In-Network Address Caching for Virtual Networks

Lior Zeno
Technion

Ang Chen
University of Michigan

Mark Silberstein
Technion

Abstract

Packet routing in virtual networks requires virtual-to-physical address translation. The address mappings are updated by a single party, i.e., the network administrator, but they are read by multiple devices across the network when routing tenant packets. Existing approaches face an inherent read-write performance tradeoff: they either store these mappings in dedicated gateways for fast updates at the cost of slower forwarding or replicate them at end-hosts and suffer from slow updates.

SwitchV2P aims to escape this tradeoff by leveraging the network switches to transparently cache the address mappings while learning them from the traffic. SwitchV2P brings the mappings closer to the sender, thus reducing the first packet latency and translation overheads, while simultaneously enabling fast mapping updates, all without changing existing routing policies and deployed gateways. The topology-aware data-plane caching protocol allows the switches to transparently adapt to changing network conditions and varying in-switch memory capacity.

Our evaluation shows the benefits of in-network address mapping, including an up to 7.8× and 4.3× reduction in FCT and first packet latency respectively, and a substantial reduction in translation gateway load. Additionally, SwitchV2P achieves up to a 1.9× reduction in bandwidth overheads and requires order-of-magnitude fewer gateways for equivalent performance.

CCS Concepts

• **Networks → Data center networks**; **Naming and addressing**; **Programmable networks**.

Keywords

Network virtualization, virtual-to-physical IP translation, in-network caching.

ACM Reference Format:
Lior Zeno, Ang Chen, and Mark Silberstein. 2024. In-Network Address Caching for Virtual Networks. In *ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24), August 4–8, 2024, Sydney, NSW, Australia.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3651890.3672213

1 Introduction

Network virtualization enables the overlay of customized virtual topologies atop a single physical network. In a cloud setting, tenants

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 components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ACM SIGCOMM '24, August 4–8, 2024, Sydney, NSW, Australia
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0614-1/24/08
https://doi.org/10.1145/3651890.3672213

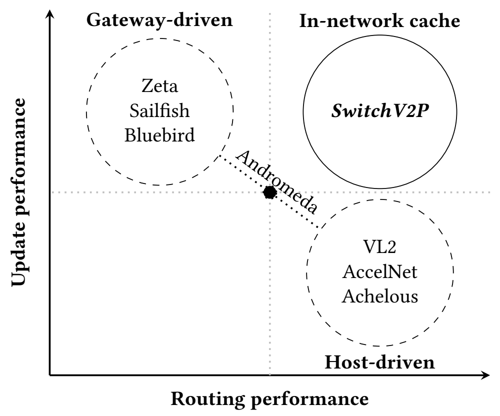


Figure 1: The tradeoff of V2P translation mechanisms between the lookup and update performance. SwitchV2P aims to escape from this tradeoff.

retain complete control over their virtual networks, enabling them to configure IP assignments and routing policies [5, 17, 32, 41, 54].

Virtual IP assignments are decoupled from the underlying physical topology [32], so virtual IPs are mere identifiers lacking network location information. Thus, routing a packet requires performing *virtual-to-physical translation*, i.e., resolving the virtual destination address into the corresponding physical address. These virtual-to-physical (V2P) mappings are frequently modified by the virtual network control plane due to the constant influx and departure of virtual machines and containers [2, 53, 56].

As a result, a V2P translation mechanism must minimize the lookup time on the critical path of the packet routing, and at the same time support frequent updates to the V2P mappings at scale. The challenge lies in harmonizing these goals, as the former leans towards a fully distributed design that stores mappings at the sender, while the latter leans towards a centralized gateway design that enables efficient updates. Although this is a longstanding issue, a solution that can satisfy both requirements remains elusive.

Figure 1 delineates these two fundamental approaches in state-of-the-art solutions. In a fully distributed *host-driven design* such as VL2 [18], AccelNet [15, 16], and Achelous [53], the mappings are installed in the hypervisors or SmartNICs. The resulting routing performance is high as the lookups are local to senders, but updates scale poorly due to the control plane overhead of proactively updating the mappings in the hypervisors across the network [53]. In contrast, in a *gateway-driven design*, such as Zeta [56], Sailfish [44], and Bluebird [6], the mappings are stored in dedicated network entities, *gateways*. The updates are efficient as they are performed in a few dedicated locations, but the routing performance deteriorates because of the increased first packet latency, gateway lookup overheads, and the emergence of in-network hotspots, as we show in the analysis in §5.1.

Andromeda [11] explores a hybrid design where the mappings are initially managed in the gateways, but the most frequent source-destination pairs (e.g., mappings for elephant flows), are dynamically installed in the source's hypervisor at runtime. In fact, Andromeda enables the network administrators to select a certain operating point along the tradeoff between the lookup performance and update speed, yet it is inherently constrained by that tradeoff, which we strive to escape in our work.

Conceptually, V2P translation can be viewed as an instance of the classical problem of state sharing in a distributed system. The state (V2P mapping of an IP) is written by a single writer (network administrator), and read by multiple readers (senders to that IP). The read-write performance tradeoffs are inherent in such systems.

Looking at the problem from this angle reveals a previously overlooked design opportunity to improve the V2P translation mechanism *by controlling the replication factor of the mappings.* As has been broadly explored in distributed systems, e.g., in distributed key-value caches [35], there is a tradeoff between the number of cached replicas and the update overhead. Therefore, such systems explicitly control the cache replication factor to optimize read-write performance tradeoffs. In contrast, none of the V2P translation mechanisms offer a similar level of control over the V2P caching. Whereas the gateway-driven designs either do not use V2P caches or statically restrict the cache to the gateway ToR [44], the host-driven systems take the replication to an extreme by caching all the mappings in every reader. Andromeda's cache is more flexible but still too coarse-grain: it caches mappings based on their usage frequency, but conservatively keeps all the rest uncached. We posit that more flexible caching mechanisms for V2P mappings may achieve a better balance between the lookup latency and the update performance.

Following this observation, our idea is to cache the V2P mappings in the switches, and thus resolve virtual addresses as the packets en route. For cache hits, forwarding does not consult the gateway. Compared to caching at the sender, this approach reduces the cache replication factor significantly, making it proportional to the number of switches instead of the number of physical machines. As a result, it speeds up the updates and enables caching of less popular mappings. Moreover, the in-switch cache is closer to the sender than the gateway so the packets that hit the cache are forwarded faster. Additionally, caching reduces the load on the gateways, improving their responsiveness and decreasing cost.

We realize this idea in **SwitchV2P**, an *in-network* translation and forwarding mechanism that builds on a novel data-plane protocol for caching V2P mappings in the network switches. The mappings are initially stored in the gateways as before, but SwitchV2P's in-network cache tier seamlessly learns the mappings from the traffic, entirely in the data plane. SwitchV2P does not modify the existing routing policies, so the packets are forwarded as usual unless the destination mapping is found in the cache.

Unlike prior works that store the whole V2P database in ToR switches [6], we choose a caching approach to simplify the deployment under limited in-switch memory. Our key premise is that, with a substantial degree of destination reuse within both a single flow and across multiple flows, even relatively small in-network caches can improve performance. Furthermore, as the cache is distributed across all the levels of the network topology, including the core switches, this design enables more efficient use of in-switch memory space by sharing a single cache entry in higher-level switches among flows originating from different hosts, racks, or pods. In addition, the opportunistic nature of the caching approach makes it resilient to switch failures, as they do not affect the correctness of packet forwarding. Last, the cache can be implemented and deployed without changing existing virtual networking systems.

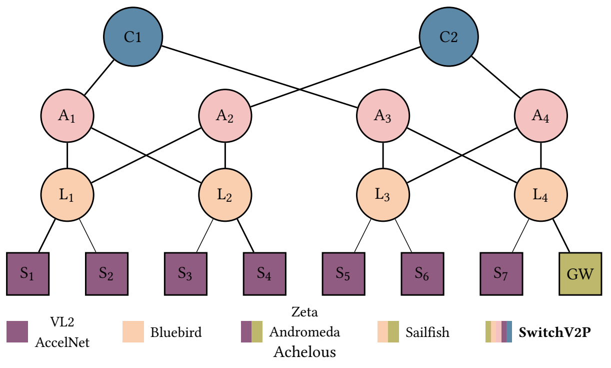


Figure 2: Comparison of previous approaches to V2P mapping. Different colors represent the locations of the mappings in the respective approach. SwitchV2P caches the mappings in all the switches in the network topology.

However, designing a distributed, dynamic caching tier in the data plane, especially under the severe memory capacity constraints of each switch is a formidable challenge. First, learning the mappings from traffic may not always be productive and waste cache capacity: network paths where mappings can be learned (e.g., from the gateway to the destination) may not intersect with the paths where forwarding logic needs them, (e.g., from the source to the gateway). Second, different switches might store duplicate redundant entries thus reducing the effective cache capacity. So it is imperative to promote certain mappings to higher-level switches to eliminate such duplicates in lower-level switches. Third, the coordination between the in-switch caches in the network should have low overhead. Finally, mapping updates, e.g., due to VM migration or workload changes, must be implemented efficiently across switches while guaranteeing packet forwarding correctness.

