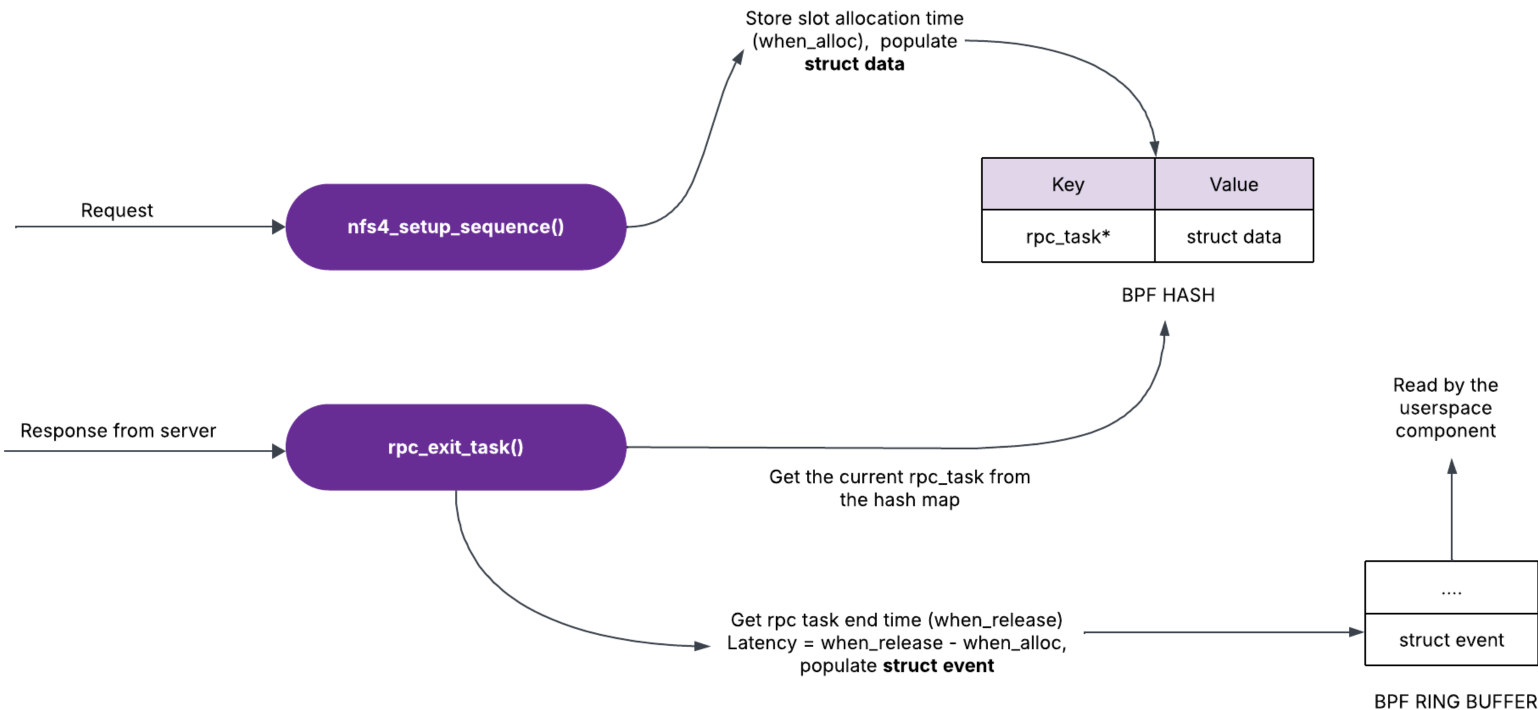



```
struct data {
    unsigned long long when_alloc;
    unsigned long long session_id;
    unsigned long long id;
    unsigned long long mid;
    unsigned short smbcommand;
    char is_compounded;
    char is_async;
    char task[TASK_COMM_LEN];
};
```

```
struct event {
    pid_t pid;
    unsigned long long when_release_us;
    unsigned long long delta_us;
    unsigned long long session_id;
    unsigned long long id;
    unsigned long long mid;
    unsigned short smbcommand;
    char is_compounded;
    char is_async;
    char task[TASK_COMM_LEN];
};
```

 ©2025 SNIA Developer Conference. All Rights Reserved.

Nfsslower Implementation




```
struct data {
    unsigned long long when_alloc;
    char sessionid[NFS4_MAX_SESSIONID_LEN];
    unsigned int slot_nr;
    unsigned int seq_nr;
    char task[TASK_COMM_LEN];
};
```

```
struct event {
    pid_t pid;
    unsigned long long when_release_us;
    unsigned long long delta_us;
    char sessionid[NFS4_MAX_SESSIONID_LEN];
    unsigned int slot_nr;
    unsigned int seq_nr;
    unsigned short nfscommand;
    char task[TASK_COMM_LEN];
};
```

 ©2025 SNIA Developer Conference. All Rights Reserved.

