Rethinking the Networking Stack for Serverless Environments: A Sidecar Approach

Vishwanath Seshagiri
Emory University
Atlanta, USA

Abhinav Gupta
Columbia University
NYC, USA

Vahab Jabrayilov
Columbia University
NYC, USA

Avani Wildani
Cloudflare and Emory University
Atlanta, USA

Kostis Kaffes
Columbia University
NYC, USA

Abstract

Serverless platforms rely on legacy networking stacks for communication and data movement. We quantitatively analyze the performance of these stacks and show their mismatch with highly consolidated, virtualized modern serverless environments, focusing on Firecracker, the most common serverless virtualization framework. As serverless applications grow in complexity and interaction, the resulting network bottleneck is a prime source of user-perceived, end-to-end latency. In this paper, we present a detailed vision of a new, sidecar-based networking stack for serverless environments. Our primary design goal is to provide low-overhead networking while maintaining existing security guarantees. We outline the research challenges in both the control and the data plane that the community needs to tackle before such a sidecar architecture can be used in practice.

CCS Concepts

• **Software and its engineering → Operating systems**; • **Computer systems organization → Cloud computing**.

Keywords

Serverless Networking, Sidecar Architecture, Virtualization, Network Performance, Scalability, Isolation

ACM Reference Format:
Vishwanath Seshagiri, Abhinav Gupta, Vahab Jabrayilov, Avani Wildani, and Kostis Kaffes. 2024. Rethinking the Networking Stack for Serverless Environments: A Sidecar Approach. In *ACM Symposium on Cloud Computing (SoCC '24), November 20–22, 2024, Redmond, WA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3698038.3698561

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
SoCC '24, November 20–22, 2024, Redmond, WA, USA
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1286-9/24/11
https://doi.org/10.1145/3698038.3698561

1 Introduction

Serverless computing is a popular cloud computing paradigm, offering developers a pay-per-use model of application development where the cloud provider takes responsibility for running, scaling, and failure recovery of the application functions. These functions are typically isolated, short-lived, and have small CPU and memory footprints, requiring packing hundreds of them in each server [31, 32]. While there is a significant body of work on managing points of tension in these deployments, including scheduling [15, 16], reducing start-up overheads [26, 37, 43], and sidecar proxies [27], we argue that the networking stack is a critical, overlooked, bottleneck in serverless deployments.

There are several sources of network overhead in serverless applications. Function workflows often execute complex invocation DAGs (Fig. 1) that necessitate complex function-to-function communication [11, 20, 28, 40]. Functions also often retrieve the data they need to process from external storage services over the network, such as AWS S3 and DynamoDB [9, 14, 21, 22]. The networking stack can therefore be the bottleneck that determines the user-perceived end-to-end latency in serverless applications, and there is a pressing need to evaluate and optimize it [1, 6].

Serverless providers like AWS Lambda and popular open-source systems like vHive [37] deploy functions using Firecracker, a lightweight virtualization platform built on top of KVM. Such lightweight isolation platforms offer a balance between security and performance, allowing for rapid function instantiation and low memory footprint while maintaining a high degree of isolation between tenants. However, this approach introduces additional networking stack complexity, as each function instance typically requires its own virtualized network interface that then communicates with the host's networking stack, impacting latency and throughput in serverless environments.

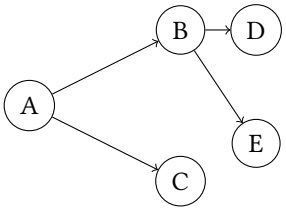


Figure 1: An example Serverless DAG

We comprehensively benchmark Firecracker's TUN/TAP-based networking stack, demonstrating that it is insufficient to support modern serverless deployments. We find that the latency and throughput achieved by each Firecracker microVM is significantly worse than that of applications running on bare-metal. For example, a single microVM can only use 5.2 Gbps of bandwidth and achieve only as low as 80 μsec p99 latency. The stack also suffers from poor scalability as the latency explodes in realistic deployments with a large number of Firecracker instances.

In this paper, we present a vision of a modern serverless networking stack that is *sidecar-based*. Our stack runs as a separate process in the host's userspace and is shared among all the VMs running in a host. We argue that a sidecar-based stack can provide performance, scalability, isolation, and the ability to enable direct function-to-function communication.

Finally, we identify a list of research challenges that need to be addressed to effectively implement our sidecar-based stack and overcome the current shortcomings in serverless networking. These challenges span multiple aspects of the networking stack design and implementation. Security and isolation are paramount, requiring innovative approaches to guarantee tenant separation while maintaining performance. Achieving zero-copy data transfer from the Network Interface Card (NICs) to the guest application presents another significant challenge, particularly in a multi-tenant environment where packet destinations are not immediately known. Scalability issues arise when supporting a large number of VMs on a single machine, necessitating efficient connection pooling and state management techniques. Additionally, close coordination between the networking stack and the function invocation scheduler is crucial in enabling performant function-to-function communication. We must develop efficient routing strategies for function-to-function communication, ensuring that traffic is directed to the correct host and target VM. Addressing these challenges will make the serverless stack more efficient in terms of resource utilization, significantly increasing throughput and reducing latency. This improved performance could unlock new use cases for serverless computing, enabling more complex and data-intensive applications to leverage the serverless paradigm.

2 Motivation

2.1 The Need for Fast Networking

Serverless DAGs. Many modern applications that make use of serverless platforms deploy the applications as a Directed Acyclic Graph (DAG), where the output of one function is sent as the input to the next function, and so on. When a function sends the output to another function, there is currently no targeted communication available, instead it is sent into a Pub-Sub queue from where the request is forwarded to the next function[2]. The function is continuously polling this queue to receive the requests, and if the function is warm then there is a trigger to reboot the function. Figure 1 depicts a DAG involving a chain of functions encompassing various design patterns characteristic of modern serverless architectures. Each request to the system involves at least 2 network calls, thus illustrating the importance of having a robust networking stack for serverless platforms. Existing work on profiling the serverless functions[10, 31, 32] showed a huge variability in the function execution times, and in some cases the network stack takes a non-trivial chunk of function execution.