We address all these challenges in SwitchV2P, prototype it in Tofino switches [26], and comprehensively evaluate using NS3 [43] simulations across a range of real-world traces and diverse settings, including different network topologies. Our results show that SwitchV2P delivers significant performance benefits, improving FCT by up to 7.8× and reducing first packet latency by up to 4.3×. In particular, SwitchV2P delivers a 1.6× improvement in FCT and a 2.9× reduction in first packet latency compared to host-driven approaches. These improvements are achieved while maintaining miss rates below 1% with realistic in-switch memory sizes. It also offers added benefits such as reduced network load and low migration costs. Our P4 prototype indicates that SwitchV2P has relatively modest resource requirements, making it a practical solution for implementation in commercial off-the-shelf switches.

2 Motivation

Existing approaches are faced with an inherent tradeoff—either they sacrifice routing performance (by placing mappings in dedicated gateway servers), or they sacrifice update performance (by placing mappings in end hosts). The degree of replication for the mappings frames existing solutions, as we illustrate in Figure 2.

2.1 Escaping the Tradeoff in SwitchV2P

SwitchV2P, instead, aims to escape from the constraints that are introduced by this tradeoff. The key idea is that the "replicas" need not be passive receptacles, but can be entities imbued with intelligence. This allows for a unique point in the design space where the degree of replication is much smaller than the number of end hosts, yet both update and lookup performance are much higher than either end of the spectrum in the tradeoff. SwitchV2P proposes to co-opt the network switches into mapping replicas. This builds upon the fact that, from networking hardware design, we know that constrained programmability can be implemented at line speed in commercial off-the-shelf switches.

At first glance, SwitchV2P might look like the hybrid design motivated before, where packets are initially routed to the gateway. However, as they traverse the network topology *en route* to the gateway, the switches can assist in the translation process by looking up their local caches at the time of packet forwarding. On a cache hit, the packets can be promptly directed to their intended destination. On a cache miss, the gateway takes over the task following the usual procedure. Thus, the data plane effectively *learns* mappings from packets, akin to MAC learning in L2 networks. We note that the switches learn V2P mappings and not 5-tuples, so they do not need to perform expensive per-flow state tracking.

The active intelligence in the mapping caches opens up a new design space with salient properties.

First, this approach is opportunistic, implying that it does not require expensive switch failure recovery protocols and can be implemented in existing networks without making any changes to the gateway, end-host networking stack, network configuration, or routing, thus simplifying its adoption. Second, it naturally serves network traffic en route, ensuring it does not increase routing overheads since packets do not take any detours during the lookup process. Third, the in-network cache is managed in the data plane, allowing it to promptly adapt to changing traffic patterns without relying on costly control loops within a centralized control plane. Moreover, it operates without the need for coordination among switches, gateways, or end hosts, facilitating deployment. Last, a high cache hit rate directly implies that most translations occur within the network. Consequently, fewer packets undergo the extended processing at the gateway, resulting in reduced FCT and first packet latency.

With the reduced load on the gateways, it becomes possible to deploy fewer gateways. This will also reduce network traffic because packets follow shorter routes, bypassing gateway pods altogether and reaching their destinations directly. With the reduced mappings stored at the end hosts, the hypervisor state is lean and lookups are accelerated inside network ASICs. Thus, the in-network design combines the low routing overhead benefits of the host-driven design, while avoiding its update scalability limitations by distributing the V2P mappings across switches instead of hosts.[1]

2.2 Impact on the First-Packet Latency

Modern virtual networks must satisfy a variety of centrally managed network policies, such as Access Control Lists (ACLs). Therefore, one may argue that the first packet in each flow has to be forwarded to a gateway or a network controller to apply these policies, so in-network V2P translation will have no impact end-to-end. However, it is worth noting that the ACL decisions are made in the virtual IP address space, as dictated by the tenant organization. Policing does not require translating a virtual address into a physical address and enforcing decisions at that level. In addition, we believe, that in common cases, network policies do not change frequently so they can be pre-installed and enforced by the host. This is the case, for example, for ACL policies in Andromeda and Achelous [11, 38, 53]. In such cases, V2P translation would remain the only reason to access the gateway, and optimizing via in-network cache will have a significant impact on the first-packet latency.

2.3 Expected Effectiveness of Caching

Naturally, in-network V2P caching would be effective only if there is a temporal locality of access to V2P mappings in the network. This locality is obviously abundant within a flow, as the sender keeps sending packets to the same destination. However, this is not enough to warrant the use of in-network caching as the mapping can be naturally cached by the sender, at least while the communication is active.

It turns out that in data center workloads there is a large degree of destination reuse across flows. For example, in the analysis of the Alibaba Cloud traces, over 95% of the total requests are processed by just 5% of the microservices [36]. A similar access pattern has been observed in a typical cloud region by prior work [44]. Thus, the in-network cache enables mappings to be shared across flows and end hosts, an operating point untenable for host-based solutions.

2.4 Cache Design Alternatives

We initially explored the idea of storing V2P mappings in the network switches, utilizing their memory as a distributed hash table, similar to the prior works on Ethernet scalability [31]. Presently, the aggregate memory capacity of all the switches in a data center should be sufficient to accommodate all the mappings (as indicated by Bluebird [6], which reported 192K mappings per switch). However, we ultimately dismissed this approach for several reasons. First, switch failures become critical, requiring inter-switch replication to prevent the loss of V2P mappings. It also becomes essential to replicate hot keys to mitigate in-network hotspots and congestion. Furthermore, it becomes increasingly necessary to create tens of thousands of IP addresses per single server [37], and in-switch storage may not scale well to meet this demand. Lastly, utilizing all the data plane memory for storing mappings is an inefficient use of space, considering that not all mappings are always active all the time, often due to skewed traffic patterns [6, 44].

[1]The servers-to-switches ratio in a k-ary fat-tree topology [3] is $\frac{k}{5}$, which translates to 12.8 for $k = 64$.

3 Design and Implementation

Objectives. SwitchV2P pursues the following goals:

- Reduced forwarding latency with low update overheads;
- No routing policy changes for backward compatibility;
- Incremental deployability;
- Seamless integration with gateway/hybrid solutions;
- Decentralized dynamic cache management oblivious to switch memory capacity.

3.1 Basic Principles

In the following, we first discuss the algorithm assuming that mappings do not change, and then explain how we deal with updates in §3.3. SwitchV2P comprises a set of in-network caches that operate within the data plane, e.g., in the switches. We assume that the system uses gateway-driven or hybrid designs where the translations are initially stored in the V2P translation gateways. As in prior works [56], we assume that the switches know the addresses of these gateways, e.g., by fixing their addresses to remain constant even if the gateways change or move.

An unresolved packet for which the destination IP mapping is unknown is sent to the gateway as before. Lookup and translation are performed opportunistically in switches along the network path that a packet takes. If the mapping is not found in any switch along the path, the process is identical to the gateway design: the gateway performs the V2P translation and forwards the packet to its destination. If the packet is resolved by a switch from the local cache, it is then forwarded to the respective physical destination without reaching the gateway. The basic idea resembles MAC learning in L2 networks but with one key distinction: in the event of a cache miss, the packet is forwarded directly to the gateway without flooding the network.

There are two basic primitives executed by each switch:
Lookup. For each unresolved packet a switch parses the virtual destination IP from the inner packet header and uses it as a lookup key for its local cache. A successful lookup operation returns the corresponding physical destination IP. The switch can then overwrite the destination IP in the outer packet header with the returned address. Thus, packets are forwarded directly to their true destinations without reaching the gateway.
Learning. A switch learns V2P mappings from the passing packets and inserts them into the cache. It inspects two virtual-physical IP pairs: source and destination. Learning mappings from the source, *source learning*, is always possible because the source physical IP is known to the sender, whereas learning from the destination, *destination learning*, is performed for packets whose destination has already been translated by a gateway or a previously traversed switch that cached the mapping. A switch may choose to perform source learning, destination learning, or both.

We are left with one key design question: what should be the *cache admission policy* that determines which entries to install into the cache?
Local greedy approach. To gain intuition, we start with the "strawman" *local greedy* solution: each switch performs destination learning and admits all insertions. We then show that this approach yields unsatisfactory results.