SDC 25

©2025 SNIA Developer Conference. All Rights Reserved.

Why AOD?

- Transient, difficult to reproduce failures.
- Persistent observation needed to catch intermittent anomalies.
- Manual diagnostics are slow, time-consuming.
- Need for low overhead smb/nfs client-side continuous monitoring tooling, especially in complex environments like Kubernetes.

 ©2025 SNIA Developer Conference. All Rights Reserved.

SDC 25

Always On-Diagnostics

- A multi-threaded Python Daemon that listens to events captured by the eBPF tools.
- Will run as a systemd service, with resource limits set in the systemd service file.
- Periodic examination of events and detection of configured anomalies.
- Triggers automatic log collection on anomaly.
- Periodic cleaning of older log bundles, as specified in the AOD config file.
- Designed to have minimal interference with the kernel hot-path.
- Low CPU and memory footprint.
- Highly configurable.

 ©2025 SNIA Developer Conference. All Rights Reserved.

Sample Config

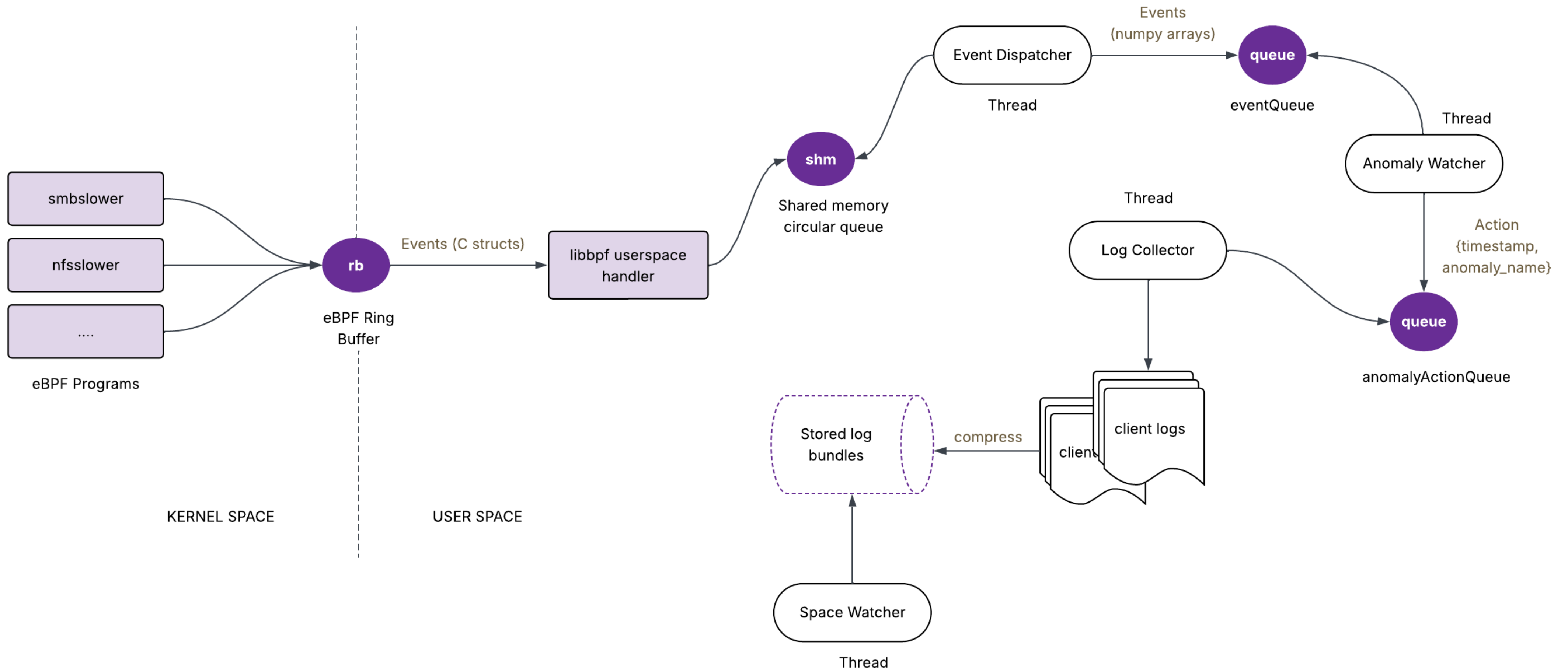
```yaml
watch_interval_sec: 1
aod_output_dir: /var/log/aod

guardian:
  anomalies:
    latency:
      type: "Latency"
      tool: "smbslower"
      mode: "all"
      acceptable_count: 10
      default_threshold_ms: 20
      track_commands:
        - command: SMB2_WRITE
          threshold: 50
      actions:
        - dmesg
        - journalctl
        - debugdata
        - stats
        - mounts
        - syslogs

cleanup:
  cleanup_interval_sec: 60
  max_log_age_days: 2
  max_total_log_size_mb: 0.5

audit:
  enabled: true
```