External Services. Serverless functions frequently need to access external services for various operations, including data storage, database queries, API calls, and more. This interaction with external services can introduce significant bottlenecks in terms of throughput and latency, potentially undermining the scalability and performance benefits of serverless architectures. Recent research, such as PyWren [14], Apiary [22], and Nightcore [13] has highlighted the impact of the networking layer in serverless environments, particularly when dealing with object storage services like Amazon S3, and in scenarios involving the concurrent execution of numerous functions performing data-intensive operations. Apiary found that the communication latencies can be up to 80% of the overall latency, even in a containarized environment; we expect the overhead to be even more pronounced in a VM setup.

2.2 Firecracker Networking Stack

To explore the current limitations of the serverless networking stack, we take a deeper dive into the networking stack of Firecracker. To ensure security, a separate Firecracker process is spawned for every microVM. Firecracker employs the TUN/TAP[35] networking model to provide virtual NICs to its microVMs. TUN and TAP are Linux kernel devices that allow for the creation of virtual network interfaces. These interfaces can be used to send and receive network packets between user-space applications and the Linux Kernel. The guest kernel reads/writes into the TUN device, which is tunneled by the TAP device in the host network stack. Currently, Firecracker only supports TUN/TAP devices in a single queue,

where the packets processed in VM cannot be parallelized. This is done using the READ and WRITEV system calls[7].

However, TUN/TAP devices perform poorly as they are implemented using file I/O operations. TUN/TAP involves copying data buffers between user space and kernel space. This copying process adds latency and consumes CPU cycles. TUN/TAP's copying of data buffers leads to inefficient memory management and decreased throughput. To illustrate this problem, we performed a series of measurements with Firecracker microVMs and collected the latency and throughput measurements.

2.2.1 Experimental Setup. We evaluated the performance and scalability of existing network stacks using `iperf3` for throughput and `sockperf` for latency measurements. Throughput tests involved saturating the bandwidth and collecting data after reaching a steady state. Latency tests used Sockperf's ping-pong test, which measures server message processing capacity. Experiments were conducted on two Cloudlab [12] Utah (d6515) nodes, each with a 32-core AMD CPU and a dual-port Mellanox ConnectX-5 NIC providing 100 Gbps bandwidth. The nodes were connected via a dedicated 100 Gbps link to minimize external interference. We used Firectl to deploy multiple the Firecracker VMs. Each VM was allocated its own dedicated TUN/TAP device, which was then connected to a shared network bridge. This configuration allowed us to test the scalability of the network stack as the number of VMs increased while isolating each VM's network traffic.

We tested several configurations:

1. Baremetal-to-baremetal (BM-BM): We deployed the server application directly on one physical machine and the client application on the other. This setup serves as our baseline, representing the maximum achievable performance without virtualization overhead.
2. Baremetal-to-VM (BM-VM or VM-BM): In this setup, we ran the server application inside individual guest VMs on one physical machine while the client application ran directly on the other physical machine (bare metal). This configuration helps quantify the performance impact of virtualization on the server side, as well as the efficiency of network communication between virtualized and non-virtualized environments.
3. VM-to-VM (VM-VM): We deployed up to 126 Firecracker VMs on each of the two physical machines, establishing a one-to-one communication pattern between VMs across the machines. This configuration represents a highly virtualized environment, allowing us to assess the scalability and performance of inter-VM communication in a dense deployment scenario.

For each configuration, we measured throughput, latency, and CPU usage to comprehensively evaluate the network stack's performance under different virtualization scenarios.

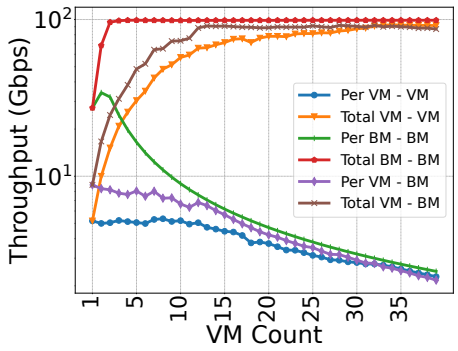
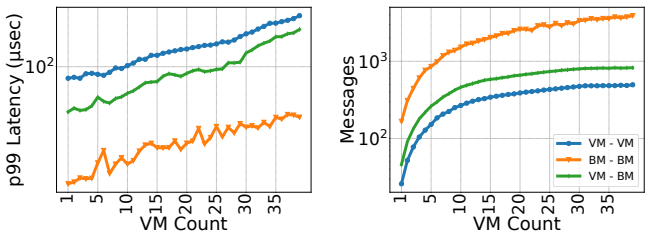


Figure 2: Throughput (Gbps) for experimental setup comparing the performance for BM-BM, and VM-VM



(a) Latencies (μsec) per VM **(b) Total Messages received**

Figure 3: Experimental setup comparing the latency and messages received for BM-BM, VM-VM and VM-BM scenarios

