A Deep Analysis of Performance Metrics and Comparative Assessment of Network Telemetry Tools in Linux Environments

ABSTRACT

As cyber-attacks targeting public cloud infrastructure increase in severity, it is essential to have strong network security measures for Linux machines. [1] Recent statistics underscore the severity of the situation, with a significant 39% of businesses experiencing security breaches within their cloud environments in 2022. This data shows a notable 35% increase in security attacks from the previous year. These breaches affected around 400 million individuals, emphasizing the urgent need for action.

As organizations increasingly migrate their operations to the cloud, addressing security risks is paramount. This needs a comprehensive approach to cloud security, focusing on monitoring and surveillance of cloud infrastructure usage by customers. Effective security observability requires deploying monitoring and alerting systems capable of promptly detecting and mitigating potential threats in real-time. [2] The Linux community has embraced Berkeley Packet Filter (BPF) technology as a cornerstone in this effort. BPF's flexibility and extensibility have led to the development of sophisticated tools, offering unparalleled capabilities in enhancing security observability and response mechanisms. This study begins by examining legacy solutions like auditd, which help auditing of all aspects of Linux machines. It also explores the origins and evolution of BPF within the Linux ecosystem, highlighting its transformative impact.

The study further delves into BPF-based monitoring tools tailored for scrutinizing Linux system processes. It elucidates their functionalities and meticulously assesses the performance of select tools and technologies. Rigorous experimental method, involving virtual machines with identical specifications subjected to network load simulations, ensures reliable and unbiased performance evaluations. Through this experimentation, valuable insights into resource consumption patterns for each tool are gleaned, aiding informed decision-making in tool selection and deployment strategies.

Keywords: Challenges with Auditd in network monitoring, bpftrace implementations in real-world scenarios, ebpf evolution and use cases, Linux network monitoring, performance comparison of bpf based network monitoring tools, security in public cloud.

1. INTRODUCTION

The paradigm shift towards cloud computing continues to accelerate, heralding a new era of technological innovation and enterprise transformation. As cloud technologies enable organizations to achieve more flexibility, growth, and performance, the digital environment becomes a place for both creating new solutions and fighting cyber-attacks. The proliferation of cloud-based infrastructures has unleashed a torrent of opportunities, yet it has also exposed a vulnerable underbelly susceptible to malicious exploits and cyber-attacks.

In cyber warfare annals, the year 2022 stands as a stark reminder of the perils that lurk in the digital abyss. Network intrusion, the specter haunting enterprises and organizations across the United States has appeared as a preeminent threat vector, accounting for 45% of all security incidents. Against this backdrop of escalating cyber threats, fortifying the bastions of network security within cloud infrastructures is of paramount importance.

Network security demands more attention and action than ever before, reverberating in the virtual pathways of cyberspace. As adversaries deploy increasingly sophisticated attack vectors to breach the sanctity of digital fortresses, it has become imperative to erect formidable defenses. Safeguarding the integrity and confidentiality of data traversing cloud networks requires a multifaceted approach that integrating innovative technologies, robust policies, and vigilant monitoring mechanisms.

The tripartite framework of security orchestration, encompassing the realms of collection, detection, and response, emerges as lodestar - guiding enterprises through the labyrinthine corridors of cybersecurity. In the crucible of digital warfare, data emerge as lifeblood coursing through the veins of network defenses, providing raw material for threat intelligence and situational awareness.

The pantheon of network security monitoring, a bastion of resilience amidst the tempest of cyber threats, stands as the vanguard of organizational defenses. Powered by a panoply of open-source and commercial tools, the armory of network defenders' bristles with the arsenal of threat detection, analysis, and mitigation capabilities. Yet, amidst the cacophony of tools and techniques, discerning the proverbial wheat from the chaff emerges as an arduous endeavor, demanding judicious choice and meticulous integration.