- Tracking modes: “all”, “trackonly”, “excludeonly”.
- Use either track_commands (trackonly) or exclude_commands (excludeonly).
- Mode type will determine which one is used.
- With mode type “all”, track commands with their custom thresholds, track other commands with the default threshold, and ignore the excluded commands.
- Can chain more anomaly types using this syntax.

 ©2025 SNIA Developer Conference. All Rights Reserved.

SDC 25

Overall Flow



 ©2025 SNIA Developer Conference. All Rights Reserved.

```
azureuser@sreyasl-aod-perf:~/linux_diagnostics$
```

```
azureuser@sreyasl-aod-perf:~$
```

```
azureuser@sreyasl-aod-perf:/var/log/aod/batches$
```

22°C
Partly sunny

Search

08:54
01-07-2025

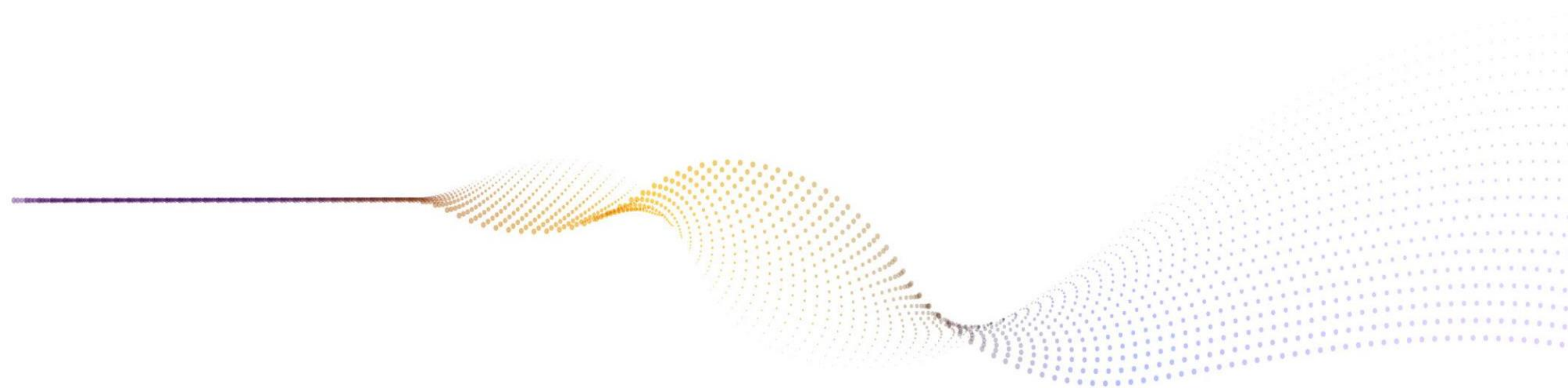
Installation

```
# Check Python version (requires 3.9+)
python3 --version

git clone https://github.com/sprasad-microsoft/linux_diagnostics.git
cd linux_diagnostics

# Install dependencies. Venv recommended
pip3 install -r requirements.txt

sudo python3 src/Controller.py
```

https://github.com/sprasad-microsoft/linux_diagnostics/blob/main/USAGE.md

 ©2025 SNIA Developer Conference. All Rights Reserved.

SDC 25



AOD Architecture

 ©2025 SNIA Developer Conference. All Rights Reserved.

Component Breakdown: AOD Controller

- Central controller process – initializes and orchestrates other components.
- Loads the config, sets up channels for inter-thread (queues) and inter-process communication (SHM).
- Starts and supervises the rest of the threads/processes, restart if exit unexpectedly.
- Manages the service lifecycle, coordinates graceful shutdown.

 ©2025 SNIA Developer Conference. All Rights Reserved.

Component Breakdown: eBPF Programs

- Spawned as a separate process by the Controller.
- Attach to the requested NFS/SMB code-paths and sends events back to the user space.
- Maximal filtering (command, error code, latency, etc.) within the kernel space BPF code, so that kernel--> user space buffer does not get choked.
- Construct the “event” struct for each valid operation, send it to user space via eBPF ringbuffer.

```
union metrics {
    unsigned long long latency_ns;
    int retval;
};

struct event {
    int pid;
    int command;
    unsigned long long cmd_end_time_ns;
    union metrics metric;
    char tool;
    char task[TASK_COMM_LEN];
};
```

 ©2025 SNIA Developer Conference. All Rights Reserved.