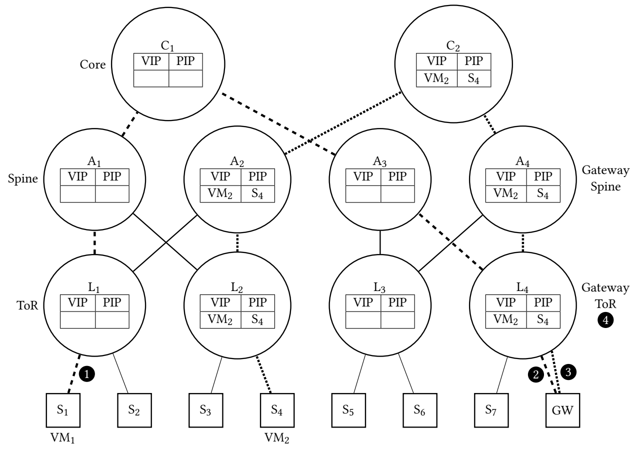


Figure 3: The sequence for packet forwarding in the local greedy approach. VM_1 sends a packet to VM_2. The packet is sent to the gateway (GW, shown by dashed lines), which determines VM_2's location. The packet is then forwarded by the gateway to VM_2 (shown by dotted lines). The V2P mapping is cached on every switch along that path.

Figure 3 shows an example of the system behavior. All caches start empty. The cache of each switch is shown as tables in Figure 3, where virtual IP addresses (VIPs) are mapped to the respective physical IP addresses (PIPs). When VM_1 sends a packet to VM_2, the packet is routed to the gateway (step ❶). Each switch along the path to the gateway performs a local lookup using the VIP address of VM_2. Since the cache is empty, all these lookups fail, and the packet eventually reaches the gateway (step ❷). The gateway resolves the address and forwards the packet to VM_2 (step ❸). Switches on the path to VM_2, and in particular, the gateway ToR switch, can learn the V2P mapping of VM_2. Therefore, if VM_1 sends another packet to VM_2, it will be directed to the cache on the gateway ToR switch (step ❹), L_4, and will be forwarded directly to VM_2 without going through the gateway again.
Analysis. There are several problems with this approach. First, VM_2 mapping is replicated across the switches on the path from the gateway to VM_2, but *none* of them except for the gateway ToR's cache (L_4) is on the path from VM_1 to the gateway. As a result, the cached entry is not used, the precious cache space in the switches is wasted, and the following packets from VM_1 to VM_2 will miss again. At the same time, the gateway's ToR cache is likely to experience thrashing because all the packets that miss are routed to the gateway. Last, this design does not differentiate between the ToR switches and the switches higher in the topology, so the entries that could have been stored only once in the core switch will be replicated and occupy space across multiple ToR switches. The root cause of these issues is that the caches are managed via *local* greedy decisions, ignoring their location in the network topology. Instead, we seek a *collaborative* approach where switches make decisions to improve the distributed cache efficiency as a whole.

3.2 Topology-Aware Caching

Cache structure. Each switch is equipped with a direct-mapped cache [23]. It can be implemented entirely in the data plane, without

Switch Type	Admission Policy	Learning from	Special Functions
Gateway ToR	All	Destination	Learning packet generation
Gateway Spine	**A** bit clear	Destination	
ToR	All	Source Learning packets	Invalidation packet generation
Spine	**A** bit clear	Destination	Promotions
Core	**A** bit clear	Promotions	

Table 1: The features of each switch type in SwitchV2P.

requiring packet mirroring, recirculation, or control plane involvement. Each cache entry includes a key (VIP), a value (PIP), and an access (A) bit turned on upon a hit. The access bit is turned off when a lookup ends up accessing that cache line but it is a miss. We use a single bit instead of sketches to indicate in-use entries, thereby making more memory available for additional mapping entries. All switches share the same cache structure.

Switch categories. We classify the switches into five categories based on their location in the network: gateway ToRs, and gateway spines; and regular ToRs, spines, and core switches. In the topology in Figure 3, L_4 is the only gateway ToR switch, as it is directly connected to the gateway. A_3 and A_4 function as gateway spines due to their direct attachment to a gateway ToR switch. Lastly, L_1, L_2, and L_3 are ToRs, while A_1 and A_2 are spines. C_1 and C_2 are core switches.

Each category differs in terms of the cache admission policy and certain additional functions performed by the respective switches. Together, the switches collaboratively strive to maximize the hit rate in the entire distributed cache while adhering to the in-switch memory capacity constraints.

Table 1 summarizes the admission policies and additional functions performed by switches in each category. We now describe them in detail.

3.2.1 *Admission policies.*

Gateway ToRs employ only destination learning because the packets on their path to the gateway have already traversed multiple switches, and most likely have their source VIP:PIP already cached.

Other ToRs perform source learning because it is likely that the receiver in the ToR's rack will send a response, i.e., TCP ACK. They also learn from special *learning packets* as discussed below.

Spines perform destination learning. However, as they process traffic from the entire pod or network, they conservatively avoid admitting new entries if it would require evicting an actively used one. The reason is that a new entry pending insertion is not guaranteed to be useful, as it could have been traveling off the network path where it is needed, whereas an active entry in the cache is known to be useful. The status of the existing entry is determined via the access bit.

Cores do not learn from the regular traffic packets as this would result in thrashing. Instead, they learn from *entry promotions* they receive from Spines, as discussed below. They cache an entry only if the entry to be evicted has its access bit cleared.

3.2.2 *Special functions.*

Learning packets. Gateway ToRs serve as a dissemination point of the learned mappings. They are exposed to all the packets that enter and leave the gateways from the whole network. Therefore, they strive to proactively move mapping closer to the sender. To this end, when they learn a new V2P mapping, they generate a packet to forward the mapping to the sender's ToR switch. To avoid excessive packet generation, however, learning packets are generated at a predefined probability P_{learn}. Consequently, the maximum bandwidth requirement for learning packets is at most $100 \times P_{\text{learn}}\%$ of the overall switch bandwidth.

Cache spillover. Switches strive to opportunistically keep mappings within the network. They append evicted entries to processed packets,[2] which subsequent switches on the route attempt to insert locally. This helps increase the effective cache capacity in the network.

Promotion of popular entries to cores. Spines may promote frequently used entries to the core switches. Specifically, if a packet destined for the gateway hits the Spine cache and the respective entry already has its access bit set, the mapping is appended to the packet and sent to the core switch. This allows sharing entries among multiple sources from different pods, thus eventually freeing the cache space in the Spine. This applies only to the packets that leave the pod, as otherwise, the promotion would not be effective. Promotion is not invoked in gateway spines because misses eventually all reach the gateway and promoting to a higher layer does not increase sharing.

3.2.3 *Putting it all together.*

Figure 4 illustrates the full system behavior in a sequence of scenarios below. In the description below, when we say that a switch learns VM_i, it means the VM_i's V2P mapping.

$VM_1 \rightarrow VM_2$ (Figure 4a). First, L_4, A_4, and A_2 learn VM_2 from the packets sent from the gateway to VM_2. Next, L_2 learns VM_1 via source learning on the way from the gateway to VM_2. This is useful if VM_2 later replies to VM_1. In addition, L_1 learns VM_1 via source learning, and VM_2 via a learning packet, sent at some point by L_4. Subsequent packets from VM_1 to VM_2 hit the cache at L_1 and are sent directly to VM_2.

$VM_3 \rightarrow VM_4$ (Figure 4b). L_4 learns VM_4, but as a result it evicts VM_2. Since A_3 is on the path from the gateway to VM_4, L_4 *spills* VM_2 to A_3. Additionally, L_4 sends a learning packet with VM_4 to L_2. As A_1 is on the path from VM_3 to VM_4, it learns VM_4 via destination learning. It further *promotes* VM_4 to C_1. L_2 and L_3 learn VM_3.

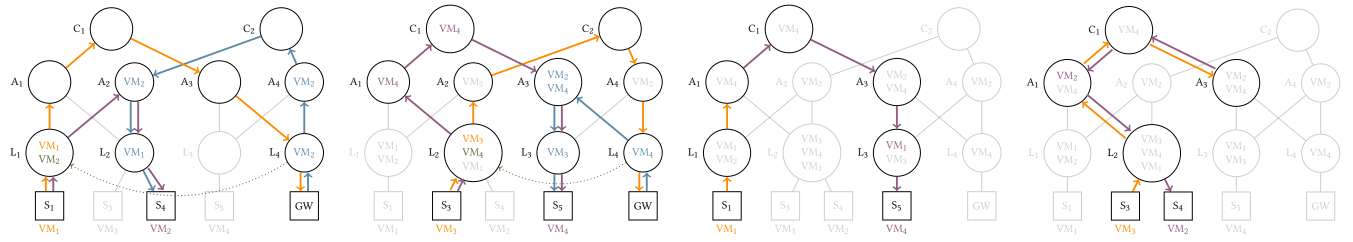
$VM_1 \rightarrow VM_4$ (Figure 4c). The packet hits the cache on A_1 and is forwarded to VM_4. VMs placed on other pods can now share that entry on C_1. L_3 learns VM_1 via source learning.

$VM_3 \rightarrow VM_2$ (Figure 4d). These packets hit the cache at A_3 on the path from VM_3 to the gateway and are directly forwarded to VM_2. A_1 learns VM_2 via destination learning.

3.3 Updates

V2P mappings are updated in the gateways, whereas the cached V2P mappings remain stale. Fortunately, we observe that the in-network cache does not have to be strongly consistent with the

[2] Using the option field in the tunnel header [19].



