hyDNS: Acceleration of DNS Through Kernel Space Resolution

Joshua Bardinelli
University of Connecticut
Storrs, CT, USA

Yifan Zhang
University of Connecticut
Storrs, CT, USA

Jianchang Su
University of Connecticut
Storrs, CT, USA

Linpu Huang
University of Connecticut
Storrs, CT, USA

Aidan Parilla
University of Connecticut
Storrs, CT, USA

Rachel Jarvi
University of Connecticut
Storrs, CT, USA

Sameer G. Kulkarni*
IIT Gandhinagar
Gandhinagar, Gujarat, India
sameergk@iitgn.ac.in

Wei Zhang†
University of Connecticut
Storrs, CT, USA
wei.13.zhang@uconn.edu

Abstract

The Domain Name System (DNS) is a core component of Internet infrastructure, mapping domain names to IP addresses. The recursive resolver plays a critical role in this process, requiring high performance due to multiple request-response exchanges. However, its performance is hindered by costly message copying, user-kernel space transitions, and kernel stack traversal. Kernel bypass techniques can mitigate these issues but often result in resource waste or deployment challenges.

To overcome these limitations, We present hyDNS, a hybrid recursive resolver that combines eBPF offloading in the kernel with a user-space resolver. The DNS kernel cache allows most requests to be served before reaching the kernel network stack. To manage limited DMA memory, excess requests are passed to user space once a threshold is reached, enabling the system to handle high query loads. hyDNS uses programmable NICs to create a scalable kernel cache, implementing a lockless per-core eBPF hash map. Filters on the NIC direct requests to each core. Preliminary results show significant performance improvements with eBPF offloading, achieving up to 4.4× the throughput and a 65% reduction in latency compared to user space implementations.

CCS Concepts

• **Networks** → **Naming and addressing**; **Programmable networks**; **In-network processing**; • **Computer systems organization** → *Heterogeneous (hybrid) systems.*

Keywords

Domain Name System (DNS), eBPF (extended Berkeley Packet Filter), XDP (eXpress Data Path), In-Kernel Cache

*†Corresponding author.

eBPF '24, August 4–8, 2024, Sydney, NSW, Australia

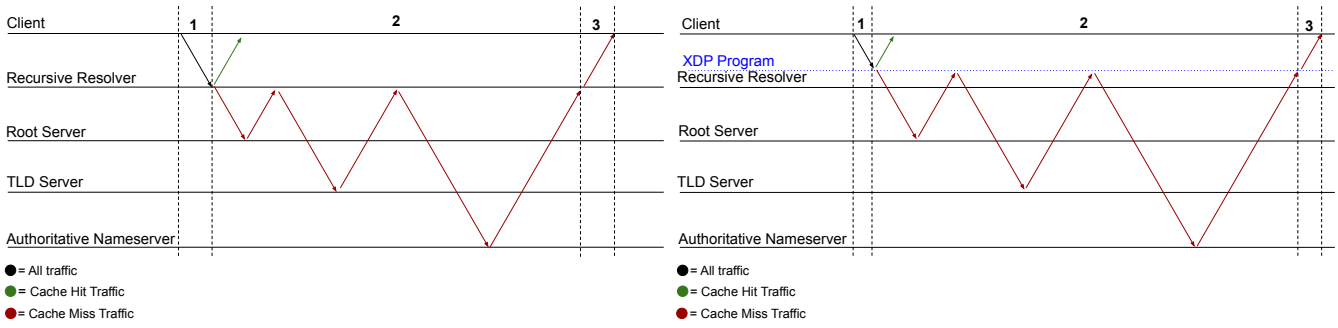
ACM ISBN 979-8-4007-0712-4/24/08
https://doi.org/10.1145/3672197.3673439

ACM Reference Format:
Joshua Bardinelli, Yifan Zhang, Jianchang Su, Linpu Huang, Aidan Parilla, Rachel Jarvi, Sameer G. Kulkarni*, and Wei Zhang†. 2024. hyDNS: Acceleration of DNS Through Kernel Space Resolution. In *Workshop on eBPF and Kernel Extensions (eBPF '24), August 4–8, 2024, Sydney, NSW, Australia.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3672197.3673439

1 Introduction

The Domain Name System (DNS) plays a cardinal role in the Internet infrastructure, enabling the bidirectional association of domain names with IP addresses [22]. Nowadays, DNS is involved in a mix of applications ranging from the initial specification to new and innovative use cases. It has become an important infrastructure that allows applications to map individual users to specific content [3]. In data centers, a variety of resources are referred to by their Uniform Resource Locator and accessed via this URL [18] using DNS. Named data networks and software-defined networks also depend on DNS [8]. DNS has been widely used by major content delivery networks (CDNs) to infer client location and direct the client request to the optimal server [13]. Given the importance of DNS for end-user experience and how much the DNS system has changed over the last decade, reducing the time required to resolve domain names into IP addresses brings substantial benefits, which lead to not only an improved user experience, but also increased revenue of online service providers [30].