[2] BPF, a huge change that sparked a new era of network monitoring and analysis, is a key part of Linux development. BPF lets developers create custom networks - monitoring solutions in a secure and isolated environment. The ascendance of BPF-based tools, epitomized by stalwarts such as tcpdump, bcc, and bpftrace, augurs a new dawn in network telemetry, replete with insights and capabilities hitherto unimaginable.

In the crucible of technological innovation, this paper endeavors to unravel the labyrinthine corridors of network security monitoring, traversing the trodden paths of legacy solutions while charting the uncharted territories of BPF-based innovations. [3] Through a prism of inquiry and introspection, we shall dissect the nuances of network monitoring tools, navigating the turbulent seas of implementation challenges and performance bottlenecks.

* Tel.: +1-480-399-0786
Email address: prachi.arizona@gmail.com

[4] In conclusion, as the maelstrom of digital transformation continues to sweep across the global landscape, the clarion call for network security vigilance reverberates with undiminished fervor. [5] By embracing the ethos of continuous innovation and adaptive resilience, organizations can fortify their digital citadels against the ravages of cyber threats, ensuring the sanctity of their networks and the integrity of their data.

2. Network monitoring with tcpdump.

Reference [7] introduces the tcpdump utility as an influential and adaptable tool intrinsic to the Linux command line environment. Its primary function revolves around the capture and analysis of bidirectional network traffic, thereby offering users indispensable insights into network performance and security paradigms. A clear benefit of tcpdump is that it comes pre-installed with most Linux distributions for free, making deployment faster and avoiding manual installation steps. This helps rapid initiation of network analysis endeavors by users with requisite privileges.

[8] Tcpdump affords users granular control over packet capture operations through the specification of network interfaces and packet count limitations, as exemplified by the "interface" argument and packet count flag "-c 5." By adjusting these parameters, one can create capture settings that suit specific analysis goals. Moreover, the tcpdump offers an array of filtering mechanisms encompassing hostname, port number, and protocol type, augmenting the precision and relevance of captured data. The "x" flag helps packet content inspection in ASCII and Hexadecimal formats, enhancing data interpretability and facilitating nuanced analysis.

A scrutiny of tcpdump's implementation at scale reveals its facile integration across diverse computing environments facilitated by its inclusion within standard Linux distributions. This versatile tool makes it easy to record and save network traffic data for later examinations by creating .pcap files. These files serve as repositories of captured data amenable to analysis by tcpdump itself or compatible tools such as Wireshark. [9] The latter, distinguished by its intuitive user interface and robust filtering capabilities, emerges as an indispensable adjunct for network

administrators and analysts in dissecting the intricacies of network traffic.

[10] Pros:

1. Pre-installed Ubiquity: One of tcpdump's most lauded attributes is its ubiquity, as it comes pre-installed with most major Linux distributions. This ensures seamless integration into operational workflows, obviating the need for arduous installation rituals and facilitating rapid deployment across diverse network environments.
2. Detailed Network Interface Selection: Tcpdump allows users to control network interfaces in detail, enabling accurate packet capture from specific sources. Through the judicious use of interface arguments, users can tailor their analysis to specific network segments, thereby enhancing the fidelity and relevance of captured data.
3. Sophisticated Filtering Capabilities: Tcpdump offers a pantheon of filtering mechanisms, encompassing hostname, port number, and protocol type, among others. This affords users the flexibility to sculpt their analysis according to bespoke criteria, facilitating targeted investigation and elucidation of pertinent network anomalies.
4. Flexible Output Options: Tcpdump facilitates the seamless exportation of captured network traffic to .pcap files, ensuring the preservation and portability of captured data for later analysis. This versatile output format is compatible with a plethora of analysis tools, including the widely acclaimed Wireshark, thereby enhancing interoperability and facilitating collaborative analysis endeavors.