(a) VM$_1$→VM$_2$ **(b) VM$_3$→VM$_4$** **(c) VM$_1$→VM$_4$** **(d) VM$_3$→VM$_2$**

Figure 4: The in-network cache contents in SwitchV2P after a sequence of communications. The sender VM and the uplink path to the GW are in orange. The path from the GW to the receiver VM is in blue. Dotted lines represent generated learning packets. The direct path toward the receiver is in purple. The colors of the cached entries represent the path through which the entries are learned. Gray entries show the entries already in the cache.

"ground truth" at the gateways. Specifically, the misrouted packets can be then rerouted to the correct destination. For example, Andromeda [11] installs a follow-me rule before migrating the VM to the new machine. Thus, control-plane updates in Andromeda can occur in parallel, as the follow-me rule ensures correct packet forwarding to the VM's new destination.

Relaxing the consistency guarantees of the cache enables more efficient updates. Thus, we adopt a *lazy* cache invalidation approach, where some mappings may remain stale for a short time. The packets that are routed using these stale mappings are re-forwarded to their correct location, thereby in the end, the packets are correctly forwarded in the network.

Our approach combines piggybacking invalidation requests on misrouted packets and the use of specially crafted *invalidation packets* targeted for the caches that are known to contain stale mappings. We discuss these below.

Misdelivery tags. When a host's hypervisor receives a packet that cannot be locally delivered, it forwards such a packet to the gateway. Note that a misdelivery to the wrong VM is prevented due to the mismatch in the inner packet header. Such packets are tagged using a *misdelivery tag*, which prevents the packet from fetching a stale cached entry again on its way to the gateway.

In principle, the tag could be added by the host's hypervisor, but that would require modifying its code. Instead, we implement such a tagging in ToRs. For this purpose, ToRs keep a mapping between the front panel ports and the PIPs of the attached servers.[3] The ToR examines the packet's physical source IP, recognizing that it did not originate from one of the directly attached servers.

A switch that received a packet with a misdelivery tag either invalidates its local cached value for that VIP, or allows the packet to use the cached value since it has already learned the new PIP of the destination. Eventually, the packet is forwarded to the correct destination by the gateway or by a switch with the correct PIP.

Invalidation packets. Although packets eventually reach their correct destinations, misrouting could persist indefinitely, increasing the load on hosts. This issue arises because misrouted packets can invalidate caches only on the route from the old destination to the gateway, so the respective stale entry in all the other switches remains cached.

[3]This mapping is expected to be updated rarely since physical topologies change infrequently [47].

A strawman approach would be to flood the entire network with invalidation packets, which is inefficient. Instead, we generate invalidation packets by the ToRs that stage them with a misdelivery tag. Specifically, ToRs send invalidation packets only to the switches that are known to have stale mappings. To do so, each switch is assigned a unique identifier, which it adds to the packet header upon a hit in its local cache. When a misdelivered tag gets assigned, the ToR switch uses the switch identifier to send an invalidation packet to the specific switch.[4] This process ensures that all the caches along the path to the destination are invalidated as well.

However, this reactive approach may still result in a large burst of invalidation packets. To mitigate this, we introduce a timestamp vector with an entry for each switch in the network topology.[5] Before generating an invalidation packet for a particular switch, the ToR switch calculates the time difference between the current timestamp and the timestamp in the vector. If the time difference is less than the base RTT in the network, no packet is generated. Otherwise, the switch overwrites the vector entry with the current timestamp and generates the packet. The timestamp vector serves two purposes: it mitigates a burst of invalidation packets sent to the same switch and also allows the retransmission of an invalidation packet in case a previously generated packet is dropped in the network.

3.4 Implementation

We implement SwitchV2P as well as the main previous approaches to V2P translation on the NS3 simulator [43]. We use the IP-in-IP protocol [1] to tunnel packets. Our prototype implementation is available as open-source software in our repository at https://github.com/acsl-technion/SwitchV2P.

P4 Prototype. We validate the feasibility of SwitchV2P by prototyping it in P4$_{16}$ [50] for Intel Tofino [26] using Intel P4 Studio [25]. To implement the in-switch cache, we utilize three register arrays: one for keys, one for values, and one for access bits. Generally, our implementation does not require packet recirculation, mirroring, or multicast. We use packet mirroring to generate invalidation and learning packets.

[4]Switch IPs can be calculated directly from the switch identifier if the location of the switch is encoded within it. Alternatively, the switch identifier can be its IP, which can reduce management complexity.

[5]This approach does not require clock synchronization since the timestamp vector maintains local timestamps only.

4 Discussion

Security. SwitchV2P has no impact on network security. Tenants cannot access unauthorized mappings in the cache because they cannot spoof physical addresses. A potential concern is whether in-network caching may enable a VM to bypass ACL rules by hitting the cache. This may not happen, however, since ACLs change infrequently and are enforced in the end-hosts [11, 38, 53].

Handling dynamic caching in the host. In hybrid systems such as Andromeda, hot V2P mappings are dynamically installed in the sender's host. Clearly, caching these mappings in the network would be wasteful. This case is automatically handled in our design. SwitchV2P does not perform lookups for already resolved packets, so if an entry cached in the host is also cached in a switch, its access bit will remain unset and it will be eventually evicted.

Heterogeneous memory allocation. We have so far centered on a solution with caching in all switches in the topology. In some cases, however, different memory allocations might be beneficial. For example, allocating memory only to the ToR switches can be sufficient for achieving some of the benefits. Indeed, in our experiments, we find that using a ToR-only cache for `Hadoop` reduces the FCT but does not reduce the first packet latency. We leave finding memory allocation policies across the switches for future work.

Packet reordering and TCP. SwitchV2P may potentially introduce packet reordering. For example, this may occur if the gateway processes a burst of packets initially missing the cache, and the V2P cache is populated as a result. If subsequent packets are sent before all the packets leave the gateway, these other packets may hit the cache and arrive earlier. Indeed, in our experiments, we observed increased packet reordering in configurations with smaller cache sizes, but it is rare with larger caches. Additionally, reordering may occur due to cache invalidation. However, this is a transient effect that resolves within the time it takes for VM migration: stale entries can be evicted by background traffic or by invalidation packets within μsecond (§5.2). In contrast, the median duration of VM migration is on the order of mseconds [11].

Packet reordering is not unique to our solution, and may also happen with other V2P mechanisms, such as when packets are dropped at the gateway [56]. We found that modern TCP implementations are more resilient, e.g., Linux TCP allows up to 300 reordered packets before fast retransmission [49], which was enough to avoid any observable performance effects of reordering in our simulations. Additionally, Linux [49], Windows [13, 40], and gVisor [20] implement the TCP RACK-TLP algorithm [8], which is even more robust to packet reordering and can also work with QUIC [27, 28].

Multitenancy support. When considering a multi-tenant context featuring multiple Virtual Private Clouds (VPCs) [54], destination reuse across them is unlikely because VPCs operate in different address spaces. Therefore, SwitchV2P may serve for maintaining a per-VPC private cache in a private memory partition in a switch. As in-switch memory is a scarce resource, an operator may decide to enable SwitchV2P for a particular VPC based on a policy, e.g., when the gateway load exceeds a certain threshold. Partitioning of the switch memory among the tenants can be achieved via runtime memory allocation [51, 58]. At the same time, the in-switch cache must be isolated to avoid performance interference between the tenants. We leave a systematic solution for the multi-tenant in-network caching for future work.

Gateway migration. Changing the location of the gateway in the network would require modifying the roles of the ToR switches. The switch's role can be dynamically adjusted through a control-plane operation. Consequently, during gateway migrations, the former gateway ToR can transition to a standard ToR behavior, while the new ToR can take on the role of a gateway ToR. The cache state does not require migration; instead, it is rebuilt at the destination.

5 Evaluation

We evaluate SwitchV2P via extensive, large-scale NS3 simulations [43] with real-world network traces. Table 2 summarizes our results. We use the following baselines:

- **NoCache – pure Gateway [11]:** packets are forwarded to their destination via gateways. A gateway resolves the destination addresses and forwards the packets to the destinations. This baseline mimics the Hoverboard model in Andromeda [11] but without host offloading. However, as we explain below, our traces offer no offloading opportunities because of the 2-tuple flow reuse.
- **LocalLearning:** the simplistic design from §3.1.
- **GwCache – Sailfish [44]**: local caches are deployed only on the gateway ToRs. Other switches are not used for caching. This mimics Sailfish [44], as the caches are deployed only at the gateways. However, unlike the controller-managed cache in Sailfish, GwCache learns the mappings dynamically in the data plane.
- **Bluebird [6]:** ToR switches resolve addresses in the data plane when they are in the cache (route cache in [6]); otherwise, the control plane (SFE) forwards packets and updates the cache. We set the data to control plane bandwidth to 20Gbps, the forwarding latency of packets by the control plane to 8.5μsec, and the cache insertion latency to 2msec—similar to the parameters in the original paper [6].
- **Controller:** To establish a theoretical baseline of the cache performance, and evaluate the option of using a centralized controller for optimal caching, we devise an analytical model for the distributed cache allocation and placement by a centralized controller (see Appendix A.1 for details). The optimization problem at hand is solved via an ILP solver on the controller, given the network topology and the precise momentary traffic matrix in the whole network. The controller periodically fetches the connection matrix statistics from each switch, solves the ILP, and installs the mappings in each switch according to the solution. This configuration is evaluated only on `WebSearch` due to its high simulation cost. This is not a practical solution as it does not scale, so it serves only as a theoretical baseline.
- **OnDemand – host-driven with a first lookup in the gateway [18]:** This resembles the on-demand in VL2 [18], the Hoverboard model [11] with an immediate rule offloading policy or the ALM mechanism from Achelous [53]. Cache misses are penalized with a 40μsec latency.
- **Direct – pure host-driven [32]:** hosts are installed with all required mappings, mimicking the preprogrammed model [32]. This serves to estimate the best network performance but ignores the overheads of mapping updates.