The existing DNS system adopts a hierarchical architecture and runs entirely in user space, which most commonly uses UDP's connectionless transport to facilitate quick transactions. The DNS system is composed of a client-side component (recursive resolvers), and server-side components (root DNS servers, top-level domain (TLD) servers, and authoritative nameservers). The recursive resolver accepts requests from end-users for a domain name translation and contains a memory cache to maintain the mapping between the domain name and IP address. If a valid translation is not found in the resolver's cache, recursive resolvers are responsible for contacting the appropriate servers to resolve any query sent by the user. A recursive resolver iteratively searches by first starting from the root and then following delegations down the naming hierarchy, until reaching a nameserver that is responsible for the domain of the query and finally returning an answer [28]. The procedure results in a total of three requests and replies as shown in Figure 1.



(a) Without eBPF **(b) eBPF Offloading**

Figure 1: Packet Path without eBPF vs. eBPF Offloading

In practice many more messages may be involved, sometimes up to one hundred pairs of requests and replies are needed to resolve a query [2]. The resolvers are involved in all of these messages and thus play the most important role in DNS resolution.

Most of the existing DNS research focuses on the problems and performance optimization on the server-side components, which includes devising more scalable delegation and distribution systems in the authoritative nameservers [28], proactively caching and structured peer-to-peer overlays[26], and replicating the entire DNS database on geographically distributed servers [15]. However, due to DNS's increasing memory sizes and DNS's high cachability [23], cache hits at the recursive resolver are approaching 90% [14, 35], so fewer requests are dependent on server-side performance at the root, TLD, and authoritative nameservers. Based on this fact, improving the performance of the recursive resolver, which has received less attention in research, is the focus of our work. In this paper, we consider the recursive resolver to make up the client-side of the DNS architecture. All current recursive resolver implementations run in user space [9], which not only involves expensive transitions between user space and kernel space, but also extensive message traversal cost inside the host network stack. In particular, when the resolver suffers a cache miss, it is required to initiate multiple pairs of request and response by contacting the appropriate upstream DNS servers until an answer is received, making the cost more severe. Our measurement shows that about half of the CPU time is spent on the kernel networking stack even in a typical cache-hit DNS deployment, shown in § 2.

To address the cost of message copying and network traversal, kernel-bypass technologies DPDK [1] or RDMA [33] are often considered. However, DPDK requires a user space networking stack to maintain compatibility with the existing kernel networking stack. Additionally, DPDK uses busy-polling for network IO, which consumes 100% CPU cycles and is wasteful when the workload is low. RDMA [40] [12] [17] faces its own unique deployment challenges in long-haul transmission scenarios [43]. In addition, kernel-bypass also eliminates security policies enforced by the kernel [11].

In this paper, we propose hyDNS to overcome the kernel bottlenecks impacting the recursive resolver by ensuring to bypass the network-stack packet processing and offloading of the name resolution functionalities to the kernel through eBPF(extended Berkeley Packet Filter) [21]. hyDNS implements the functionalities in the eXpress Data Path (XDP) layer [32], which can intercept requests directly from the network driver before delivering packets to the standard network stack. If the local recursive resolver does not have the cache entry for a request, then the recursive resolver needs to locally pend the request and then issue requests to upstream servers in the DNS system. Pending requests will be added to an eBPF map to consume less DMA memory. Once the number of pending requests reaches a threshold, the additional requests will be passed to the user space for processing. To make the system scalable, hyDNS uses programmable NIC capabilities to implement a lockless cache with per-core eBPF maps by programming filters on the programmable NIC to dispatch requests. These requests will be delivered to a core responsible for this record. Unlike kernel-bypass solutions, the eBPF verifier [10] establishes the safe execution of offloading functionalities through validation of a program's memory safety and information flow security while loading.

The main contributions of this paper are as follows:

- We identify the operational bottlenecks in a recursive DNS resolver and show that more than 50% of the CPU time is spent inside the kernel network stack regardless of cache-hit or miss on the resolver.
- We propose hyDNS, a hybrid DNS system with eBPF offloading in the kernel and an arbitrary user space resolver implementation to serve a large volume of DNS queries from the recursive resolver to provide low latency resolution. We also present the mechanisms to address the limitations of the DMA memory region for pending requests in the kernel.
- We design a scalable kernel cache using programmable NIC devices and per-core lockless eBPF maps. By programming filters on a programmable NIC, we direct the requests to the CPU core that would contain the query mapping in its map table.
- Our preliminary results of a kernel cache system with eBPF offloading show significant performance gains on cache hits and improves throughput by $\sim 4.4x$ a 65% reduction in latency.

hyDNS does not raise any ethical problems as it does not involve human subjects in testing or sensitive data.

2 Background and Motivation

2.1 DNS Resolution Process

The DNS resolution process involves several types of servers working together to fulfill queries, including local DNS resolvers, root DNS servers, top-level domain (TLD) servers, and authoritative nameservers [5, 28].

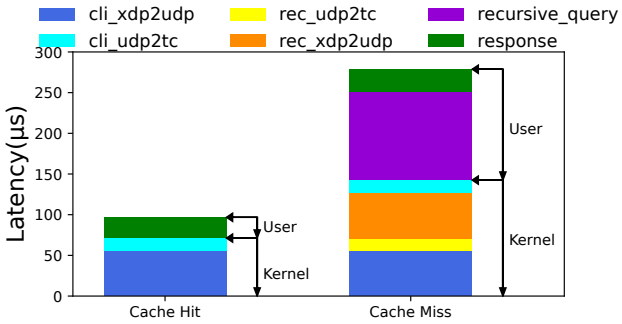


Figure 2: Kernel vs User Space Processing Latency

DNS queries can be resolved through either a recursive or an iterative process [4, 19]. In a recursive query, the local resolver sets the Recursion Desired (RD) bit and sends the query to other servers, which handle the resolution process on its behalf if they also support recursion. In contrast, an iterative resolver directly queries the root, TLD, and authoritative servers until it finds the answer. Recursive resolvers offer faster access to cached data, while iterative resolvers provide better caching of intermediate step, such as finding the nameserver for a given domain name [5].