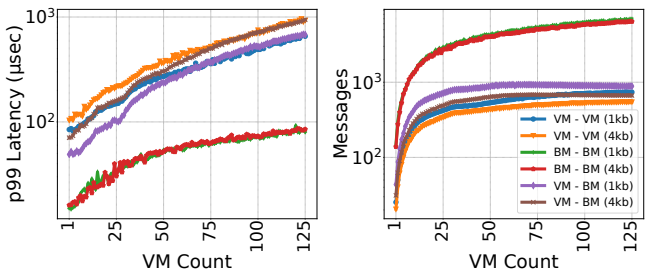
2.2.2 Evaluation. Figure 2 presents a comprehensive analysis of throughput performance across all experimental scenarios, delineating both aggregate bandwidth utilization and per-host bandwidth efficiency. Our findings reveal notable disparities between virtualized and non-virtualized environments. The virtualized configurations (VM-VM and BM-VM) exhibited markedly diminished performance, even in single-VM deployments, suggesting significant virtualization overhead. In the context of single VM deployment within the Firecracker host, we observed bandwidth utilization of 5.19% and 8.81% for VM-VM and VM-BM configurations, respectively. This represents a substantial underutilization of available network resources. In contrast, a single non-virtualized application can utilize 27.3% of the network bandwidth. Furthermore, based on these observations, theoretically the bandwidth saturation should occur at approximately 20 VMs for VM-VM and 13 VMs for VM-BM scenarios. However, our empirical data diverged from these projections. Peak bandwidth utilization was not achieved until the deployment of 35 VMs, a finding that indicates non-linear scaling behavior in the system. This discrepancy between projections and observed results underscores the complexity of network performance in virtualized

environments and highlights the potential for unforeseen interactions in multi-VM scenarios.

Firecracker VMs significantly underutilize network bandwidth, with single VMs reaching only 5-9% of potential throughput, and exhibit non-linear scaling, peaking at higher-than-predicted VM counts.

Figure 3 presents an analysis of latency (Fig 3a) and message throughput (Fig 3b) across various experimental configurations. Our study utilizes the sockperf workload generator, operating in a closed-loop manner, which ensures subsequent packet transmission only upon receipt of responses to all previously sent packets. This methodology provides a controlled environment for our observations, enabling precise measurement of system performance under varying loads. Our analysis reveals a substantial disparity in base latency between baremetal-to-baremetal (BM-BM) and virtualized deployments. In single VM scenarios, the latencies in virtualized configurations were markedly higher, with VM-VM and VM-BM configurations exhibiting increases of 600% and 350% respectively, compared to the baremetal configuration. Notably, this significant latency differential persisted across increasing VM counts, indicating a consistent virtualization overhead. Concurrently, we observed a plateau in message processing capacity at approximately 10 VMs. Beyond this threshold, increasing the number of VMs led to a pronounced elevation in latency, while the total number of messages sent and received remained relatively constant. This juxtaposition of rising latency and stagnant message throughput strongly suggests the emergence of substantial queuing delays within the Firecracker environment.

Firecracker virtualization introduces substantial latency overhead (350-600% increase) and limits message throughput scalability beyond 10 VMs, revealing significant challenges in optimizing network performance for serverless environments.



(a) Latencies (μsec) per VM **(b) Total Messages received**

Figure 4: Comparing the latency and messages received with varying message size

Figure 4 illustrates the impact of virtualization on network performance across varying Ethernet frame sizes. Our analysis reveals a contrast between bare-metal and virtualized configurations in their handling of increased message sizes that necessitate additional Ethernet frames. In the bare-metal configuration, we observe negligible variations in latency and message throughput when the message size is marginally increased to require an additional Ethernet frame. This consistency underscores the efficiency of native network processing in non-virtualized environments. Conversely, virtual machine (VM) configurations exhibit a pronounced difference in both latency and message throughput under similar conditions. This disparity is clearly demonstrated by the (blue, orange) and (green, red) line pairs in Figures 4a and 4b, respectively. The marked increase in latency and decrease in message throughput for VMs when processing larger messages points to a significant overhead associated with virtualized network processing. We attribute this additional overhead in VM environments primarily to the TUN/TAP mechanism employed for network virtualization and the lack of optimizations like Generic Receive Offload (GRO) and TCP Segmentation Offload (TSO).

VM configurations show significant performance degradation when processing messages requiring additional Ethernet frames, unlike bare-metal setups, highlighting the overhead of processing packets in virtualized environments.

2.3 Possible Alternatives

In the previous section, we identified and characterized the inefficiencies present within the networking stack of Firecracker. Our analysis revealed that these inefficiencies have a detrimental impact on both the throughput and latency of even a single VM. Furthermore, we observed that the performance degradation compounds as the number of deployed VMs increases. The overhead introduced by the suboptimal network stack has significant implications for the overall performance and scalability of the serverless computing environment built upon Firecracker. We now explain how other existing approaches used in different settings are also insufficient, and a clean slate redesign is necessary.

2.3.1 Single Root IO Virtualization. Single Root IO Virtualization (SR-IOV)[3] is a technology that enables multiple virtual instances of the same physical networking device with separate resources. These virtual functions(VFs) can be provisioned separately. Each VF can be seen as an additional device and is commonly used in conjunction with an SR-IOV-enabled hypervisor to provide virtual machines direct hardware access to network resources hence offering native performance.

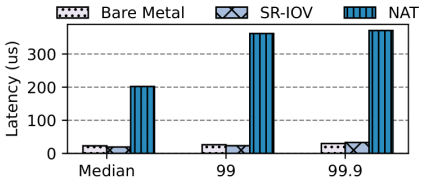


Figure 5: SR-IOV achieves near bare-metal performance, in contrast to traditional network virtualization (NAT).

Evaluation. Figure 5 illustrates that SR-IOV approaches bare metal performance, in contrast to the default setup in virtual machines that use a virtual network switch operating in Network Address Translation(NAT) mode[8]. NAT is the generic networking solution for VMs that, similar to Firecracker's TUN/TAP approach, involves both the guest and the host kernel's networking stack. The experiment runs on two nodes, each with a 16-core Intel Xeon Silver 4314 at 2.40 GHz, 128 GB RAM, and a Dual-port Mellanox ConnectX-6 LX 25Gb Ethernet adapter. One node operated as the sockperf[5] server, and the other as a sockperf client, sending 64-byte messages at peak throughput using TCP.