Property	Observation
Application performance	SwitchV2P reduces FCT and first packet latency, even when a cache is small.
Updates	SwitchV2P reduces packet latency overheads and the number of misdelivered packets.
Bandwidth overheads	SwitchV2P reduces the overall number of processed bytes in the network, thus improving the effective network utilization.
Gateway resources	SwitchV2P allows reducing the number of gateways by an order of magnitude while maintaining the same application performance.
Sensitivity to topology	SwitchV2P maintains its advantages in a scale-up network topology.
Topology-aware caching	Caching in core and spine switches is essential to achieve SwitchV2P benefits.
Switch resources	SwitchV2P is lightweight and implementable with low resource consumption.

Table 2: Summary of experimental results.

Reported metrics. We focus on the cache hit rate, first packet latency, and flow completion times (FCT). The first estimates the cache effectiveness in reducing the load on the gateways, while the second and the third reflect the application performance. We also examine the per-pod and intra-pod traffic distribution to demonstrate the effect of caching on traffic hotspots near the gateways, load distribution in the network, bandwidth overheads and packet path length. Finally, we measure the costs of mapping updates.
SwitchV2P configuration. We set the frequency of learning packet generation to 0.5% of all the traffic passing through the gateway switch. In addition, all caches are empty at the beginning of each simulation.
Datasets. We use three real-world traces: `Hadoop` [46], `WebSearch` [4], and `Alibaba` [36]. Additionally, we use two synthetic UDP traces: a `Microbursts` trace (with a 99th percentile burst duration of 158μsec) [7, 30, 52, 55], and an 8K `Video` trace (with 64 senders at 48Mbps) [56, 57]. We generate packet traces according to `Hadoop` and `WebSearch` with a network load of 30% with 100Gbps links. This is similar to the network load used in the HPCC evaluation [34]. We uniformly draw sources and destinations from a pool of 10240 VMs, with 80 VMs on each server, exceeding the largest VPC size that was experimented with in Zeta [56]. For `Alibaba`, we utilize a prefix of the microservice call trace. On average, there are 32 containers on each server, for a total of 410,865 containers, as specified in the trace.
Address reuse characteristics. `Hadoop` mainly consists of short flows, and destination reuse between multiple flows is high. Each VM serves as a destination in at least one flow in this trace. 10,233 VMs serve as destinations in at least two flows with a total of 99,297 flows. The reuse distance is relatively high, with an average of 2.5 msec. `WebSearch` is mostly comprised of heavy flows with minimal cross-flow destination sharing, with only 48% of the VMs being a destination in at least one flow, and only 1,466 VMs are destinations in at least two flows. `Alibaba` consists of RPC calls, with 24% of the VMs being a destination in at least one RPC. It has high cross-flow destination reuse: over 18K VMs appear as destinations in at least 10 different flows. `Microbursts` consists of mice flows with over 2.6K VMs that appear as destinations in at least 10 different flows. `Video` consists of 64 heavy flows with no destination reuse. Note that all the 2-tuple flows in the `Hadoop` and `WebSearch` are shorter than tens of milliseconds, as none of them repeat more than twice in the trace. Thus, in the gateway-based design, these flows cannot be offloaded to the host as this number is below the threshold to offload the rule, e.g., in Zeta [56]. Additionally, `Alibaba` only includes RPC calls. Therefore, all packets must be resolved by the gateway. However, to illustrate the upper bound on the Andromeda performance we also evaluate the OnDemand policy, where all the translations are cached at the host after the first miss.

	FT8-10K	FT16-400K
#Pods	8	50
#Racks per pod	4	8
#ToR switches	32	400
#Core switches	16	16
#Gateways	40	250
#VMs	10240	410865
#Physical servers	128	12800

Table 3: The network topologies' characteristics.

Network parameters. We consider two FatTree [3] network topologies in our evaluation, summarized in Table 3. We use FT16-400K with the `Alibaba` trace [36] and place the microservices according to the information included within. We use FT8-10K with all other traces. We also experiment with other topology parameters in §5.3. We use a similar setup to the setup in HPCC [34]. In both topologies, we set the link propagation delay to 1 μsec, resulting in a 12 μsec base round-trip time (RTT). Flows are balanced among multiple paths using ECMP routing [24]. Each server is equipped with a single 100Gbps NIC, and the capacity of switch-to-switch links is 400Gbps, leading to a 4:1 oversubscription. We set the switch buffer size to 32MB.

We deploy gateways in 50% of the pods and set the gateway processing time to 40 μsec, following Sailfish [44]. The gateways are replicated, with load balancing performed by each server on a per-flow basis. We deploy an adequate number of gateways to meet the peak throughput demands of our traces. Given that we generate traces with a network load of 30%, FT8-10K and FT16-400K have 40 and 250 gateways respectively. Unless otherwise noted, we do not observe any packet drops during our tests at the gateways. We further evaluate the sensitivity to the number of gateways in §5.3.
In-switch memory size. The cache size is considered to be the aggregate memory of all the switches used for caching in the network. Inversely, the cache size per switch is $\frac{1}{\#switches}$ of the total cache. The cache size is reported relative to the total number of addresses in a given experiment.

5.1 End-to-End Benchmarks

Overview. We vary the cache size (the aggregate memory of all switches) from 1% to 1500× of the total VIP address space size in the network, based on the following reasoning: a switch has the capacity to store 192K entries in its data plane memory [6], which translates to 15M entries in total for an 80-switch topology. Considering that the VIP address space size in Hadoop and WebSearch is about 10K, this represents a 1500× ratio, which corresponds to the maximum cache size used in our experiments. However, considering the potential scaling of the number of isolated virtual networks, e.g. millions of VPCs in an Alibaba Cloud region [44], we also examine smaller cache sizes where the amount of memory available on each switch for each VPC is small.

We present the results for FT8-10K in Figures 5a to 5d, and the results for FT16-400K in Figure 6. The left graph shows the total hit rate in each system, i.e., the fraction of all sent packets that do not reach the gateways. Clearly, this is 0% for NoCache. Bluebird and Direct are omitted because they do not access the gateways, whereas OnDemand assumes infinite cache and thus its cache hit rate is not representative. The middle and right graphs show the improvement factor in the FCT and first-packet latency normalized by the results of NoCache (higher is better). SwitchV2P consistently outperforms all the techniques besides Direct across the majority of the configurations. As expected, Direct shows better latency but suffers from high update overheads, as shown in §5.2. Importantly, SwitchV2P has no negative effects on FCT or packet latency since it does not modify routing paths, so the packet routes are at most as long as in the NoCache system. Notably, SwitchV2P with a cache size of one entry per switch in FT8-10K[6], decreases the number of gateway accesses by up to 20%.

***Benefits of in-network mapping in* `Hadoop`*,* `Microbursts`*, and* `Alibaba`**. Figure 5a shows that SwitchV2P outperforms GwCache and LocalLearning for FCT for any cache size. This is because it retains more critical entries in the cache than GwCache and LocalLearning: unlike them, SwitchV2P evicts entries on one switch and inserts them into another. Additionally, SwitchV2P exploits the cross-flow destination reuse in the trace more efficiently: it resolves hits in the network switches, whereas GwCache suffers from compulsory misses in the gateway caches due to load balancing among the gateways. Cross-flow destination reuse also allows SwitchV2P to outperform the OnDemand baseline for larger caches. Finally, Bluebird drops packets due to the bandwidth-limited link between the data and control planes in the switch, so its performance is inferior to all other techniques. Similar behavior is observed in Alibaba (Figure 6) and Microbursts (Figure 5b). In Alibaba, this is primarily due to source learning at the ToRs. This trace comprises RPCs, allowing VMs to identify their destination at their ToRs when sending their responses back. Moreover, SwitchV2P leverages the large cross-flow destination reuse, significantly reducing FCT and first-packet latency.