Cons:

1. Absence of HTTP Session Displays: A notable limitation of tcpdump is the absence of dedicated displays for HTTP sessions. Consequently, users are compelled to manually sift through voluminous packets to discern individual sessions, a laborious and time-intensive endeavor that can impede workflow efficiency and detract from the overall user experience.
2. Processing Overhead: Tcpdump's efficacy in capturing network traffic at scale may be marred by processing overhead, particularly in scenarios characterized by high throughput and data volume. This can manifest as latency in packet capture and analysis, potentially impeding real-time responsiveness and compromising the timeliness of threat detection and mitigation efforts.

3. Network monitoring with auditd.

[11] The Linux Audit system provides a robust framework for logging events within Linux environments. At its core, auditd facilitates comprehensive auditing capabilities, including monitoring filesystems, processes, and network activities. Notably, auditd comes pre-installed with most major Linux distributions, simplifying deployment and eliminating the need for manual setup.

The audit system includes two key components: Audit.rules and Audit.conf, located within the /etc/audit/ directory. Audit.rules allows for the configuration of auditing rules, while Audit.conf defines essential parameters such as log file location and buffer size.

You can start auditd easily with the auditctl command. However, it's important to note that initial network tracing may yield extraneous socket-related calls. To focus on IPv4 and IPv6 network connections, specific audit rules are configured accordingly.

In terms of scalability, installing auditd across various systems is standardized, with later configuration managed through auditctl. This ease of deployment and management is a significant advantage, particularly for organizations with diverse computing environments.

There are also utilities like ausearch and augenrules that complement auditd, which help with searching audit logs and applying durable audit rules, respectively. Additionally, tools like aulast and aulastlog provide insights into user login activities.

Despite its utility, auditd is not without scalability challenges. While installation and configuration are standardized, managing audit rules across many systems can become cumbersome. Furthermore, performance overhead, particularly during context switching and information transfer between kernel and user space, can be significant, impacting scalability in high-volume environments.

* Tel.: +1-480-399-0786
Email address: prachi.arizona@gmail.com

In conclusion, while auditd is a valuable part of network monitoring, organizations should consider its scalability limitations and complement it with appropriate tools to effectively navigate modern security landscapes.

[12] The growing challenge that practitioners face is how to manage the enormous amount of data generated by auditd effectively. ausearch is a useful tool for event analysis, but it becomes less effective when faced with the huge challenge of searching large data sets, especially in the context of vast infrastructures with millions of linked machines. In response to this exigency, a slew of sophisticated tools, exemplified by AuditBeat, has emerged within the market landscape. These solutions specialize in the aggregation of logs emanating from disparate nodes scattered across the organizational infrastructure. Using the features offered by such tools, administrators can create user-friendly dashboards, set up alerting systems, and design intelligent detection algorithms, thus strengthening their alertness against security threats and compliance requirements.

The audit framework has made some substantial progress in clarifying the complex routes taken by network packets, but it also has some inherent limitations. The main one is the noticeable slowness of auditd in data collection. This slowness is caused by the complicated process of switching and moving data from kernel space to user space, which affects auditd's performance profile. On the other hand, Berkeley Packet Filter (BPF) has an advantage by mostly doing computational tasks within kernel space, giving it a faster operational speed under certain operational contexts.

Another salient deficiency intrinsic to auditd pertains to its restricted concurrency model, wherein only a single program can be accommodated at any given instance. Therefore, after event records are received, they are permanently lost, preventing the preservation of a complete audit trail that shows system events. While dealing with these challenges, these limitations should be carefully considered when designing the security monitoring strategy and choosing appropriate tools that fit the specific needs of the organizational infrastructure.

4. Network monitoring with BCC.

[13] The Berkeley Packet Filter Compiler (BCC) represents a potent toolkit engineered to facilitate the development of efficient kernel tracing and manipulation programs, harnessing the capabilities conferred by the extended Berkeley Packet Filter (eBPF) paradigm. Noteworthy for its versatility, BCC streamlines the process of crafting BPF programs by affording developers the flexibility to express kernel instructions in C and construct a user-friendly front-end utilizing Python scripting. This discourse will concentrate on the utilization of BCC for network tracing endeavors.