SR-IOV - Serverless Incompatibility. Based on the insights from the experiment, the idea of implementing SR-IOV support for microVMs emerges. While initially promising, SR-IOV faces scalability issues that are not well-suited to the serverless paradigm. Cloud providers aim to maximize the number of functions on a single host[32], necessitating a proportional number of virtual functions. For example, Alibaba hosts over 2500 serverless functions, i.e., microVMs, per machine[25]. However, even the most advanced NICs currently support only up to 127 virtual functions per port [4]. Freeflow[19] and Falcon[23] identify the insufficient number of virtual functions as a primary limitation to deploy them at large scale for containerized systems. HD-IOV[42] also pointed out that SR-IOV falls short due to limited management capabilities. For example, the attached VM must be destroyed to reuse a VF in another VM instance, making it impossible to cache the instance – a common solution to mitigate the cold start problem. Additionally, the allocation and deallocation of Virtual Functions (VFs) to microVMs could conflict with the Service Level Agreement (SLA) guarantees of serverless workloads, as they necessitate configuration changes that typically require microVM reboots.

2.3.2 Other Networking Solutions. Currently, organizations solve the network bottleneck issues by using tools such as Consul, where a few TUN/TAP devices are created and reused for the VMs. However, these solutions do not target the fundamental problem of the overheads introduced by the TUN/TAP device. Some solutions are targeted towards OpenFaas[36, 41], a higher-level framework for deploying functions in Kubernetes using containers. Nightcore [13], a state-of-the-art serverless platform designed for microsecond scale applications, utilizes a shared memory model for fast intra-server communication. It makes use of a non-VM based execution model, similar to WASM-based solutions such as Faasm [33]. These solutions are targeted for container or language-based runtimes and make assumptions, such as having direct access to the host interface, that make them unusable for VM-based deployments.

While existing solutions provide stopgap measures to mitigate some of the drawbacks of current networking approaches in serverless environments, there is a pressing need for a fundamental reimagining of the network stack. A ground-up redesign has the potential to yield significant improvements across multiple dimensions of serverless performance. By developing a more efficient and tailored networking stack, we can substantially reduce overall request latency, increase throughput, and enable direct function-to-function communication. This holistic approach to network optimization could dramatically enhance the capabilities of serverless platforms, enabling more complex workflows and improving resource utilization. Such advancements would not only address current limitations but also pave the way for novel serverless applications that were previously constrained by networking inefficiencies.

3 Proposed Solution

The increasing popularity of serverless computing and the limitations of current networking solutions in this domain necessitate a rethinking of the network stack for serverless environments. To address these challenges and unlock the full potential of serverless platforms, we consider the following key requirements for a networking stack:

Low overhead and high performance. The networking stack should aim to minimize latency and maximize throughput, addressing the performance limitations observed with current solutions like TUN/TAP devices. It should strive to approach bare-metal performance levels while operating in a virtualized environment.

Scalability for high-density deployments. The design should support efficient operation with a large number of concurrent functions/microVMs on a single host. It needs to overcome the scalability issues seen with approaches like SR-IOV, which are limited in the number of virtual functions they can support.

Fast startup and teardown. Given the short-lived nature of serverless functions, the network stack should be optimized for rapid initialization and cleanup. It should not introduce

significant overhead to function cold start times or require operations that conflict with serverless SLAs.

Support for direct function-to-function communication. The design should facilitate efficient communication between functions in a serverless DAG without relying on intermediary queues or excessive network hops. This would help reduce latency in multi-function applications and support more complex serverless workflows.

Strong security and isolation. The networking stack must maintain the high level of security and isolation provided by current serverless platforms. It should prevent unauthorized access between functions and ensure that the improved performance and communication capabilities do not compromise the security guarantees expected in multi-tenant serverless environments

3.1 System

To address the challenges outlined in Section 2, we propose a sidecar networking stack for serverless platforms, as illustrated in Figure 6. This approach builds upon and extends concepts introduced in recent literature [18, 29, 30, 34], offering a more comprehensive and adaptable solution. The proposed sidecar model provides greater flexibility to platform providers, enabling the integration of advanced networking paradigms such as user-space TCP or DPDK-based kernel bypass transport layers. These solutions aim to significantly reduce latency and increase throughput, thus addressing the key performance bottlenecks identified in Section 2.2. Our design establishes a uniform interface between microVMs and the sidecar, ensuring consistency and interoperability regardless of the sidecar's implementation logic. This abstraction layer decouples the networking logic from function invocation and execution, allowing for modular upgrades and optimizations without disrupting the core infrastructure.

To implement this vision, we will develop a virtual network device for microVMs, leveraging shared memory communication for ultra-low latency interactions. The sidecar network stack handles inbound and outbound connections, zero-copy processing of packets, and routing requests to the correct function. For functions running on the same machine, we employ a shared memory paradigm to locally forward packets to the target VM, further optimizing performance. This centralized approach not only simplifies the overall network architecture but also opens up new possibilities for inter-function communication, a critical aspect for implementing serverless Directed Acyclic Graphs (DAGs). Moreover, it enables operators to implement advanced function placement, routing, and network resource management strategies.

Our design prioritizes low overhead and high performance, aiming to minimize latency and maximize throughput while approaching bare-metal performance levels in a virtualized environment. It addresses the scalability challenges associated with high-density deployments, supporting efficient operation with a large number of concurrent functions/microVMs on a single host. This overcomes the limitations seen with approaches like SR-IOV, which are constrained in the number of virtual functions they can support. Recognizing the ephemeral nature of serverless functions, our network stack will be optimized for rapid initialization and cleanup, minimizing the impact on function cold start times and avoiding operations that could conflict with serverless Service Level Agreements (SLAs). Crucially, our approach maintains the high level of security and isolation provided by current serverless platforms, preventing unauthorized access between functions and ensuring that the improved performance and communication capabilities do not compromise the security guarantees expected in multi-tenant serverless environments. This comprehensive design addresses the key challenges in serverless networking while opening new avenues for performance optimization and application complexity in serverless computing.