***Benefits of moving mappings to traffic in* `WebSearch` *and* `Video`**. Figure 5c shows that SwitchV2P performs better than LocalLearning because it reduces the network path length by placing mappings closer to where they are needed. Despite the relatively high cache hit rate, first-packet latency is not significantly improved, as there is minimal destination reuse in the trace. Learning packets improve the hit rate in Video (Figure 5d). The main benefit is a significant reduction of the gateway load (presented in §5.3). However, in this trace SwitchV2P has no effect on application performance since the lookup overhead is negligible given the flow size, and there is no destination reuse in the trace.

FCT vs. cache hit rate. One might expect similar FCT in SwitchV2P and GwCache given the same hit rates, but this is not the case. For example, in Hadoop SwitchV2P achieves better FCT than GwCache even with the same cache hit rate. This is because SwitchV2P can access the cache on the upward path towards the gateway, while GwCache requires four more hops to reach the cache and exit the pod from the core switch. The latency increases as the traffic is skewed toward the gateways (more details in §5.3).

First-packet latency vs. cross-flow reuse. First-packet latency reduction shows clearly in the Hadoop, Alibaba, and Microbursts traces with significant cross-flow destination sharing. The trends in these traces are quite similar, though the magnitude of the reduction is higher in Hadoop. As expected, the other traces show no improvements in the first-packet latency due to low destination reuse.

GwCache performs slightly better than SwitchV2P for smaller cache sizes due to its larger per-switch cache size (recall that in our experiments the per-switch memory is divided equally between the switches). Since in GwCache, only four Gateway switches are used for caching, each switch is configured with 20× more memory than in SwitchV2P which uses 80. As the reuse distance is quite large (2.5 msec), a larger centralized cache in a switch performs better than a cache distributed among multiple switches. However, SwitchV2P outperforms all other baselines for larger cache sizes.

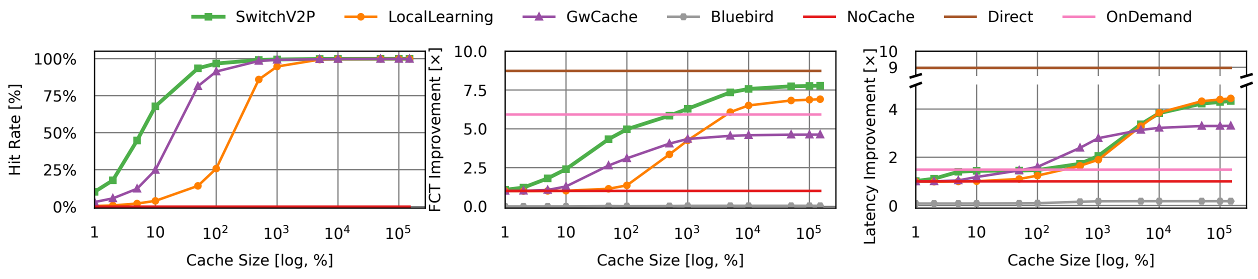
Centralized allocation via ILP. We fully analyze the Controller baseline in Appendix A.2. We conclude that it is impractical because even when it is run at a high frequency it still lacks timely information thus its benefits diminish in configurations with cache sizes larger than 50%.

5.2 VM Migration Overheads

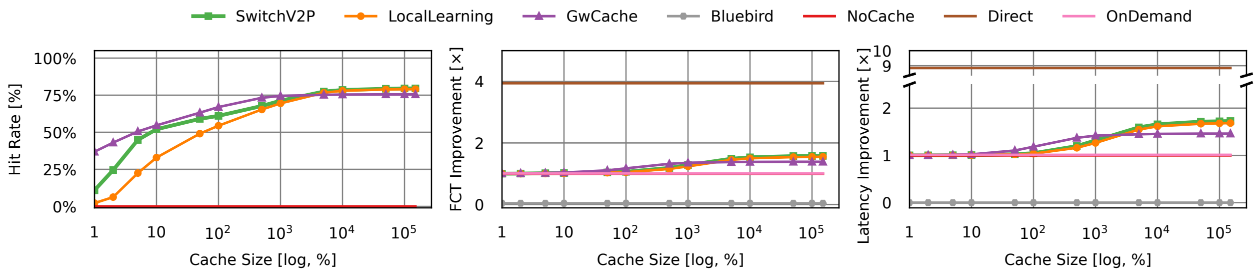
We generate a synthetic incast traffic pattern by simulating 64 UDP senders, each running on a distinct physical server in the FT8-10K topology, and sending packets to the same destination VM. Subsequently, we migrate the VM to a different rack, thus changing the physical address of the destination VM. The entire trace lasts 1 msec, totaling 64K packets. The migration occurs in the middle of the simulation, at 500μsec (simulation time). We compare several SwitchV2P variants with NoCache and OnDemand, and normalize them by the measurements of NoCache. Under NoCache and OnDemand, misdelivered packets are sent to the new destination by the old destination using a "follow-me" rule [11]. In contrast, in SwitchV2P misdelivered packets are sent from the old destination to the gateway. In both cases, we set the additional overhead of processing the packet in the old destination to 10μsec. For OnDemand, we assume the controller cannot update the mappings on the hosts within 500μsec, as the rule installment takes in the order of milliseconds [56] (this assumption also applies to Achelous [53]).

Table 4 summarizes the results. NoCache has the fewest misdelivered packets because these are merely the packets that were buffered in the gateway when the migration occurred. In contrast, in OnDemand, the migration causes many misdelivered packets and clearly affects the packet latency. A fully optimized SwitchV2P (last row) achieves OnDemand's packet latency with only 20% more

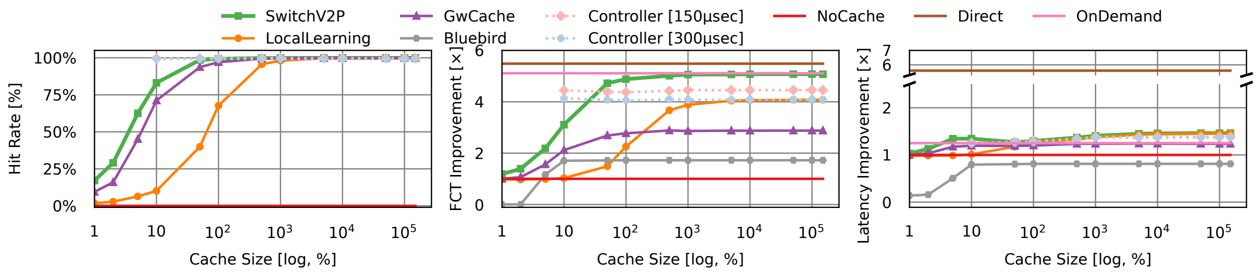
[6] 1% of 10K IPs is 100 entries, uniformly distributed across 80 switches – 1 per switch.



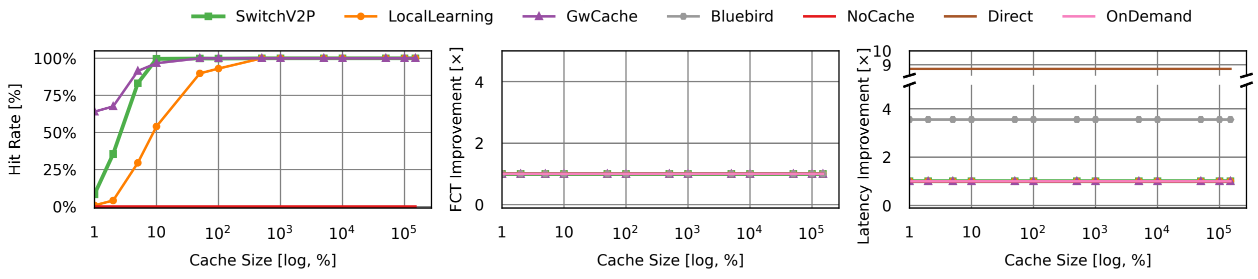
(a) Hadoop



(b) `Microbursts`



(c) `WebSearch`



(d) `Video`

Figure 5: Setup=FT8-10K: hit rate, average FCT, and first packet latency improvement (normalized by NoCache).

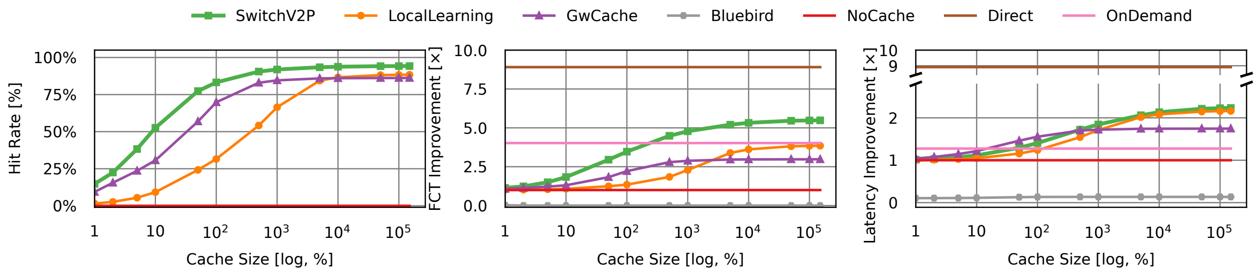


Figure 6: `Alibaba` in FT16-400K: hit rate, average FCT, and first packet latency improvement factor (normalized by NoCache).