A Python-scripted BCC program can attach kernel entry or exit points and get socket information and reveal complex details about network events. For comprehensive insights into authoring BCC programs or perusal of sample implementations, refer to the iovisor/bcc repository available at: BCC - Tools for BPF-based Linux IO analysis, networking, monitoring, and more (github.com).

You can find many tools in the BCC framework in the bin directory.

4.1. Examining the Challenges and Trade-offs with BCC: Evaluating Advantages and Drawbacks

The utilization of BCC hinges on a unique approach, wherein the compilation of Berkeley Packet Filter (BPF) code happens dynamically. However, this method engenders a spectrum of considerations warranting thorough scrutiny before embarking on its adoption:

1. Scalability Conundrums: While BCC stands as an invaluable asset for crafting intricate applications, its scalability is contingent upon the installation of kernel header packages on host machines. This requisite presents formidable hurdles when trying to deploy BCC across expansive networks comprising millions of interconnected machines. Additionally, the inherent intricacies of building a robust network tracing tool at scale pose formidable challenges, needing meticulous planning and resource allocation.
2. Problems with Clang Front-End Integration: BCC programs use the Clang

* Tel.: +1-480-399-0786
Email address: prachi.arizona@gmail.com

front-end to update BPF programs that users create. This link adds complexity, making it hard to find the root causes of problems. Troubleshooting becomes slow and costly, hurting the developer experience and slowing down operations.

3. Performance Quandaries: Notwithstanding its utility, the performance of BCC programs lags alternative BPF-based technologies, precipitating suboptimal outcomes, particularly in projects where performance optimization assumes paramount significance. This performance difference requires careful evaluation before using BCC for certain operational situations, to avoid reducing operational effectiveness.
4. Compromised Compilation Workflow: The compilation process inherent to BCC causes redundancy when deployed across a fleet of machines, engendering superfluous resource consumption, and escalating operational costs. This redundancy underscores the imperative of streamlining the compilation workflow to mitigate inefficiencies and enhance deployment agility in large-scale operational environments.

Given the many difficulties mentioned before, the effectiveness of BCC as a universal solution for large-scale and scalable BPF-based programs is questionable. Wise decision-making requires a careful balance of the advantages and disadvantages, making sure they fit with the needs of specific operational models and strategic goals.

5. Network monitoring with bpftrace: A comprehensive guide to effective network tracing.

[14] Bpftrace is a complex and advanced tracing language that was created specifically for eBPF (extended Berkeley Packet Filter) and works well with most Linux distributions. Crafted as an invaluable tool for system administrators and developers alike, its robust capabilities aid in system monitoring and troubleshooting endeavors. Leveraging LLVM as its backend compiler, Bpftrace seamlessly transcribes scripts into BPF bytecode, thereby ensuring optimal efficiency and performance, particularly in the realm of network monitoring. The bpftrace language combines powerful features with a syntax that is like awk and C and builds on earlier tools such as DTrace and System Tap.

Accessible through Linux package managers or compilation from open-source repositories, Bpftrace exhibits a tendency towards compatibility issues with certain Linux kernel versions when obtained directly from package managers, as per firsthand experiences. To mitigate such concerns, it is advisable to obtain the latest compiled version from the open-source repository, ensuring seamless functionality and compatibility. The repository link can be accessed here: iovisor/bpftrace: High-level tracing language for Linux eBPF (github.com).

Focusing on network tracing, we delve into select scripts within bpftrace tailored for TCP passive and active connections. These scripts give invaluable insights into network behavior, helping the identification of bottlenecks, diagnosis of issues, and optimization of performance. Links to pertinent scripts can be found below:

- Script for TCP passive connections: bpftrace/tools/tcpaccept.bt at master · iovisor/bpftrace (github.com)
- Script for TCP active connections: bpftrace/tools/tcpconnect.bt at master · iovisor/bpftrace (github.com)

[15] To start the scripts, the binary must be downloaded from the following link: Release v0.19.1 · iovisor/bpftrace (github.com).