2.2 Motivation

To assess the potential benefits of implementing a DNS resolver in kernel space using eBPF, we conducted a benchmarking experiment to measure the processing latency in both kernel and user space for DNS queries. We developed a benchmarking application that uses eBPF kernel probes and XDP and TC hooks to store timestamps in an eBPF map, allowing us to measure the time spent processing the DNS packets in the kernel.

For user space latency measurements, we modified the SmartDNS [24] resolver to store timestamps in a hash table, which we then processed. We considered the total user space latency for cache hits and misses, accounting for the time spent parsing queries, searching the cache, and processing responses from upstream DNS servers.

The results of our experiment, shown in Figure 2, demonstrate that kernel space processing accounts for a significant portion of the total client-side processing latency: about 73.6% for cache hits and about 51.1% for cache misses with only one intermediate request made to the upstream server. The time spent in the network stack becomes more prominent as the number of intermediate requests sent to different upstream servers increase. We do not consider the end-to-end latency as the time a request spends traveling through the Internet varies greatly and can be very small in a cloud datacenter setting, as all hosts are in the same physical location.

Most of the kernel space processing time is between the XDP hook and the _skb_recv_udp() function, which is the point where the packet is copied from the kernel into user space. A smaller amount of kernel processing is found in the sending path between the udp_sendmsg() function, the function that is responsible for switching from user to kernel space to send a packet, and the TC egress hook, which is the last hook point in the transmit path that we can access the packet using eBPF in the kernel.

From these tests, we see that a large percentage of the latency is due to traversing the kernel networking stack and copying the packets into user space memory. These findings indicate that offloading the DNS resolver functionality into kernel space using eBPF could substantially reduce the overall latency, as offloading into the kernel will avoid unnecessary packet processing and copying.

Based on these observations, we propose the design of hyDNS, a hybrid resolver supporting both eBPF offloading in the kernel and user space resolution to handle requests when there is limited DMA memory in the kernel for bending requests. This hybrid approach enables hyDNS to eliminate the expensive memory copy operations between the user space and kernel space and avoid the kernel network-stack traversal cost for the majority of the requests, while addressing limitations of the kernel.

3 Design

In this section, we present the design of hyDNS, a hybrid resolver for the DNS system. Figure 3 shows the high level architecture of the proposed design. hyDNS enables the acceleration of a DNS recursive resolver by offloading the resolution functionalities into the XDP network layer in the kernel, which can serve the DNS queries at the earliest possible point on the receiving side of network stack. However,we must store pending requests in the DMA memory region as we resolve the queries by forwarding requests to upstream DNS servers. Once the available DMA memory size is beyond a certain threshold, hyDNS will begin to forward the cache-miss requests to the userspace resolver. hyDNS will also forward requests differing from the common UDP DNS requests, such as DNS over TLS and DNS over HTTPS, both of which are supported by many user space implementations but are not supported by hyDNS. Further, hyDNS uses programmable NIC capabilities to provide a scalable kernel DNS cache, which is a lockless per-core eBPF map table. By programming the filters on the programmable NIC, hyDNS directs the DNS queries to the respective CPU core responsible for keeping the appropriate DNS records in their cache.

3.1 Hybrid Model

We implement hyDNS to allow for both kernel and user space handling of requests. To achieve this objective, it is essential to handle the DNS requests entirely in the kernel and also provide a bypass option to handle subsequent requests from user space. For small workloads, handling DNS requests entirely in kernel will significantly improve throughput and latency. However, for large workloads where it is not possible to queue all the requests in the kernel, it is desirable to handle requests from user space to avoid any throughput and latency penalties.

hyDNS leverages a hybrid kernel and user space model to achieve better performance than kernel processing alone. As we see in Figure 7, the eBPF solution for caching can have much longer tail latencies than the user space solution at higher workloads. Despite these longer tail latency, we observed that under every workload tested, the eBPF-based solution provides higher throughput than the user space implementation. Further, eBPF based solution achieves better mean and p75 latency. For this reason, we decided to pursue a hybrid solution that allows resolution of DNS queries in both kernel and user space. When the kernel has too many queries to process, these queries can be passed on to user space where the user space DNS resolver can handle the requested queries.