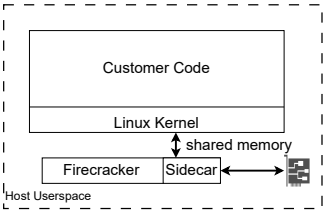


Figure 6: Sidecar Network Functionality in User Space Interfacing with MicroVMs via Shared Memory.

4 Research Directions

4.1 Security and Isolation

Serverless platforms need to guarantee isolation between multiple tenant VMs, which also extends to the networking stack. Since the VMs would be sharing a networking stack, we must ensure isolation for individual tenants and alleviate performance interference. This is critical not only for security but also for maintaining consistent performance across different workloads. One of the ways we could potentially achieve isolation is to have a separate set of threads per VM, similar to the approach used in Snap[29]. However, this would require specialized scheduling policies to manage the increased number of threads efficiently. The challenge lies in balancing the granularity of isolation with the overhead of managing numerous threads. Furthermore, we need to consider the implications of shared network resources, such as network interface cards (NICs) and their queue. Since there is no isolation guarantee provided by these commodity NICs, they're prone to side-channel attacks, leading to data leaks [44]. We must

consider the trade-offs between isolation and performance when making these decisions. While stronger isolation generally improves security, it may introduce additional latency and reduce overall system throughput. Thus, adaptive isolation techniques that can dynamically adjust the level of isolation based on workload characteristics and security requirements could provide a flexible solution to this challenge.

4.2 Zero-copy

In the current network stack, the use of TUN/TAP devices results in multiple copies being made as data travels from the Network Interface Card (NIC) to the guest VM. This process introduces unnecessary overhead, impacting both latency and CPU utilization. Ideally, in our sidecar networking stack, we aim to achieve zero-copy data transfer all the way from the NIC to the guest application inside the VM. This optimization would significantly reduce memory bandwidth usage and CPU overhead, leading to improved performance and efficiency. However, implementing zero-copy in a sidecar architecture is non-trivial. The primary challenge stems from the fact that the sidecar does not know the target VM before parsing the packet, as it receives packets into a generic message buffer pool shared between multiple VMs. This shared buffer approach, while flexible, complicates the direct mapping of packet data to VM-specific memory spaces. One promising approach is to dynamically change the page table mappings for packet data[17, 24, 39]. This method involves initially placing packets in a shared buffer pool, and then remapping the physical pages to appropriate virtual pages in target VM's address space once the destination is determined. This eliminates the need for copying data, however requires efficient page table manipulation to avoid performance penalties. We could also implement a two-stage buffer system where packets are initially received into a shared, high-performance buffer. Once the target VM is identified, we use a zero-copy mechanism like vhost-user to transfer the data directly into the VM's memory[38]. This approach maintains flexibility while minimizing data movement. Each of these solutions come with their own set of tradeoffs in terms of performance, complexity and hardware requirements. The optimal solution may involve a combination of these techniques, dynamically applied based on workload characteristics and system conditions.

4.3 Scalability

The sidecar networking stack needs to support a large number of VMs on the same machine, which introduces significant challenges in terms of scalability and efficiency. One of the primary concerns is the potential overhead introduced by VMs polling the network stack to receive packets. This polling mechanism, if not optimized, can lead to excessive CPU utilization and increased latency. Moreover, maintaining state for numerous connections across multiple VMs can result in cache unfriendliness, further impacting performance. To address these challenges, we need to perform sophisticated connection pooling that leverages a multiplexed notification system along with direct, efficient data transfer channels. A shared notification socket, that is multiplexed across various idle VMs, can be used for notifying the VMs about incoming packets. This multiplexing can be achieved by using techniques similar to `io_uring` in Linux. Upon receiving a notification, instead of establishing a new connection, we can make use of a dynamic connection pool. Pool contains a pre-established number of connections that is assigned to VMs, based on need. The pool size can be adjusted based on the current load and system resources. To maximize the efficiency of these connections, they will not immediately be torn down after use, instead they will be returned to the pool with a time-to-live (TTL) value, thus reducing the overhead of frequent connection establishment and teardown. This could be combined with a priority based-assignment mechanism, where the latency-sensitive VMs get higher priority based on SLA agreements. To address cache unfriendliness, we can optimize the state management by using compact data structures and cache-conscious algorithms. This includes techniques like cache line alignment and data structure partitioning to improve locality.

This approach would require the networking stack to make dynamic, intelligent decisions about resource utilization. It must balance the trade-offs between maintaining a large connection pool (consumes resources; decreases latency) and frequent connection setup/teardown (saves resources; increases latency). By designing and employing predictive algorithms, the system can perform optimal decisions during run-time.

4.4 Coordination with Function Scheduler

The networking stack in a serverless environment needs to coordinate closely with the function invocation scheduler for effectively managing data transfers to and from functions. This coordination is crucial for optimizing resource utilization, minimizing latency, and ensuring efficient communication between functions. There are several approaches to achieve this coordination, each with its own set of trade-offs and considerations. One approach is for the networking stack to synchronously communicate with the scheduler, which maintains knowledge of all function invocation locations[36]. This method ensures up-to-date information but may introduce latency in packet routing decisions. Alternatively, the networking stack can function like a Software-Defined Networking (SDN) router, receiving periodic updates from the scheduler, similar to the approach used in DirectFaaS[41]. This method reduces communication overhead but may lead

to temporary inconsistencies. A more complicated approach involves a bi-directional information sharing between the networking stack and function scheduler. The scheduler can use the knowledge from networking stack's open connection pools, for making smarter scheduling decisions, and load balancing. The scheduler can also make use of the network topology information to co-locate the functions that frequently communicate with each other on the same host to minimize data network overhead for function-to-function communication. Offloading the control plane to a SmartSwitch with P4 programming capabilities can provide a global view of the system, monitor all scheduling assignments, and dynamically re-route packets. This approach can potentially reduce the coordination burden on both the network stack and the scheduler. By carefully designing the coordination between the network stack and function scheduler, we can create a more efficient, responsive, and scalable serverless platform that can adapt to diverse and dynamic workloads.