This command initiates monitoring of active TCP connections and outputs pertinent details including process ID, time, command executing the TCP connection, source and destination IP addresses, and ports. Preliminary tests show minimal CPU usage and memory consumption, even on medium-sized virtual machines. Detailed results will be expounded later in this discourse.

Bpftrace stands out for its small installation size, having only 1 KB for the program itself and a compact binary. With an extensive repository of pre-made tracing tools, facile one-liner scripting capabilities, and broad support across various distributions, it emerges as a stalwart tool for scaling network observability. However, it's noteworthy that bpftrace remains in the BETA release phase for ARM 64 processors, signifying an area for potential improvement. We will explain later how to implement bpftrace effectively for network monitoring purposes.

* Tel.: +1-480-399-0786
Email address: prachi.arizona@gmail.com

5.1 BPFTrace Implementation at Scale: Overcoming Challenges for Linux Distributions:

Implementing bpftrace based logging was quite simple since it supports writing one liner programs. This simplicity greatly benefits developers when creating and maintaining their code.

There are a couple of things to note when starting to build scalable solutions for Linux machines. Since Linux supports various distributions and versions, finding the binary that's statically built becomes more important than ever. We will investigate the concepts of statically linked binaries shortly.

Until now, we have talked about the official binary release on bpftrace repository artifact release page. Here is the link: Releases · iovisor/bpftrace (github.com)

As of this writing, the binary provided on the above page is a dynamically linked binary, which means the dependencies will be pulled at runtime on the respective distributions. This becomes a challenge when building a widespread solution for various distributions. To support various distributions with this solution, we must build this binary using each distro and use that executable with the respective distros and kernel versions. Obviously, this solution is not going to scale with new Linux kernel versions and distributions growing rapidly over time. Hence, the need for a solution to create a statically linked binary became clear. This solution was designed separately to solve the portability issues with Linux.

Bpftrace also supports this solution and is explained in more detail over here : bpftrace/INSTALL.md at master · iovisor/bpftrace (github.com)

However, I would like to delve deeper into the Appimage tooling, as it is open source and is needed for implementing bpftrace on a larger scale. The Appimage team provides various tools and utilities to simplify the shipping and packaging process. One of the most used tools in Appimage is linuxdeploy. Linuxdeploy is an AppDir maintenance tool.

In the case of any modern build system such as CMake, you can use the regular make install commands to create a directory-like structure which will then be used by linuxdeploy for packaging. CMake also comes with an inbuilt parameter to specify where the files should be installed instead of the root directory, called DESTDIR. This feature allows developers to have more control over the installation process and support a cleaner system structure.

For our use case, we logged tcp connect events and it required us to attach our bpftrace script to tcpconnect kernel probe. It also required us to listen to the socket object to fetch the connection details. Tcpconnect script is available in the bpftrace open-source project and most of it can be used as-is. Once the script is ready, we can place it in some folder in our VM.

This will start recording all the events and printing them on Linux terminal. As I already mentioned the downside of using Linux terminal as consumer, we can also write these events into a file, but that will increase CPU consumption because of I/O operations.

To trace the UDP connections, there is no example script available in bpftrace. But we can write our own script and listen to the "udp_sendmsg" and "udp_recvmsg" probe to trace the outgoing and incoming UDP connections respectively.

Deploying bpftrace-based logging solutions presents a straightforward process, owing to its support for writing concise one-liner programs. This inherent simplicity significantly streamlines code creation and maintenance efforts, enhancing developer productivity and codebase manageability.

However, as we embark on building scalable solutions for Linux environments, several considerations come into play. Given the diverse landscape of Linux distributions and versions, the importance of statically linked binaries cannot be overstated. The quest for statically built binaries, which encapsulate all dependencies within the executable itself,

* Tel.: +1-480-399-0786
Email address: prachi.arizona@gmail.com

becomes imperative to ensure seamless deployment across heterogeneous environments.