SDC 25

Component Breakdown: eBPF Programs

- One ring(buffer) to rule them all (i.e. pinned map). Delivers a stream of ordered events from all eBPF tools.
- One shared user space loader and event-handler program (MPSC).
- The user-space handler waits on the ringbuffer for events (blocking), drains all events and sends them to an in-memory circular queue to be consumed by the EventDispatcher thread.
- Kernel-->user space wakeup latency; batching events prevents waking up the user-space component too often.

 ©2025 SNIA Developer Conference. All Rights Reserved.

Component Breakdown: Event Dispatcher

- Managed by the controller as a separate thread.
- Shared memory with the eBPF userspace handler. Reads raw C structs from the shared circular buffer (SPSC).
- Reads the entire SHM and bulk converts all the raw C struct bytes to an equivalent, memory-aligned NumPy structured array.
- Pushes the NumPy array to eventQueue to be picked up by Anomaly Watcher thread.

```
class Metrics(ctypes.Union):
    _fields_ = [("latency_ns", ctypes.c_ulonglong), ("retval", ctypes.c_int)]


class Event(ctypes.Structure):
    """Event c struct."""

    _fields_ = [
        ("pid", ctypes.c_int),
        ("cmd_end_time_ns", ctypes.c_ulonglong),
        ("command", ctypes.c_ushort),
        ("metric", Metrics),
        ("tool", ctypes.c_ubyte),
        ("task", ctypes.c_char * TASK_COMM_LEN),
    ]
```

```
event_dtype = np.dtype(
    [
        ("pid", np.int32),
        ("cmd_end_time_ns", np.uint64),
        ("command", np.int16),
        ("metric_latency_ns", np.uint64),
        ("tool", np.uint8),
        ("task", f"S{TASK_COMM_LEN}"),
    ],
    align=True,
)
```

 ©2025 SNIA Developer Conference. All Rights Reserved.

SDC 25

Component Breakdown: Anomaly Watcher

- Sleeps for “watch_interval_sec”, consumes the NumPy structured arrays from eventQueue on waking up.
- Masks events based on the eBPF tool type for targeted anomaly analysis.

```python
def _generate_action(self, anomaly_type: AnomalyType) -> dict:
    """Generate an action based on the detected anomaly."""
    timestamp_ns = int(time.time() * 1e9)  # nanoseconds since epoch
    return {
        "anomaly": anomaly_type,
        "timestamp": timestamp_ns,
    }
```

- Multiple anomaly handlers with custom rules and configured thresholds to detect any anomalies in their batch of events.
- NumPy for analysis at native C speed.
- If anomaly detected, logs to syslog and pushes “action” to the anomalyActionQueue to be picked up by the Log Collector thread.

 ©2025 SNIA Developer Conference. All Rights Reserved.

SDC 25

Component Breakdown: Log Collector

- Waits to get any actions in the anomalyActions queue.
- Configurable log-collection tasks for each anomaly type.
- Utilizes Python's asyncio framework to facilitate concurrent log collection for multiple anomalies (max number of anomalies bounded by semaphore).
- Compresses the log files.

```
└── /var/log/aod/batches/
    ├── aod_quick_<batch_id1>/
    │   ├── dmesg.log
    │   ├── cifsstats.log
    │   ├── debugdata.log
    │   ├── mounts.log
    │   └── ... and so on
    ├── aod_quick_<batch_id2>.tar.zst
    ├── aod_quick_<batch_id3>.tar.zst
    └── ... and so on
```

 ©2025 SNIA Developer Conference. All Rights Reserved.

SDC 25

Component Breakdown: Space Watcher

- Housekeeping thread that periodically deletes old logs.
- Honors the retention policy (allowed cumulative size or duration of logs provided in the config) and purges the oldest tarballs.
- Ensures the diagnostic data footprint remains small and confined to the specified disk space limits.

 ©2025 SNIA Developer Conference. All Rights Reserved.

Performance Benchmarks

 ©2025 SNIA Developer Conference. All Rights Reserved.