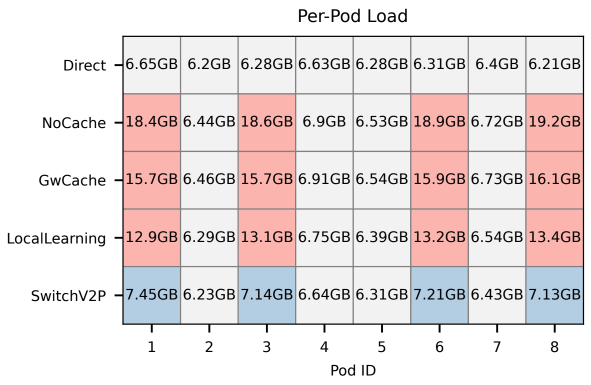


Figure 7: The number of processed bytes in each pod. Gateways are in pods 1,3,6,8.

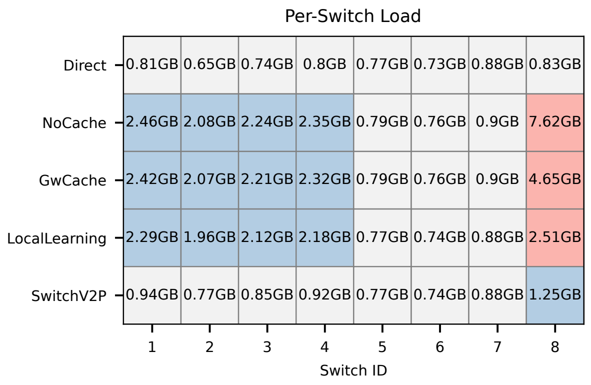


Figure 8: The number of processed bytes across switches in pod 8: spines (switches 1-4), ToRs (switches 5-7), and a gateway ToR (switch 8).

misdelivered packets than NoCache, and nearly identical total migration latency as indicated by the last misdelivered packet arrival.

Invalidation packets are essential for SwitchV2P to reduce the number of misdelivered packets. However, in the naive solution without the timestamp vector, the leaf switch sends more packets than if it had sent invalidation packets to every switch in the network (80 in this simulation). By introducing a timestamp vector, we reduce the number of invalidation packets by over 100×.

5.3 Analysis

We analyze several aspects of SwitchV2P using the Hadoop trace with a cache size of 50%.

Reduced bandwidth overheads and shorter packet stretch.

SwitchV2P is expected to reduce the traffic to the gateways, thereby alleviating network hotspots and reducing network bandwidth overheads. To analyze this, we measure the total number of bytes handled by switches in each pod (presented as a heatmap in Figure 7). Note that the byte counts are based on the total number of packets processed by each switch. Therefore, if a packet exits a pod, it is counted twice: once by the ToR switch and once by the spine switch.

SwitchV2P significantly reduces the load on the gateway pods compared to all other baselines. Although GwCache and SwitchV2P achieve similar cache hit rates, in SwitchV2P packets hit the cache without entering gateway pods, thus reducing the load. We validate this result by measuring the total number of bytes that are handled by the switches in one of the pods that host a gateway (Figure 8 shows the eighth pod). The heatmap clearly shows that SwitchV2P significantly reduces the processed traffic in the pod. Compared to NoCache, and GwCache it reduces the amount of network traffic at the gateway switch (switch number 8) by a factor of 6.1× and 3.7× respectively.

SwitchV2P introduces merely 7% more traffic to the network compared to the Direct baseline which does not perform mapping lookups. Compared to NoCache and GwCache, it reduces the amount of network traffic by a factor of 1.9× and 1.7× respectively. Additionally, SwitchV2P significantly reduces the average packet stretch, i.e. the number of switches traversed by a packet, dropping from 9.4, 8.9, and 8.5 for NoCache, LocalLearning, and GwCache down to 5.1.

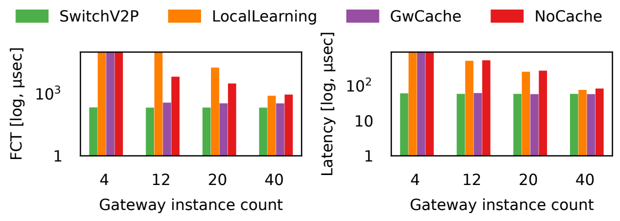


Figure 9: Performance with fewer gateways for Hadoop.

Same performance with fewer gateways. Reducing the gateway load allows for a decrease in the number of gateways in the system, thereby lowering overall costs. To evaluate this, we measure the performance while varying the number of gateways from 40 to 4.

Figure 9 shows that even with 10× fewer gateway instances, SwitchV2P achieves nearly the same FCT and first packet latency (up to a 3% reduction). In contrast, NoCache and LocalLearning are not as effective. As the number of deployed gateways decreases, their FCT and first packet latency increase. With only 4 deployed gateways we observe packet drops, causing the latency metrics go off the scale.

Topology scaling. We modify the topology by adjusting the number of servers in each rack, resulting in a varying number of pods. Specifically, we increase the number of pods from 8 to 32 by reducing the number of servers in each rack and decrease the number of pods to 1 by increasing the number of servers in each rack up to 32.

Figure 10 shows that SwitchV2P scales better with the topology size compared to LocalLearning. Specifically, SwitchV2P achieves lower FCTs as the topology size increases, whereas LocalLearning faces challenges in disseminating learned information to the appropriate locations for very large topologies. GwCache is stable across all configurations because the gateway count remains consistent across topology sizes, ensuring a constant per-switch cache size.

Cache hit distribution. We analyze the location of cache hits within the topology. Table 5 provides a summary of cache hit rates for the complete trace and the first packets, segmented based on the switch hierarchy in the topology. First packets hit the cache in the upper layers of the topology, leveraging cross-flow address reuse. In the TCP traces, the majority of cache hits occur in ToR switches, largely due to generated learning packets and source

	Gateway Packets	Avg Packet Latency	Last Misdelivered Packet Arrival [μsec]	Misdelivered Packets	Total Invalidation Packets
NoCache	100%	1×	545	1×	0
OnDemand	0%	0.25×	1005	11×	0
SwitchV2P w/o invalidations	8.4%	0.31×	1005	5.9×	0
SwitchV2P w/o timestamp vector	8.7%	0.25×	563	1.2×	3503
SwitchV2P w/ timestamp vector	8.7%	0.25×	563	1.2×	24

Table 4: The effect of VM migration on network performance, normalized by NoCache.

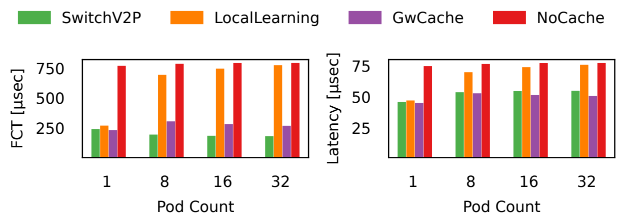


Figure 10: Topology scaling results for the Hadoop trace in a scaled FT8-10K topology.

Dataset	Total			First packet		
	Core	Spine	ToR	Core	Spine	ToR
Hadoop	1.7%	6.6%	91.7%	30.1%	29.2%	40.7%
WebSearch	1.5%	4.4%	94.1%	20.7%	33.8%	45.5%
Alibaba	2.4%	2.0%	95.6%	25.4%	15.0%	59.6%
Microbursts	6.8%	16.1%	77.1%	28.9%	30.3%	40.8%
Video	16.3%	12.2%	71.5%	0%	0%	0%

Table 5: The distribution of SwitchV2P cache hits within the network topology for each dataset at a cache size of 50%.

learning. In contrast, in the UDP traces, such as the Microbursts trace, about a third of the packets hit the cache at the core and spine switches due to better cache placement of frequently used destinations. These results highlight the significance of every switch in the topology in contributing to the performance advantages delivered by SwitchV2P.

Switch resource utilization. Table 6 summarizes the average per-stage resource utilization for the 50% cache size configuration. SwitchV2P leaves plenty of resources to implement other functionality on the switch. Moreover, Hash Bits and SRAM utilization are the only components that increase proportionally as the cache size is expanded. The gateway utilization can be further decreased by replacing a few if-else clauses with a table with ternary keys.

6 Related Work

V2P translation in virtual networks. Numerous projects aim to optimize V2P translations in virtual networks [6, 11, 18, 44, 53, 56]. SwitchV2P sets itself apart by proposing a new design point in the tradeoff between the routing overheads and V2P mapping update cost. A recent industry protocol from Cisco [9] takes a first step toward this vision and suggests data plane address learning. However, this protocol requires keeping all the mappings in spine switches and is specifically designed for Clos topologies.