Presently, the official binary release available on the bpftrace repository artifact release page is dynamically linked, causing runtime dependency resolution on respective distributions. This dynamic linkage poses a formidable challenge when orchestrating widespread deployments across diverse Linux ecosystems. To address this challenge effectively, a solution must be devised to create statically linked binaries, thereby obviating the need for dependency resolution at runtime.

Bpftrace, acknowledging the significance of this challenge, extends support for creating statically linked binaries, as elaborated in detail in the documentation available at: bpftrace/INSTALL.md. However, I advocate for a deeper exploration into the use of the Appimage tooling, an open-source initiative crucial for implementing bpftrace at scale. The Appimage ecosystem offers an array of tools and utilities geared towards simplifying the packaging and shipping processes. Notably, “linuxdeploy” appears as a pivotal tool within the Appimage toolkit, facilitating the maintenance of “AppDir” structures.

Incorporating modern build systems such as CMake enables developers to use conventional install commands to generate directory-like structures, which are later utilized by “linuxdeploy” for packaging. Furthermore, “CMake” offers the flexibility of specifying the installation destination via the DESTDIR parameter, empowering developers to exert greater control over the installation process and maintain an organized system structure.

In our specific use case, wherein we logged TCP connect events, integration required attaching our bpftrace script to the “tcpconnect” kernel probe and monitoring socket objects to fetch connection details. Leveraging the readily available “tcpconnect” script from the bpftrace open-source project sped up this process. Upon script preparation, deployment entailed placing it in a designated folder within the virtual machine.

This command initiates event recording and displays outputs on the Linux terminal. Recognizing the limitations of terminal-based consumption, an alternative approach involves directing event outputs to a file, albeit at the expense of increased CPU consumption due to heightened I/O operations.

While bpftrace offers extensive capabilities for tracing TCP connections, a similar out-of-the-box solution for UDP connections is absent. However, developers can craft custom scripts using “udp_sendmsg” and “udp_recvmsg” probes to monitor outbound and inbound UDP connections, respectively.

6. Performance comparison between audit and bpf based programs.

[16] In our relentless pursuit of network data insights, we embark on a meticulous examination of performance between two stalwart tools: BPFTrace and Auditd. Our endeavor revolves around network-based logging, with a keen focus on scrutinizing CPU and memory consumption to unravel the depths of efficacy and efficiency.

Leveraging standard TCP kernel probes as our benchmark, we embark on a journey to record events, delving into the comparative performance of these tools. Using the built-in file output features of both BPFTrace and Auditd, our analysis keeps a sharp focus, making sure a careful evaluation without unnecessary factors.

For our experimentation, we deploy a virtual machine endowed with robust specifications—an Intel Xeon processor boasting 4 cores and 16 GB of RAM. This judicious allocation of resources ensures an environment ripe for exhaustive performance analysis. Noteworthy is the adherence to standard operating system configurations, enhancing the universality of our findings.