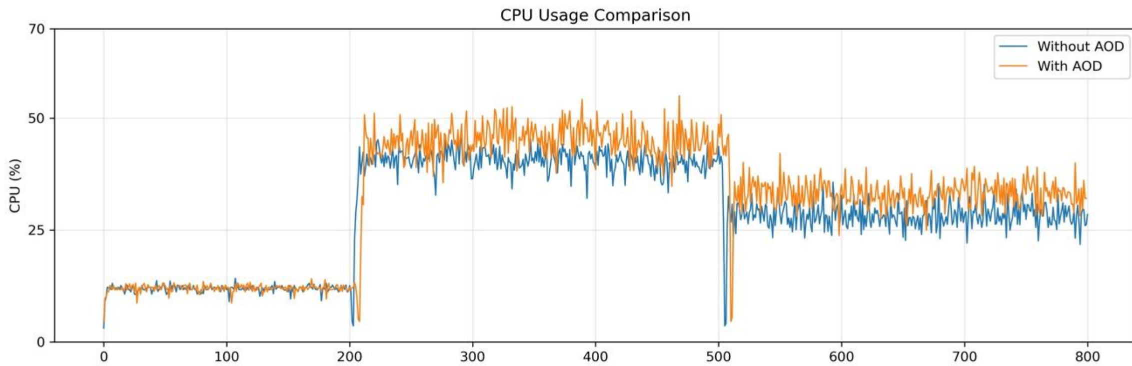
Setup Details

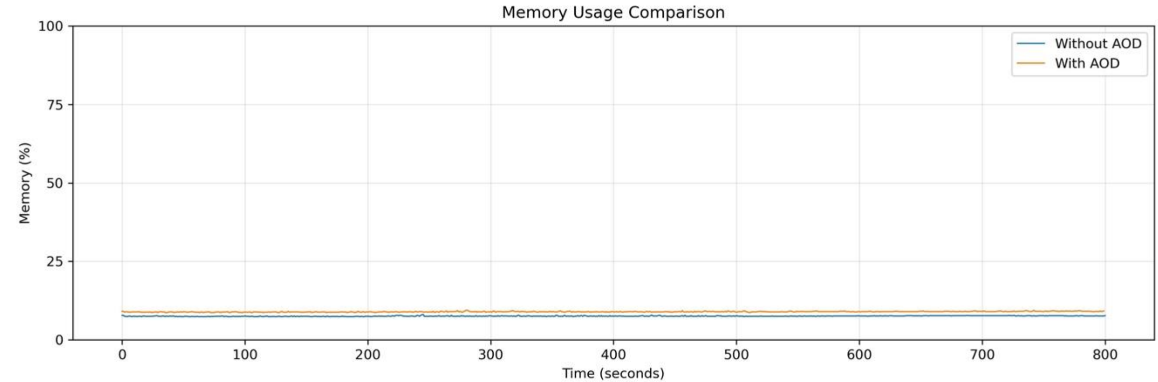
- D2S8 VM running Ubuntu-Azure 24.04, 2 CPUs, 8 GB memory.
- Fio workload 13 mins + 2 mins warmup
  - File Creation
  - Small random IO
  - Large Sequential IO

 ©2025 SNIA Developer Conference. All Rights Reserved.