5 Conclusion

As users increasingly deploy complex, multi-function applications, the need for efficient inter-function communication has become paramount. Our research addresses this gap by focusing on the design of a networking stack specifically tailored for serverless systems. Through extensive benchmarking, we have demonstrated the limitations of current networking stacks, which often underutilize available bandwidth and introduce high latency. These issues, combined with scalability challenges in potential alternate solutions, make them inadequate for the demands of modern serverless applications. To address these shortcomings, we have proposed a sidecar-based userspace networking stack that balances the key requirements of a serverless environment: strong isolation, multi-VM scalability, low overhead, fast startup, and efficient function-to-function communication. Our design aims to overcome the bandwidth underutilization and latency issues observed in current systems, while also addressing scalability concerns. Key challenges in implementing this design include achieving zero-copy data transfer, ensuring security and isolation in a multi-tenant environment, and coordinating effectively with the function scheduler to optimize resource utilization and minimize latency. By addressing these challenges, we can significantly reduce packet processing overhead and improve overall system throughput. This approach has the potential to enhance network performance while making more efficient use of available CPU, memory and network resources in serverless environments while also paving way for future use cases.

Acknowledgments

We'd like to thank our shepherd, George Neville-Neill, anonymous reviewers of SoCC'24, SESAME'24 for their invaluable feedback and conversations that have greatly improved this work. We thank Amanda Raybuck, Deepti Raghavan and Guanzhou Hu for reading early versions of the draft and offering feedback. We thank Cloudlab[12] for hardware and technical support.

References

[1] [n. d.]. Adding vhost-net support to firecracker. https://github.com/firecracker-microvm/firecracker/issues/3707#issuecomment-1893454882

[2] [n. d.]. Azure Web PubSub trigger and bindings for Azure Functions. https://learn.microsoft.com/en-us/azure/azure-web-pubsub/reference-functions-bindings?tabs=csharp

[3] [n. d.]. Introduction to Single Root I/O Virtualization (SR-IOV). https://learn.microsoft.com/en-us/windows-hardware/drivers/network/single-root-i-o-virtualization--sr-iov-

[4] [n. d.]. NVIDIA Mellanox Virtual Functions. https://docs.nvidia.com/networking/display/mlnxofedv590560125/single+root+io+virtualization+(sr-iov)

[5] [n. d.]. sockperf: A Network Benchmarking Tool. https://github.com/Mellanox/sockperf

[6] [n. d.]. Virtual networking 101: bridging the gap to understanding TAP. https://blog.cloudflare.com/virtual-networking-101-understanding-tap

[7] [n. d.]. writev system call in firecracker. https://github.com/firecracker-microvm/firecracker/blob/b56d201186a7928e0e303c66fc475a89520d6d97/resources/seccomp/x86_64-unknown-linux-musl.json#L33

[8] Updated 2024. Virtual Networking. https://wiki.libvirt.org/VirtualNetworking.html

[9] Daniel Barcelona-Pons, Pierre Sutra, Marc Sánchez-Artigas, Gerard París, and Pedro García-López. 2022. Stateful Serverless Computing with Crucial. ACM Trans. Softw. Eng. Methodol. 31, 3, Article 39 (mar 2022), 38 pages. https://doi.org/10.1145/3490386

[10] André Bauer, Haochen Pan, Ryan Chard, Yadu Babuji, Josh Bryan, Devesh Tiwari, Ian Foster, and Kyle Chard. 2024. The globus compute dataset: An open function-as-a-service dataset from the edge to the cloud. Future Generation Computer Systems 153 (2024), 558–574. https://doi.org/10.1016/j.future.2023.12.007

[11] Marcin Copik, Roman Böhringer, Alexandru Calotoiu, and Torsten Hoefler. 2023. FMI: Fast and Cheap Message Passing for Serverless Functions. In Proceedings of the 37th ACM International Conference on Supercomputing (Orlando, FL, USA) (ICS '23). Association for Computing Machinery, New York, NY, USA, 373–385. https://doi.org/10.1145/3577193.3593718

[12] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The Design and Operation of CloudLab. In 2019 USENIX Annual Technical Conference (USENIX ATC 19). USENIX Association, Renton, WA, 1–14. https://www.usenix.org/conference/atc19/presentation/duplyakin

[13] Zhipeng Jia and Emmett Witchel. 2021. Nightcore: efficient and scalable serverless computing for latency-sensitive, interactive microservices. In Proceedings of the 26th ACM International

Conference on Architectural Support for Programming Languages and Operating Systems (Virtual, USA) (ASPLOS '21). Association for Computing Machinery, New York, NY, USA, 152–166. https://doi.org/10.1145/3445814.3446701

[14] Eric Jonas, Qifan Pu, Shivaram Venkataraman, Ion Stoica, and Benjamin Recht. 2017. Occupy the cloud: distributed computing for the 99%. In Proceedings of the 2017 Symposium on Cloud Computing (Santa Clara, California) (SoCC '17). Association for Computing Machinery, New York, NY, USA, 445–451. https://doi.org/10.1145/3127479.3128601

[15] Kostis Kaffes, Neeraja J. Yadwadkar, and Christos Kozyrakis. 2019. Centralized Core-granular Scheduling for Serverless Functions. In Proceedings of the ACM Symposium on Cloud Computing (Santa Cruz, CA, USA) (SoCC '19). Association for Computing Machinery, New York, NY, USA, 158–164. https://doi.org/10.1145/3357223.3362709