Note that the kernel space eBPF resolver of hyDNS will be invisible to a user space resolver. This allows hyDNS to work with any

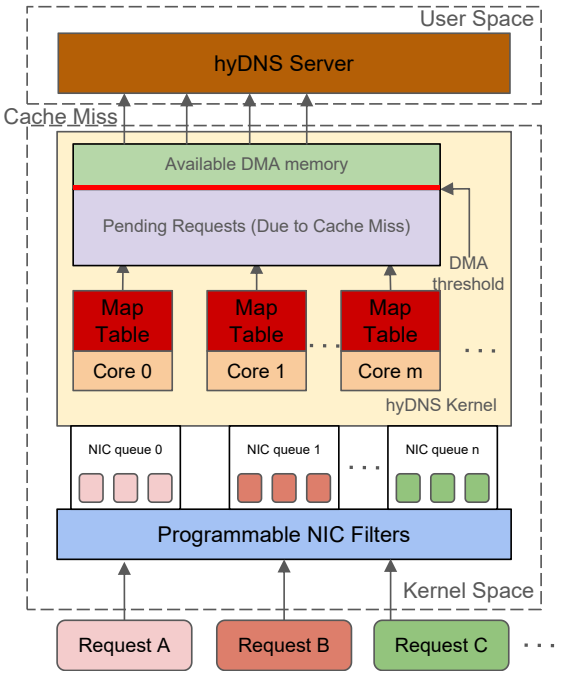


Figure 3: hyDNS Architecture

user space implementation, as the user space implementation will only handle requests that are passed to it from the kernel resolver.

One of the main challenges of handling the resolution of DNS queries in the kernel is the asynchronous nature of resolving the queries. In order to serve a single request, the recursive resolver often needs to send and receive DNS messages from multiple different upstream DNS servers. Conversely, an XDP program must return an appropriate action for the packet before the next will be processed. To deal with this challenge, we create an eBPF map of the current outstanding requests. This eBPF map enables to maintain the mapping of the requested record to the client that requested it. When an upstream response is received and contains the correct record, the XDP program will get the IP address of the client from this map to send the response. The requests stored in this eBPF map are known as the pending requests.

Another critical design challenge is to determine which DNS queries to handle in the kernel and which queries to pass to user space. In § 4, we compare the Xpress DNS in-kernel cache to the user space SmartDNS cache and observe that Xpress DNS achieves better latency up to a certain amount of concurrent DNS requests. Xpress DNS is an in-kernel cache that caches DNS A records using an eBPF map [27]. We use this data to find a threshold for kernel processing of queries and when the number exceeds this threshold, we forward the DNS queries to user space. Again, when the number of pending requests drops below the threshold, then hyDNS would automatically begin processing the DNS requests within the kernel. The exact number or throughput of queries to set as the threshold may be dependent on the query size, query rate, and hardware specifications, which needs to be tested further.

Another consideration to make is whether or not to intercept outgoing packets from the user space DNS resolver. Such an operation can allow us to add DNS records to the in-kernel cache, even when the resolution is done in user space. However, this introduces some additional challenges, such as tracking of the CPU cores where the DNS records need to be cached. Further, we need to add a new eBPF program at the traffic control (TC) egress hook, as there is no access to the packet beyond that point in software on the sender's side. We believe that calling the eBPF program for every packet exiting the kernel from user space may contribute drastically to the overall latency, making this addition unreasonable.

3.2 Scalable Caching

DNS resolvers need to be able to support high traffic workloads. For this reason, a caching solution should be able to scale to accommodate these higher workloads. A solution to allow for a more scalable kernel cache is to create per-CPU BPF maps. The per-CPU maps allow the cache size to scale with the number of CPUs of the machine that will be running the resolver. To fully utilize the per-CPU caches, a SmartNIC can be used as a load balancer to distribute traffic among the CPUs. Policies can be created to route packets to specific CPUs based on different header fields. Leveraging these SmartNIC capabilities allows us to ensure that the same queries will be handled by the same CPU, allowing full usage of the extra cache from the per-CPU maps. This separation of DNS traffic also solves issues relating to cache consistency, as all of the CPU cores are handling separate traffic and should not have overlapping data.

The number of the per-core eBPF maps must also be considered. eBPF maps require keys and values to have a set size and type, making it difficult to add all record types to the same eBPF map. It may be beneficial to create different maps for the different record types and to distribute the traffic based on the record type of the queries. For example, the first two cores handle all A records, the third core handles AAAA records, and the fourth core handles NS and CNAME records. This gives an easy way to divide the traffic between the cores, but will require some tuning to determine the percentage of traffic that each query type typically account for.

Another consideration to be made in the design of hyDNS is the offloading of the eBPF XDP program to the SmartNIC. This would give an increased benefit of lowering the processing latency, as incoming packets would not have to be copied into the kernel. The downside of this design consideration is the limited processing power of the SmartNIC. While implementing hyDNS, we will evaluate the performance of both of these solutions to see which will give the greatest performance benefits.