SDC 25

“Good” config



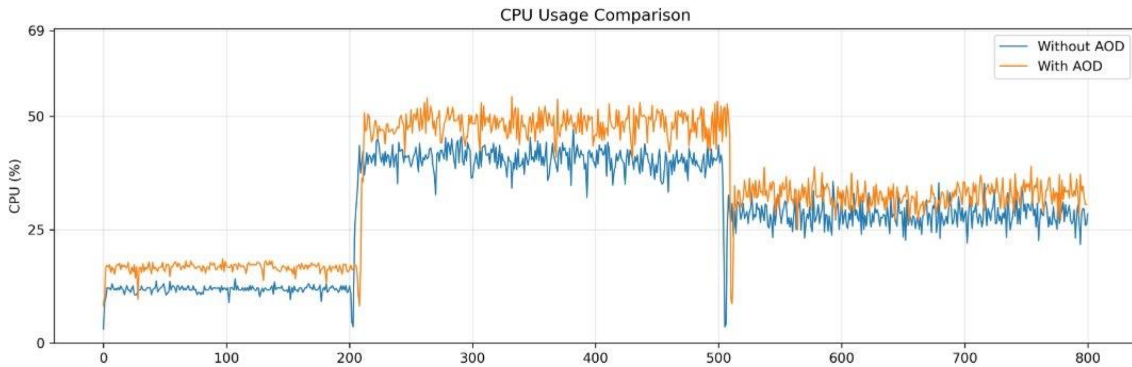


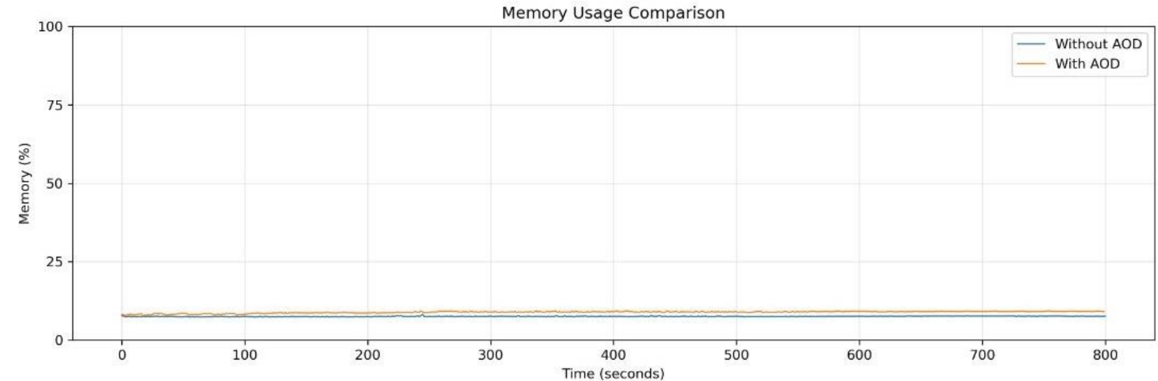
Metric	Without AOD	With AOD	Δ (AOD Impact)
CPU Usage	Avg: 28.59%, Max: 47.20%	Avg: 31.97%, Max: 55.00%	**+3.38% avg,** +7.80% on max
Memory Usage	Avg: 7.57%, Max: 8.10%	Avg: 8.96%, Max: 9.40%	**+1.39% avg,** +1.30% on max

- SMB Latency anomaly monitoring via Smbslower
- Per-second anomaly watching, reasonable latency thresholds for anomaly flagging

 ©2025 SNIA Developer Conference. All Rights Reserved.

“Suboptimal” config





Metric	Without AOD	With AOD	Δ (AOD Impact)
CPU Usage	Avg: 28.59%, Max: 47.20%	Avg: 34.22%, Max: 54.40%	**+5.63% avg,** +7.20% on max
Memory Usage	Avg: 7.57%, Max: 8.10%	Avg: 8.90%, Max: 9.40%	**+1.33% avg**, +1.30% on max

- SMB Latency anomaly monitoring via Smbslower
- Per-second anomaly watching, flag all events as anomalous (0 latency threshold).

 ©2025 SNIA Developer Conference. All Rights Reserved.

Idle Share (no-workload)

Metric	Without AOD	With AOD	Δ (AOD Impact)
CPU Usage	Avg: 2.83%, Max: 4.10%	Avg: 2.91%, Max: 5.10%	**+0.08% avg,** +1.00% on max
Memory Usage	Avg: 20.31%, Max: 20.40%	Avg: 20.41%, Max: 20.50%	**+0.10% avg,** +0.10% on max

- SMB Latency anomaly monitoring via Smbslower
- No active operation on the share.

 ©2025 SNIA Developer Conference. All Rights Reserved.

 ©2025 SNIA Developer Conference. All Rights Reserved.

Extensions

- Support observation of more events – lease-break timeouts, reconnect failures, TCP and socket level monitoring, etc.
- Improve the anomaly detection algorithms
- Chaining anomalies – hitting one anomaly triggers detection of others.
- Integrating support for long-running log captures, tracepoints, etc.
- Observer mode – for periodic log collection, in addition to on-anomaly mode.
- Improve API ref to enable anyone to add support for watching more anomalies.

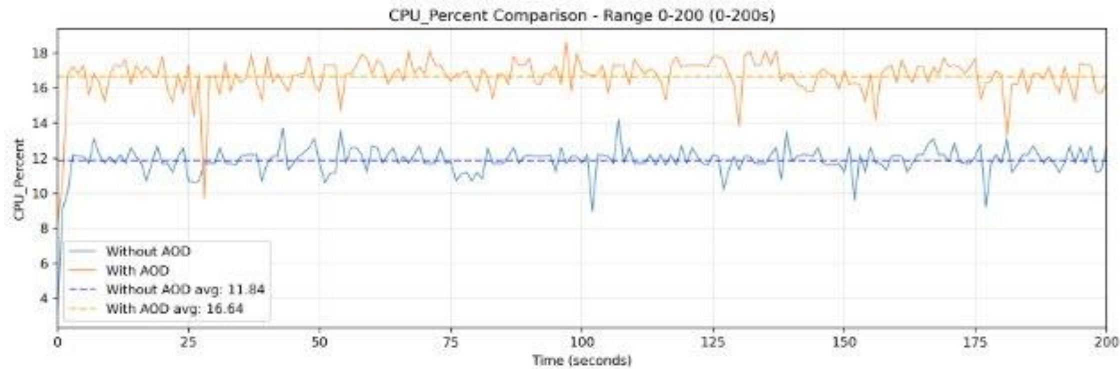
 ©2025 SNIA Developer Conference. All Rights Reserved.