* Tel.: +1-480-399-0786
Email address: prachi.arizona@gmail.com

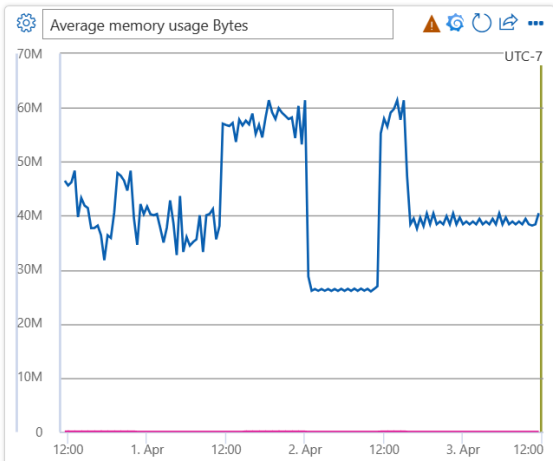

Fig 2. Average memory usage with bpftrace

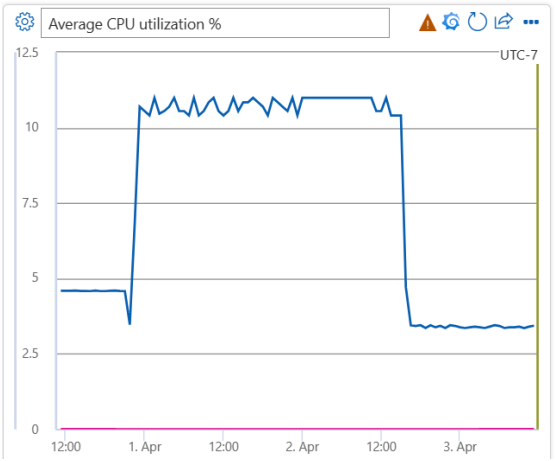

Fig 4. Average CPU utilization with bpftrace

Fig 2 and Fig 4 shows the graphs showing the average cpu and memory utilization when running bpftrace tracing 1000 connections/sec, which is huge for Linux machines. With peak load, it is seen that the CPU utilization went up to 11%, which is expected. Average memory usage was 60M, but this is a runtime memory usage. This doesn't include the memory used by bpftrace process to perform dynamic compilation using LLVM, which is approx. 150M. This makes bpftrace difficult to use for very small workloads. However, as part of last year's conference, it looks like the bpftrace is coming up with options to mitigate the dynamic compilation issue.

On the other hand, with the same load auditd were not able to perform such. Average CPU use for auditd with 1000 connections/sec was above 24%. The memory use was variable ranging from 150 to 300 MB.

Our investigation unfolds against a backdrop of concrete metrics:

- CPU and Memory Utilization with BPFTrace: Graphical representations unveil the intricate interplay of CPU and memory during BPFTrace's execution. With an imposing load of 1000 connections/sec, the CPU's use peaks at a modest 11%, a testament to its efficiency. However, the snapshot of memory usage—averaging 60M—barely scratches the surface, as it excludes the overhead incurred by dynamic compilation processes, pegged at approximately 150M. This nuance underscores the necessity of nuanced optimization efforts, especially considering the tool's unsuitability for minute workloads. Notably, sample sizes of 1000 connections/sec were rigorously selected to mirror real-world scenarios.
- Comparison with Auditd: In stark contrast, Auditd grapples with performance under identical loads. Bearing witness to CPU use exceeding 24% with 1000 connections/sec, coupled with a volatile memory footprint ranging from 150 to 300 MB, Auditd struggles to attain resource efficiency.

Amidst BPFTrace's commendable CPU performance, a labyrinth of limitations beckons:

- Script Complexity and Tooling Support: While BPFTrace excels in one-liner scripts, the specter of complexity looms large over intricate tasks. The absence of robust tooling support presents a bottleneck, hindering seamless integration with high-performance systems. Even though file-based output is a viable option, the risk of increased CPU usage because of more I/O operations requires urgent attention.
- Terminal Processing Limitations: The confines of the Linux terminal's print buffer impede concurrent print statement processing, fostering an environment ripe for data loss amidst surging event volumes.
- Memory Usage Issues: In the tough conditions of production environments, LLVM compilation and runtime memory usage cause problems for workloads with low RAM (< 2 GB), needing strong optimization efforts.

* Tel.: +1-480-399-0786
Email address: prachi.arizona@gmail.com

- Executable Compatibility: BPFTrace's flirtation with compatibility across Linux distributions presents a hidden danger to portability. Though recent efforts towards Appimage adoption show some promise, significant obstacles remain.

Despite Auditd's valiant efforts, particularly in terms of elevated CPU consumption, BPFTrace appears as the paragon of choice for constructing high-performance observability tools. However, a clear comprehension of BPFTrace's drawbacks guides the focused optimization efforts to unlock its full ability in creating effective and reliable observability solutions.