4 Evaluation

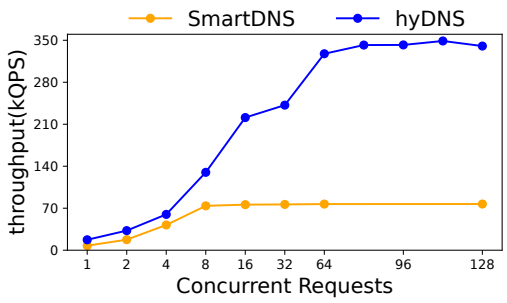
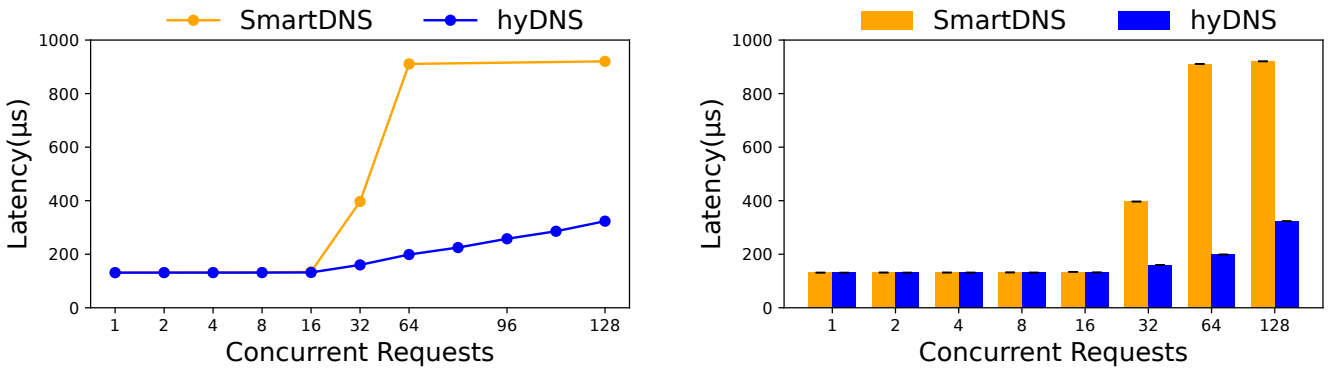


Figure 4: Throughput of hyDNS and SmartDNS cache



(a) Mean Latency

(b) Mean & Confidence Intervals

Figure 5: Mean Latency Comparison

Testbed. Our experiments were conducted on the Chameleon Cloud testbed using two Intel(R) Xeon(R) Gold 6126 CPU @ 2.60GHz devices. They have 12 cores each and are connected via 10 Gbps network. The switch between the two nodes is a Dell S6000-ON switch. One device served as the DNS server, while the other acted as the client generating DNS queries [16] using a benchmark called dnspyre [31] to generate DNS queries. Dnspyre allows us to set a number of concurrent requests to be sent, as well as the query type and domain name, allowing us to specifically test cache hits using a repeated query. Our tests were ran with A records, which are a domain name to IPv4 mapping common in many DNS scenarios. All tests used were run for one minute. We use SmartDNS as the baseline, which is a popular user-space recursive resolver. We set the cache size to support 65, 536 entries in both implementations, but we use repeated queries to guarantee a 100% cache hit rate in the experiments. The setup does not use the programmable NIC support described in the previous sections.

4.1 Throughput Analysis

Figure 4 compares the throughput of hyDNS and SmartDNS under different concurrency levels. At lower concurrencies (1, 2, and 4), the difference in throughput between the two systems is minimal, with both hyDNS and SmartDNS achieving similar performance. This suggests that for low-traffic scenarios or when the number of concurrent requests is small, the choice between kernel space and user space caching may not have a significant impact on throughput.

However, as the concurrency level increases, hyDNS begins to demonstrate a significant performance advantage over SmartDNS. Starting from a concurrency of 8, the throughput of SmartDNS plateaus at slightly over 70,000 QPS(Queries per Second), indicating that it reaches its maximum capacity and cannot handle additional concurrent requests effectively. In contrast, hyDNS continues to scale up its throughput as the concurrency level increases, reaching a peak of about 340,000 QPS at a concurrency of 64. This represents a nearly 5x improvement in throughput compared to SmartDNS under high concurrency conditions.

The scalability and higher throughput achieved by hyDNS can be attributed to its kernel-space caching mechanism, which leverages eBPF and XDP to process DNS queries entirely within the kernel. By eliminating the need for context switches between user space and kernel space, hyDNS reduces the overhead associated with packet processing and cache lookups. This allows it to handle a larger number of concurrent requests more efficiently, resulting in improved throughput at higher concurrency levels.

Furthermore, the use of XDP enables hyDNS to intercept and process DNS queries at the earliest possible stage in the network stack, minimizing the processing overhead and latency. This early interception and processing of packets contribute to the higher throughput observed in hyDNS, especially when the system is under heavy load. It is worth noting that while hyDNS demonstrates significant throughput improvements, it also reaches a point of saturation at a concurrency of 64, beyond which the throughput starts to level off. This suggests that there may be room for further optimization to push the scalability limits of hyDNS even higher. Techniques such as load balancing, and fine-tuning of eBPF and XDP map size parameters could potentially help in achieving even better throughput performance.

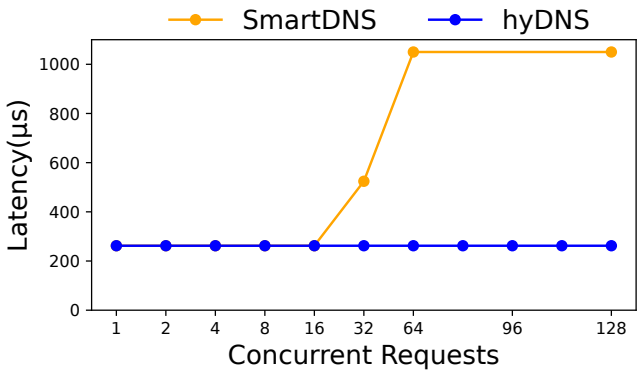
4.2 Latency Analysis

To gain a comprehensive understanding of the latency characteristics of hyDNS and SmartDNS, we analyzed various latency metrics across different concurrency levels.