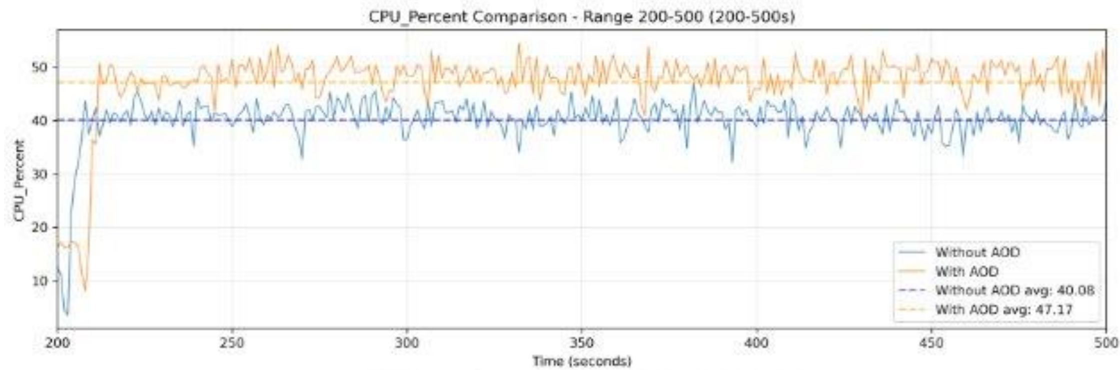
Optimizations

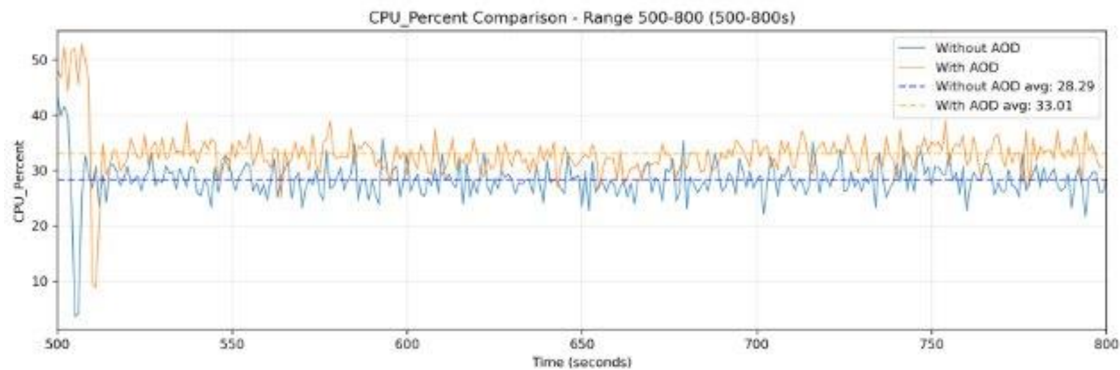
- Removing the C-Python SHM to avoid double-copy of the data.
- More places where asyncio can help!
- Offloading log compression to a separate thread.
- Add fault tolerance and robustness to the Space Watcher and for log compression.

 ©2025 SNIA Developer Conference. All Rights Reserved.

Backup (Suboptimal Config Perf)