[16] Kostis Kaffes, Neeraja J. Yadwadkar, and Christos Kozyrakis. 2022. Hermod: principled and practical scheduling for serverless functions. In Proceedings of the 13th Symposium on Cloud Computing (San Francisco, California) (SoCC '22). Association for Computing Machinery, New York, NY, USA, 289–305. https://doi.org/10.1145/3542929.3563468

[17] Dong Hyun Kang, Gihwan Oh, Dongki Kim, In Hwan Doh, Changwoo Min, Sang-Won Lee, and Young Ik Eom. 2018. When Address Remapping Techniques Meet Consistency Guarantee Mechanisms. In 10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 18). USENIX Association, Boston, MA. https://www.usenix.org/conference/hotstorage18/presentation/kang

[18] Antoine Kaufmann, Tim Stamler, Simon Peter, Naveen Kr. Sharma, Arvind Krishnamurthy, and Thomas Anderson. 2019. TAS: TCP Acceleration as an OS Service. In Proceedings of the Fourteenth EuroSys Conference 2019 (Dresden, Germany) (EuroSys '19). Association for Computing Machinery, New York, NY, USA, Article 24, 16 pages. https://doi.org/10.1145/3302424.3303985

[19] Daehyeok Kim, Tianlong Yu, Hongqiang Harry Liu, Yibo Zhu, Jitu Padhye, Shachar Raindel, Chuanxiong Guo, Vyas Sekar, and Srinivasan Seshan. 2019. FreeFlow: Software-based Virtual RDMA Networking for Containerized Clouds. In 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19). USENIX Association, Boston, MA, 113–126. https://www.usenix.org/conference/nsdi19/presentation/kim

[20] Ana Klimovic, Yawen Wang, Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, and Christos Kozyrakis. 2018. Pocket: Elastic Ephemeral Storage for Serverless Analytics. In 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18). USENIX Association, Carlsbad, CA, 427–444. https://www.usenix.org/conference/osdi18/presentation/klimovic

[21] Swaroop Kotni, Ajay Nayak, Vinod Ganapathy, and Arkaprava Basu. 2021. Faastlane: Accelerating Function-as-a-Service Workflows. In 2021 USENIX Annual Technical Conference (USENIX ATC 21). USENIX Association, 805–820. https://www.usenix.org/conference/atc21/presentation/kotni

[22] Peter Kraft, Qian Li, Kostis Kaffes, Athinagoras Skiadopoulos, Deeptaanshu Kumar, Danny Cho, Jason Li, Robert Redmond, Nathan Weckwerth, Brian Xia, Peter Bailis, Michael Cafarella, Goetz Graefe, Jeremy Kepner, Christos Kozyrakis, Michael Stonebraker, Lalith Suresh, Xiangyao Yu, and Matei Zaharia. 2023. Apiary: A DBMS-Integrated Transactional Function-as-a-Service Framework. arXiv:2208.13068 [cs.DB] https://arxiv.org/abs/2208.13068

[23] Jiaxin Lei, Manish Munikar, Kun Suo, Hui Lu, and Jia Rao. 2021. Parallelizing packet processing in container overlay networks. In Proceedings of the Sixteenth European Conference on Computer Systems (Online Event, United Kingdom) (EuroSys '21). Association for Computing Machinery, New York, NY, USA, 261–276. https://doi.org/10.1145/3447786.3456241

[24] Baptiste Lepers and Willy Zwaenepoel. 2023. Johnny Cache: the End of DRAM Cache Conflicts (in Tiered Main Memory Systems). In 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23). USENIX Association, Boston, MA, 519–534. https://www.usenix.org/conference/osdi23/presentation/lepers

[25] Zijun Li, Jiagan Cheng, Quan Chen, Eryu Guan, Zizheng Bian, Yi Tao, Bin Zha, Qiang Wang, Weidong Han, and Minyi Guo. 2022. RunD: A Lightweight Secure Container Runtime for High-density Deployment and High-concurrency Startup in Serverless Computing. In 2022 USENIX Annual Technical Conference (USENIX ATC 22). USENIX Association, Carlsbad, CA, 53–68. https://www.usenix.org/conference/atc22/presentation/li-zijun-rund

[26] Zijun Li, Linsong Guo, Quan Chen, Jiagan Cheng, Chuhao Xu, Deze Zeng, Zhuo Song, Tao Ma, Yong Yang, Chao Li, and Minyi Guo. 2022. Help Rather Than Recycle: Alleviating Cold Startup in Serverless Computing Through Inter-Function Container Sharing. In 2022 USENIX Annual Technical Conference (USENIX ATC 22). USENIX Association, Carlsbad, CA, 69–84. https://www.usenix.org/conference/atc22/presentation/li-zijun-help

[27] Qingyuan Liu, Dong Du, Yubin Xia, Ping Zhang, and Haibo Chen. 2023. The Gap Between Serverless Research and Real-world Systems. In Proceedings of the 2023 ACM Symposium on Cloud Computing (<conf-loc>, <city>Santa Cruz</city>, <state>CA</state>, <country>USA</country>, </conf-loc>) (SoCC '23). Association for Computing Machinery, New York, NY, USA, 475–485. https://doi.org/10.1145/3620678.3624785

[28] Ashraf Mahgoub, Karthick Shankar, Subrata Mitra, Ana Klimovic, Somali Chaterji, and Saurabh Bagchi. 2021. SONIC: Application-aware Data Passing for Chained Serverless Applications. In 2021 USENIX Annual Technical Conference (USENIX ATC 21). USENIX Association, 285–301. https://www.usenix.org/conference/atc21/presentation/mahgoub

[29] Michael Marty, Marc de Kruijf, Jacob Adriaens, Christopher Alfeld, Sean Bauer, Carlo Contavalli, Michael Dalton, Nandita Dukkipati, William C Evans, Steve Gribble, et al. 2019. Snap: a microkernel approach to host networking. In Symposium on Operating Systems Principles (SOSP). ACM.