7. Conclusion

After trying out various tools, bpftrace stands out as the clear winner in terms of performance—a crucial factor in tool assessment for network observability, especially in the domain of the newest Linux kernel versions.

To set up an observability framework on a scale, both bpftrace and auditd necessitate added functionalities to be developed around them, optimizing their performance and ensuring event preservation without loss. Auditd, as a conventional logging system, lacks kernel name spacing, resulting in a deficiency of crucial details about the container that triggered network or system calls, making it less suitable for containerized environments. Furthermore, contemporary applications often encrypt network traffic, making packet capture prohibitively expensive and diminishing the efficacy of auditd in cloud-native settings.

While bpftrace and auditd were thoroughly evaluated, numerous other tools stay unexplored within this article's purview. When selecting the right tool for a specific use case, performance stays paramount. Additionally, the investigation primarily focused on monitoring virtual machines, while commercial products cater to containers and Kubernetes workload monitoring—a topic left unaddressed.

Integration ease with existing infrastructure, the learning curve associated with tool adoption, security, scalability, and customizability are crucial factors in tool choice for network observability. Keeping up with the latest trends, changes, and recommendations in network inspection tools helps to make smart choices that suit an organization's specific needs and infrastructure, as the field of network inspection tools is constantly changing.

* Tel.: +1-480-399-0786
Email address: prachi.arizona@gmail.com

Competing Interests

The author has declared that no competing interests exist.

References

[1] "39% of businesses faced a cloud environment data breach last year," Security Magazine. [Online]. Available: https://www.securitymagazine.com/articles/95044-of-businesses-faced-a-cloud-environment-data-breach-last-year. [Accessed: 01-May-2024].

[2] C. Humber, "eBPF — a new Swiss army knife in the system," Medium. [Online]. Available: https://medium.com/@chivierhumber/ebpf-a-new-swiss-army-knife-in-the-system-8964ad280eab. [Accessed: 01-May-2024].

[3] "Getting Started with BPFtrace: The Simple Guide" by Michael Arvanitopoulos [Accessed: 01-May-2024]

[4] “eBPF: Past, Present, and Future” by Thomas Graf [Accessed: 01-May-2024]

[5] D. Calavera and L. Fontana, "Linux Observability with BPF," 2020. [Accessed: 01-May-2024]

[6] "Introduction to BPF: A New Type of Software" by Jessie Frazelle [Accessed: 30-Apr-2024]

[7] "How to Capture Network Traffic in Linux with tcpdump," MakeUseOf. [Online]. Available: https://www.makeuseof.com/tag/capture-network-traffic-linux-tcpdump/. [Accessed: 01-May-2024].

[8] "Practical Packet Analysis: Using Wireshark to Solve Real-World Network Problems" by Chris Sanders [Accessed: 30-Apr-2024]

[9] "Packet Analysis with Wireshark & Tcpdump Cheat Sheet" by SANS Institute [Accessed: 01-May-2024]

[10] "TCPdump Tutorial: How to Sniff Network Traffic" by Laura Chappell [Accessed: 01-May-2024]

[11] "Linux Audit System: Anatomy of a System Call" by Michael Boelen [Accessed: 30-Apr-2024]

[12] "Auditing with Linux Auditd" by Red Hat [Accessed: 30-Apr-2024]

[13] iovisor, "bcc," GitHub. [Online]. Available: https://github.com/iovisor/bcc. [Accessed: 01-May-2024].

[14] iovisor, "bpftrace," GitHub. [Online]. Available: https://github.com/iovisor/bpftrace. [Accessed: 01-May-2024].

[15] "Introduction to BPFtrace: Dynamic Tracing for Linux" by Julia Evans [Accessed: 01-May-2024]

[16] "BPF Performance Tools" by Brendan Gregg [Accessed: 01-May-2024]

* Tel.: +1-480-399-0786
Email address: prachi.arizona@gmail.com