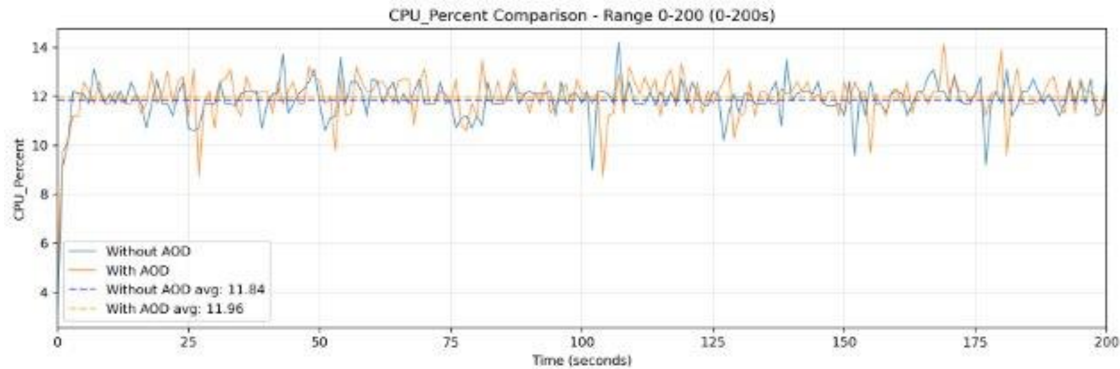


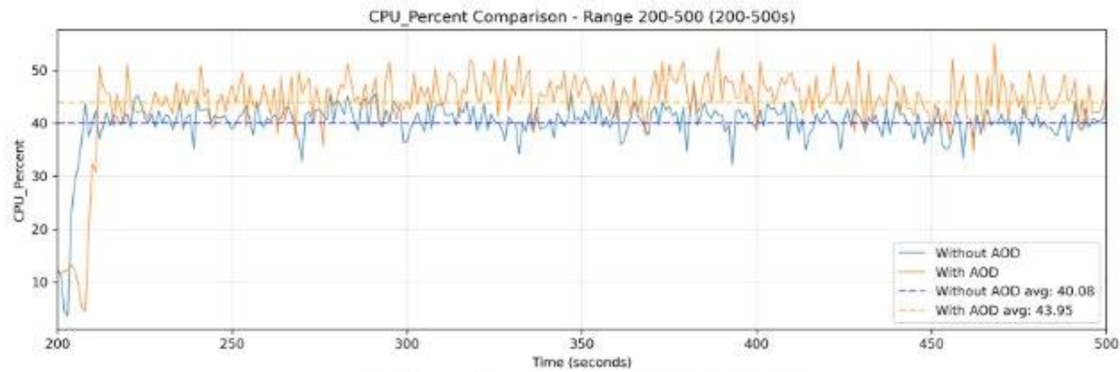


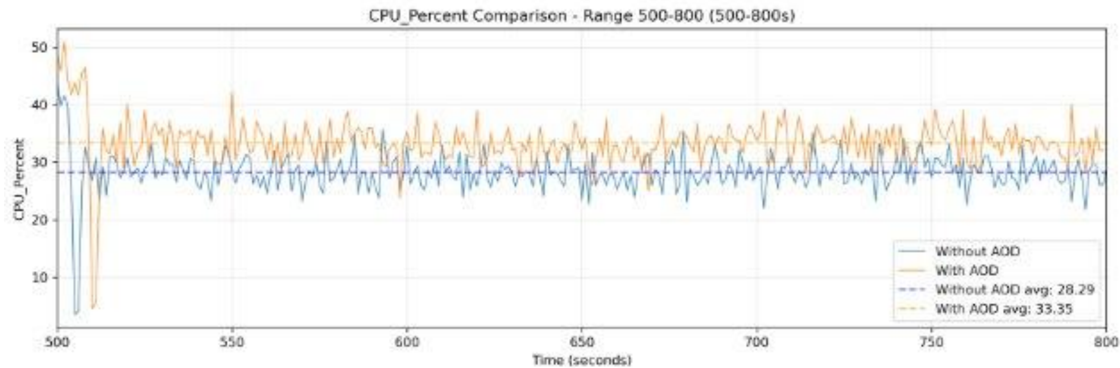


 ©2025 SNIA Developer Conference. All Rights Reserved.

Backup (Good Config Perf)







 ©2025 SNIA Developer Conference. All Rights Reserved.

Backup: How to support observing new anomalies

- Create your eBPF tool. To make it compatible with AOD, use the same pinned ring buffer used by the rest of the programs, and send the output to the ring buffer in the common “event” struct.
- Use the same userspace handler for all eBPF programs.
- In the controller, provide support for invoking your eBPF tool.
- Add the anomaly handler – the logic to flag anomalies in the batch of events given by your eBPF scripts.
- Add the log collection handlers (if they do not exist already).

More details here: https://github.com/sprasad-microsoft/linux_diagnostics/blob/main/USAGE.md

API Ref: https://github.com/sprasad-microsoft/linux_diagnostics/blob/main/docs/API_REFERENCE.md

 ©2025 SNIA Developer Conference. All Rights Reserved.

References

- https://ebpf.io/what-is-ebpf/
- https://nakryiko.com/posts/bpf-portability-and-co-re/#bpf-loader-libbpf
- https://nakryiko.com/posts/bpf-core-reference-guide/
- https://www.deep-kondah.com/deep-dive-into-ebpf-ring-buffers/
- https://nakryiko.com/posts/bpf-ringbuf/
- https://patchwork.ozlabs.org/project/netdev/patch/20200529075424.3139988-5-andriin@fb.com/
- https://superfastpython.com/why-not-always-use-processes-in-python/
- https://superfastpython.com/gil-removed-from-python/
- https://superfastpython.com/numpy-vs-gil/
- Official eBPF and Python docs

- eBPF scripts: https://github.com/meetakshi253/libbpf-bootstrap
- AOD: https://github.com/sprasad-microsoft/linux_diagnostics
- SambaXP presentation on the eBPF scripts: https://www.youtube.com/watch?v=OUbX23OAhb4

 ©2025 SNIA Developer Conference. All Rights Reserved.

Please remember to rate this session. You get access the presentations at http://sniadeveloper.org/conference

©2025 SNIA Developer Conference. All Rights Reserved.