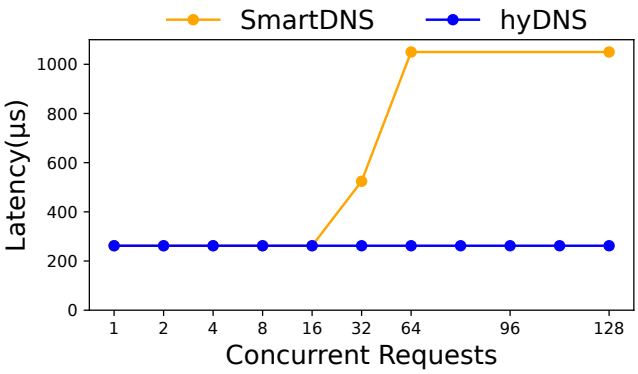
Figure 5(a) presents the mean latency comparison between hyDNS and SmartDNS. The results show that hyDNS consistently exhibits lower mean latency compared to SmartDNS for concurrency levels up to 128. At lower concurrencies (1 to 16), the difference in mean latency is relatively small, indicating that both systems can handle low-traffic scenarios effectively. However, starting from a concurrency of 32, the mean latency of SmartDNS begins to increase sharply, while hyDNS maintains a relatively stable and low mean latency. This trend continues up to a concurrency of 128, where hyDNS still provides lower mean latency than SmartDNS.

To assess the statistical significance of the latency differences between hyDNS and SmartDNS, we calculate the confidence intervals for the mean latency, as shown in Figure 5(b). The non-overlapping confidence intervals concurrency levels (32 and above) demonstrate that the latency improvements of hyDNS over SmartDNS are statistically significant and not due to random variations.

In addition to the mean latency, we also examine the p50 and p75 latency metrics to understand the distribution of latencies. Figure 6(a) and Figure 6(b) illustrate the p50 and p75 latencies, respectively, for hyDNS and SmartDNS across different concurrency



(a) p50 Latency



(b) p75 Latency

Figure 6: Percentile Latency Comparison

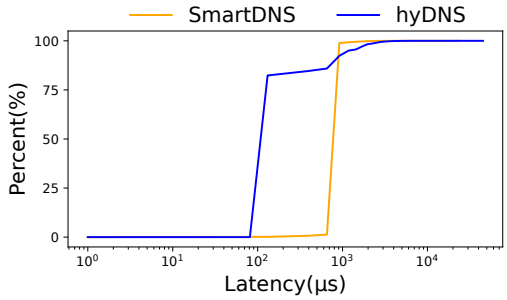


Figure 7: CDF of latencies for hyDNS and SmartDNS

levels. For hyDNS, the p50 and p75 latencies remain stable and low up to a concurrency of 128, indicating that a majority of the requests are served with consistent and low latency. In contrast, SmartDNS experiences an increase in p50 and p75 latencies starting from a concurrency of 32, suggesting that a larger fraction of requests have higher latencies as the load increases.

To further analyze the distribution of latencies, Figure 7 presents the cumulative distribution function (CDF) of latencies for hyDNS and SmartDNS at a concurrency of 128. The plot shows that approximately 80% of the requests in hyDNS are served with lower latency compared to SmartDNS. This shows that hyDNS provides a latency advantage for the majority of the requests.

However, it is important to note that the CDF also highlights a potential limitation of hyDNS. The tail end of the CDF shows that a small fraction of requests in hyDNS experience higher latencies compared to SmartDNS. This suggests that hyDNS may be more susceptible to outliers or extreme latencies under heavy load conditions. Investigating and addressing this tail latency issue could be an area for future improvement in hyDNS. This tail latency serves as a main motivation for implementing hyDNS to allow for kernel and user space processing of DNS traffic.

The lower latencies achieved by hyDNS can be attributed to several factors. Firstly, the kernel-space caching mechanism employed by hyDNS eliminates the need for context switches between userspace and kernel space, reducing the processing overhead and latency. Secondly, the use of eBPF and XDP allows hyDNS to intercept and process DNS queries at the earliest possible stage in the network stack, minimizing processing time. Lastly, the efficient caching and lookup mechanisms of hyDNS contribute to faster response times, especially for frequently accessed records. This research does not involve any ethical issues.

5 Related Work

eBPF-based Application: eBPF has been extensively used not only for monitoring [34, 38] and security [6, 29], but also to accelerate the networking applications [7, 11, 20, 25, 36, 37, 39, 41, 42]. Katran [7] and HEELS [36] implement an eBPF-based load balancer. BMC [11] offloads UDP-based GET operations of Memcached to XDP hooks by implementing an eBPF-map cache. Electrode [42] accelerates distributed protocols using safe in-kernel eBPF-based packet processing. Logreducer [37] uses eBPF to implement a non-intrusive and language-independent log reduction framework. XRP [39] improves storage IO performance by bypassing the kernel's block, file system and system call layers. Spright [25] uses eBPF's socket message mechanism to support shared memory processing. XAgg [39] deploys an eBPF-based aggregator to accelerate heterogeneous gradient aggregation in distributed machine learning. Inspired by these works, hyDNS leverages eBPF to accelerate the performance of DNS resolvers by offloading key DNS functionalities into the XDP layer.