[30] Zhixiong Niu, Hong Xu, Peng Cheng, Qiang Su, Yongqiang Xiong, Tao Wang, Dongsu Han, and Keith Winstein. 2020. NetKernel: Making Network Stack Part of the Virtualized Infrastructure. In 2020 USENIX Annual Technical Conference (USENIX ATC 20). USENIX Association, 143–157. https://www.usenix.org/conference/atc20/presentation/niu

[31] Alireza Sahraei, Soteris Demetriou, Amirali Sobhgol, Haoran Zhang, Abhigna Nagaraja, Neeraj Pathak, Girish Joshi, Carla Souza, Bo Huang, Wyatt Cook, Andrii Golovei, Pradeep Venkat, Andrew Mcfague, Dimitrios Skarlatos, Vipul Patel, Ravinder Thind, Ernesto Gonzalez, Yun Jin, and Chunqiang Tang. 2023. XFaaS: Hyperscale and Low Cost Serverless Functions at Meta. In Proceedings of the 29th Symposium on Operating Systems Principles (Koblenz, Germany) (SOSP '23). Association for Computing Machinery, New York, NY, USA, 231–246. https://doi.org/10.1145/3600006.3613155

[32] Mohammad Shahrad, Rodrigo Fonseca, Inigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. 2020. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider. In 2020 USENIX Annual Technical Conference (USENIX ATC 20). USENIX Association, 205–218. https://www.usenix.org/conference/atc20/presentation/shahrad

[33] Simon Shillaker and Peter Pietzuch. 2020. Faasm: Lightweight Isolation for Efficient Stateful Serverless Computing. In 2020 USENIX Annual Technical Conference (USENIX ATC 20). USENIX Association, 419–433. https://www.usenix.org/conference/atc20/presentation/shillaker

[34] Matheus Stolet, Liam Arzola, Simon Peter, and Antoine Kaufmann. 2023. Virtuoso: High Resource Utilization and μs-scale Performance Isolation in a Shared Virtual Machine TCP Network Stack. arXiv:2309.14016 [cs.NI]

[35] The Linux Kernel Development Team. Updated 2023. TUN/TAP: Universal TUN/TAP Device Driver. https://docs.kernel.org/networking/tuntap.html

[36] Dmitrii Ustiugov, Shyam Jesalpura, Mert Bora Alper, Michal Baczun, Rustem Feyzkhanov, Edouard Bugnion, Boris Grot, and Marios Kogias. 2023. Expedited Data Transfers for Serverless Clouds. arXiv:2309.14821 [cs.DC]

[37] Dmitrii Ustiugov, Plamen Petrov, Marios Kogias, Edouard Bugnion, and Boris Grot. 2021. Benchmarking, analysis, and optimization of serverless function snapshots. In Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (Virtual, USA) (ASPLOS '21). Association for Computing Machinery, New York, NY, USA, 559–572. https://doi.org/10.1145/3445814.3446714

[38] Dongyang Wang and Bei Hua. 2019. ZCopy-Vhost: Replacing Data Copy With Page Remapping in Virtual Packet I/O. IEEE Access 7 (2019), 51047–51057. https://doi.org/10.1109/ACCESS.2019.2911905

[39] Chao Yang, Yunfei Guo, and Hongchao Hu. 2019. MemWander: Memory Dynamic Remapping via Hypervisor Against Cache-Based Side-Channel Attacks. IEEE Access 7 (2019), 2179–2199. https://doi.org/10.1109/ACCESS.2018.2886372

[40] Minchen Yu, Tingjia Cao, Wei Wang, and Ruichuan Chen. 2023. Following the Data, Not the Function: Rethinking Function Orchestration in Serverless Computing. In 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23). USENIX Association, Boston, MA, 1489–1504. https://www.usenix.org/conference/nsdi23/presentation/yu

[41] Qingyang Zeng, Kaiyu Hou, Xue Leng, and Yan Chen. 2024. DirectFaaS: A Clean-Slate Network Architecture for Efficient Serverless Chain Communications. In Proceedings of the ACM on Web Conference 2024 (Singapore, Singapore) (WWW '24). Association for Computing Machinery, New York, NY, USA, 2759–2767. https://doi.org/10.1145/3589334.3645333

[42] Zongpu Zhang, Jiangtao Chen, Banghao Ying, Yahui Cao, Lingyu Liu, Jian Li, Xin Zeng, Junyuan Wang, Weigang Li, and Haibing Guan. 2024. HD-IOV: SW-HW Co-designed I/O Virtualization with Scalability and Flexibility for Hyper-Density Cloud. In Proceedings of the Nineteenth European Conference on Computer Systems (Athens, Greece) (EuroSys '24). Association for Computing Machinery, New York, NY, USA, 834–850. https://doi.org/10.1145/3627703.3629557

[43] Shixuan Zhao, Pinshen Xu, Guoxing Chen, Mengya Zhang, Yinqian Zhang, and Zhiqiang Lin. 2023. Reusable Enclaves for Confidential Serverless Computing. In 32nd USENIX Security Symposium (USENIX Security 23). USENIX Association, Anaheim, CA, 4015–4032. https://www.usenix.org/conference/usenixsecurity23/presentation/zhao-shixuan

[44] Yang Zhou, Mark Wilkening, James Mickens, and Minlan Yu. 2024. SmartNIC Security Isolation in the Cloud with S-NIC. In Proceedings of the Nineteenth European Conference on Computer Systems (Athens, Greece) (EuroSys '24). Association for Computing Machinery, New York, NY, USA, 851–869. https://doi.org/10.1145/3627703.3650071