DNS Optimization: Traditional DNS-related research has focused on the problems and performance optimization in the server-side DNS infrastructure. Akamai DNS [28] proposes a two-tier delegation system and distributes DNS queries among locations by anycast traffic engineering in the authoritative nameservers, and then provides rapid answers with low TTLs. CoDoNS [26] uses structured peer-to-peer overlays and analytically informed proactive caching to provide high lookup performance. Kangasharju [15] proposes replicating the entire DNS database on geographically distributed servers, significantly reducing DNS lookup time. hyDNS focuses on the performance optimization of the recursive resolver.

6 Conclusion

We present hyDNS, a hybrid DNS resolver that accelerates DNS recursive resolution using eBPF-based packet processing to improve the performance of DNS queries. hyDNS intercepts DNS queries at the XDP NIC driver layer just as the DNS request packets arrive at the XDP layer from the NIC to offer high throughput and low latency DNS resolution with negligible overhead. To accommodate a surge of cache-miss queries and the limited size of DMA memory region, once the number of pending requests exceeds a certain threshold, all additional requests are passed to the user space for processing. Our preliminary results show substantial performance benefits *i.e.*,$\sim 4.4x$ improvement in throughput and $\sim 3x$ reduction in latency compared to the traditional user space DNS resolvers.

References

[1] [n. d.]. DPDK. https://www.dpdk.org/.

[2] Yehuda Afek, Anat Bremler-Barr, and Lior Shafir. 2020. {NXNSAttack}: Recursive {DNS} Inefficiencies and Vulnerabilities. In *29th USENIX Security Symposium (USENIX Security 20)*. 631–648.

[3] Bernhard Ager, Wolfgang Mühlbauer, Georgios Smaragdakis, and Steve Uhlig. 2010. Comparing DNS resolvers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. 15–21.

[4] Rami Al-Dalky, Michael Rabinovich, and Mark Allman. 2018. Practical challenge-response for DNS. *ACM SIGCOMM Computer Communication Review* 48, 3 (2018), 20–28.

[5] Thomas Callahan, Mark Allman, and Michael Rabinovich. 2013. On modern DNS behavior and properties. *ACM SIGCOMM Computer Communication Review* 43, 3 (2013), 7–15.

[6] Cilium. [n. d.]. Cilium - Cloud Native, eBPF-based Networking, Observability, and Security — cilium.io. https://cilium.io/. [Accessed 22-05-2024].

[7] Facebook. [n. d.]. Katran. [EB/OL]. https://github.com/facebookincubator/katran Accessed Oct 25, 2020.

[8] Hongyu Gao, Vinod Yegneswaran, Yan Chen, Phillip Porras, Shalini Ghosh, Jian Jiang, and Haixin Duan. 2013. An empirical reexamination of global DNS behavior. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*. 267–278.

[9] Hongyu Gao, Vinod Yegneswaran, Jian Jiang, Yan Chen, Phillip Porras, Shalini Ghosh, and Haixin Duan. 2014. Reexamining DNS from a global recursive resolver perspective. *IEEE/ACM Transactions on Networking* 24, 1 (2014), 43–57.

[10] Elazar Gershuni, Nadav Amit, Arie Gurfinkel, Nina Narodytska, Jorge A Navas, Noam Rinetzky, Leonid Ryzhyk, and Mooly Sagiv. 2019. Simple and precise static analysis of untrusted linux kernel extensions. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 1069–1084.

[11] Yoann Ghigoff, Julien Sopena, Kahina Lazri, Antoine Blin, and Gilles Muller. 2021. {BMC}: Accelerating Memcached using Safe In-kernel Caching and Pre-stack Processing. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. 487–501.

[12] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. 2016. RDMA over commodity ethernet at scale. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 202–215.

[13] Hadrien Hours, Ernst Biersack, Patrick Loiseau, Alessandro Finamore, and Marco Mellia. 2016. A study of the impact of DNS resolvers on CDN performance using a causal approach. *Computer Networks* 109 (2016), 200–210.

[14] Jaeyeon Jung, Emil Sit, Hari Balakrishnan, and Robert Morris. 2001. DNS performance and the effectiveness of caching. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*. 153–167.

[15] Jussi Kangasharju and Keith W Ross. 2000. A replicated architecture for the domain name system. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064)*, Vol. 2. IEEE, 660–669.

[16] Kate Keahey, Jason Anderson, Zhuo Zhen, Pierre Riteau, Paul Ruth, Dan Stanzione, Mert Cevik, Jacob Colleran, Haryadi S. Gunawi, Cody Hammock, Joe Mambretti, Alexander Barnes, François Halbach, Alex Rocha, and Joe Stubbs. 2020. Lessons Learned from the Chameleon Testbed. In *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)*. USENIX Association.

[17] Xinhao Kong, Yibo Zhu, Huaping Zhou, Zhuo Jiang, Jianxi Ye, Chuanxiong Guo, and Danyang Zhuo. 2022. Collie: Finding Performance Anomalies in {RDMA} Subsystems. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 287–305.

[18] Haifeng Liu, Shugang Chen, Yongcheng Bao, Wanli Yang, Yuan Chen, Wei Ding, and Huasong Shan. 2018. A high performance, scalable dns service for very large scale container cloud platforms. In *Proceedings of the 19th International Middleware Conference Industry*. 39–45.

[19] Zhuoqing Morley Mao, Charles D Cranor, Fred Douglis, Michael Rabinovich, Oliver Spatscheck, and Jia Wang. 2002. A Precise and Efficient Evaluation of the Proximity Between Web Clients and Their Local DNS Servers.. In *USENIX Annual Technical Conference, General Track*. 229–242.

[20] Andrea Mayer, Pierpaolo Loreti, Lorenzo Bracciale, Paolo Lungaroni, Stefano Salsano, and Clarence Filsfils. 2021. Performance monitoring with h^2: Hybrid kernel/ebpf data plane for srv6 based hybrid sdn. *Computer Networks* 185 (2021), 107705.

[21] Steven McCanne and Van Jacobson. 1993. The BSD Packet Filter: A New Architecture for User-level Packet Capture.. In *USENIX winter*, Vol. 46. 259–270.

[22] Paul V Mockapetris. 1987. RFC1034: Domain names-concepts and facilities.

[23] KyoungSoo Park, Vivek S Pai, Larry L Peterson, and Zhe Wang. 2004. CoDNS: Improving DNS Performance and Reliability via Cooperative Lookups.. In *OSDI*, Vol. 4. 14–14.

[24] Nick Peng. 2024. SmartDNS. https://github.com/pymumu/smartdns.

[25] Shixiong Qi, Leslie Monis, Ziteng Zeng, Ian-chin Wang, and KK Ramakrishnan. 2022. SPRIGHT: extracting the server from serverless computing! high-performance eBPF-based event-driven, shared-memory processing. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 780–794.

[26] Venugopalan Ramasubramanian and Emin Gün Sirer. 2004. The design and implementation of a next generation name service for the internet. *ACM SIGCOMM Computer Communication Review* 34, 4 (2004), 331–342.

[27] Bas Schalbroeck. 2021. Xpress DNS - Experimental XDP DNS server. https://github.com/zebaz/xpress-dns.

[28] Kyle Schomp, Onkar Bhardwaj, Eymen Kurdoglu, Mashooq Muhaimen, and Ramesh K Sitaraman. 2020. Akamai dns: Providing authoritative answers to the world's queries. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 465–478.

[29] David Soldani, Petrit Nahi, Hami Bour, Saber Jafarizadeh, Mohammed Soliman, Leonardo Di Giovanna, Francesco Monaco, Giuseppe Ognibene, and Fulvio Risso. 2023. ebpf: A new approach to cloud-native observability, networking and security for current (5g) and future mobile networks (6g and beyond). *IEEE Access* (2023).

[30] Erik Sy. 2019. Enhanced Performance and Privacy via Resolver-Less DNS. *arXiv preprint arXiv:1908.04574* (2019).

[31] Tantalor93. [n. d.]. dnspyre. https://github.com/Tantalor93/dnspyre.

[32] Marcos AM Vieira, Matheus S Castanho, Racyus DG Pacífico, Elerson RS Santos, Eduardo PM Câmara Júnior, and Luiz FM Vieira. 2020. Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications. *ACM Computing Surveys (CSUR)* 53, 1 (2020), 1–36.

[33] Xingda Wei, Zhiyuan Dong, Rong Chen, and Haibo Chen. 2018. Deconstructing {RDMA-enabled} distributed transactions: Hybrid is better!. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 233–251.

[34] Tianjun Weng, Wanqi Yang, Guangba Yu, Pengfei Chen, Jieqi Cui, and Chuanfu Zhang. 2021. Kmon: An in-kernel transparent monitoring system for microservice systems with ebpf. In *2021 IEEE/ACM International Workshop on Cloud Intelligence (CloudIntelligence)*. IEEE, 25–30.

[35] C Wills and Hao Shang. 2000. *The contribution of DNS lookup costs to web object retrieval*. Technical Report. Citeseer.

[36] Rui Yang and Marios Kogias. 2023. HEELS: A Host-Enabled eBPF-Based Load Balancing Scheme. In *Proceedings of the 1st Workshop on eBPF and Kernel Extensions*. 77–83.

[37] Guangba Yu, Pengfei Chen, Pairui Li, Tianjun Weng, Haibing Zheng, Yuetang Deng, and Zibin Zheng. 2023. Logreducer: Identify and reduce log hotspots in kernel on the fly. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1763–1775.

[38] Timothy D Zavarella. 2022. *A methodology for using eBPF to efficiently monitor network behavior in Linux Kubernetes clusters*. Ph. D. Dissertation. Massachusetts Institute of Technology.

[39] Qianyu Zhang, Gongming Zhao, Hongli Xu, and Peng Yang. 2023. XAgg: Accelerating Heterogeneous Distributed Training Through XDP-Based Gradient Aggregation. *IEEE/ACM Transactions on Networking* (2023).

[40] Yiwen Zhang, Yue Tan, Brent Stephens, and Mosharaf Chowdhury. 2022. Justitia: Software {Multi-Tenancy} in Hardware {Kernel-Bypass} Networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 1307–1326.

[41] Yuhong Zhong, Haoyu Li, Yu Jian Wu, Ioannis Zarkadas, Jeffrey Tao, Evan Mesterhazy, Michael Makris, Junfeng Yang, Amy Tai, Ryan Stutsman, et al. 2022. {XRP}:{In-Kernel} Storage Functions with {EBPF}. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 375–393.

[42] Yang Zhou, Zezhou Wang, Sowmya Dharanipragada, and Minlan Yu. 2023. Electrode: Accelerating Distributed Protocols with {eBPF}. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 1391–1407.

[43] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. 2015. Congestion control for large-scale RDMA deployments. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 523–536.