Resource	Utilization
Match Crossbar	7.2%
Meter ALU	17.5%
Gateway	25.0%
SRAM	3.9%
TCAM	1.7%
VLIW Instruction	10.0%
Hash Bits	4.7%

Table 6: The average per-stage resource utilization of a cache size of 50%.

Virtual networks in data centers. Virtual network architecture and optimization techniques, in both software and specialized hardware, have been extensively studied [10, 14, 16, 21, 32, 33, 42, 45, 48]. SwitchV2P aims to enhance virtual networks as well, utilizing a distributed in-network mapping cache to improve packet forwarding.

In-network caching. NetCache [29] and DistCache [35] demonstrate the benefits of in-network caching in the context of key-value stores. DistCache further considers scaling beyond a single rack and presents a distributed design. However, the design considerations for DistCache are completely different, as it requires routing changes and, as a key-value store, needs to keep cache coherency.

7 Conclusions

SwitchV2P is a distributed, in-network caching system that enables switches to cache V2P mappings for virtual networks. State-of-the-art solutions either use gateway- or host-driven designs and suffer from an inherent tradeoff between mapping update and lookup performance. SwitchV2P, in contrast, aims to escape this inherent tradeoff by a novel design where in-network caches are hosted on switches and self-govern using a distributed protocol. It requires no changes to the gateways or servers while delivering significant performance benefits.

Ethics. This work does not raise any ethical issues.

Acknowledgments

We thank the anonymous reviewers and our shepherd, Li Chen, for their help in shaping the final version of this paper. Lior Zeno was partially supported by the HPI-Technion Research School. We gratefully acknowledge support from Israel Science Foundation (Grant 1998/22), VMware Early Career Faculty Grant, as well as NSF grants ITE-2226339, CNS-1942219, CNS-2106751, CNS-2107147, CNS-2214272.

References

[1] 1995. IP in IP Tunneling. RFC 1853. https://doi.org/10.17487/RFC1853

[2] 2024. Google Containers. https://cloud.google.com/containers.

[3] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A Scalable, Commodity Data Center Network Architecture. *ACM SIGCOMM computer communication review* 38, 4 (2008), 63–74.

[4] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data Center TCP (DCTCP). In *Proceedings of the ACM SIGCOMM 2010 Conference*. 63–74.

[5] Amazon. 2024. Amazon Virtual Private Cloud. https://aws.amazon.com/vpc.

[6] Manikandan Arumugam, Deepak Bansal, Navdeep Bhatia, James Boerner, Simon Capper, Changhoon Kim, Sarah McClure, Neeraj Motwani, Ranga Narasimhan, Urvish Panchal, et al. 2022. Bluebird: High-performance SDN for Bare-metal Cloud Services. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association.

[7] Xiaoqi Chen, Shir Landau Feibish, Yaron Koral, Jennifer Rexford, and Ori Rottenstreich. 2018. Catching the Microburst Culprits with Snappy. In *Proceedings of the Afternoon Workshop on Self-Driving Networks* (Budapest, Hungary) *(SelfDN 2018)*. Association for Computing Machinery, New York, NY, USA, 22–28. https://doi.org/10.1145/3229584.3229586

[8] Yuchung Cheng, Neal Cardwell, Nandita Dukkipati, and Priyaranjan Jha. 2021. The RACK-TLP Loss Detection Algorithm for TCP. RFC 8985. https://doi.org/10.17487/RFC8985

[9] Cisco. 2024. ACI Fabric Endpoint Learning White Paper. https://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-739989.html.

[10] Andrew R. Curtis, Jeffrey C. Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. 2011. DevoFlow: Scaling Flow Management for High-Performance Networks. In *Proceedings of the ACM SIGCOMM 2011 Conference*. 254–265.

[11] Michael Dalton, David Schultz, Jacob Adriaens, Ahsan Arefin, Anshuman Gupta, Brian Fahs, Dima Rubinstein, Enrique Cauich Zermeno, Erik Rubow, James Alexander Docauer, et al. 2018. Andromeda: Performance, Isolation, and Velocity at Scale in Cloud Network Virtualization. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 373–387.

[12] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems: 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings 14*. Springer, 337–340.

[13] Dropbox. 2021. Boosting Dropbox upload speed and improving Windows' TCP stack. https://dropbox.tech/infrastructure/boosting-dropbox-upload-speed.

[14] Andrew D. Ferguson, Steve Gribble, Chi-Yao Hong, Charles Killian, Waqar Mohsin, Henrik Muehe, Joon Ong, Leon Poutievski, Arjun Singh, Lorenzo Vicisano, Richard Alimi, Shawn Shuoshuo Chen, Mike Conley, Subhasree Mandal, Karthik Nagaraj, Kondapa Naidu Bollineni, Amr Sabaa, Shidong Zhang, Min Zhu, and Amin Vahdat. 2021. Orion: Google's Software-Defined Networking Control Plane. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. 83–98.

[15] Daniel Firestone. 2017. VFP: A Virtual Switch Platform for Host SDN in the Public Cloud. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 315–328.

[16] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert Greenberg. 2018. Azure Accelerated Networking: SmartNICs in the Public Cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 51–66.

[17] Google. 2024. Virtual Private Cloud (VPC). https://cloud.google.com/vpc.

[18] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. 2009. VL2: A Scalable and Flexible Data Center Network. In *Proceedings of ACM SIGCOMM 2009*. ACM, Barcelona, Spain.

[19] Jesse Gross, Ilango Ganga, and T. Sridhar. 2020. Geneve: Generic Network Virtualization Encapsulation. RFC 8926. https://doi.org/10.17487/RFC8926

[20] gVisor. 2021. gVisor RACK. https://gvisor.dev/blog/2021/08/31/gvisor-rack/.

[21] Soheil Hassas Yeganeh and Yashar Ganjali. 2012. Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications. In *Proceedings of the first workshop on Hot topics in software defined networks*. 19–24.

[22] Victor Heorhiadi, Michael K. Reiter, and Vyas Sekar. 2016. Simplifying Software-Defined Network Optimization Using SOL. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. 223–237.

[23] Mark D. Hill. 1988. A Case for Direct-Mapped Caches. *Computer* 21, 12 (1988), 25–40.

[24] Christian Hopps. 2000. Analysis of an Equal-Cost Multi-Path Algorithm. RFC 2992. https://doi.org/10.17487/RFC2992

[25] Intel. 2024. P4 Studio. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/p4-suite/p4-studio.html.

[26] Intel. 2024. Tofino. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series/tofino.html.

[27] Jana Iyengar and Ian Swett. 2021. QUIC Loss Detection and Congestion Control. RFC 9002. https://doi.org/10.17487/RFC9002

[28] Jana Iyengar and Martin Thomson. 2021. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000. https://doi.org/10.17487/RFC9000

[29] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. 2017. NetCache: Balancing Key-Value Stores with Fast In-Network Caching. In *Proceedings of the 26th Symposium on Operating Systems Principles* (Shanghai, China) *(SOSP '17)*. Association for Computing Machinery, New York, NY, USA, 121–136.

[30] Raj Joshi, Ting Qu, Mun Choon Chan, Ben Leong, and Boon Thau Loo. 2018. BurstRadar: Practical Real-time Microburst Monitoring for Datacenter Networks. In *Proceedings of the 9th Asia-Pacific Workshop on Systems*. 1–8.

[31] Changhoon Kim, Matthew Caesar, and Jennifer Rexford. 2008. Floodless in SEATTLE: A Scalable Ethernet Architecture for Large Enterprises. *ACM SIGCOMM Computer Communication Review* 38, 4 (2008), 3–14.

[32] Teemu Koponen, Keith Amidon, Peter Balland, Martin Casado, Anupam Chanda, Bryan Fulton, Igor Ganichev, Jesse Gross, Paul Ingram, Ethan Jackson, Andrew Lambeth, Romain Lenglet, Shih-Hao Li, Amar Padmanabhan, Justin Pettit, Ben Pfaff, Rajiv Ramanathan, Scott Shenker, Alan Shieh, Jeremy Stribling, Pankaj Thakkar, Dan Wendlandt, Alexander Yip, and Ronghua Zhang. 2014. Network Virtualization in Multi-tenant Datacenters. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. 203–216.

[33] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, and Scott Shenker. 2010. Onix: A Distributed Control Platform for Large-scale Production Networks. In *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*. USENIX Association, Vancouver, BC.

[34] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. 2019. HPCC: High Precision Congestion Control. In *Proceedings of the ACM Special Interest Group on Data Communication* (Beijing, China) *(SIGCOMM '19)*. Association for Computing Machinery, New York, NY, USA, 44–58.

[35] Zaoxing Liu, Zhihao Bai, Zhenming Liu, Xiaozhou Li, Changhoon Kim, Vladimir Braverman, Xin Jin, and Ion Stoica. 2019. DistCache: Provable Load Balancing for Large-Scale Storage Systems with Distributed Caching. In *Proceedings of the 17th USENIX Conference on File and Storage Technologies* (Boston, MA, USA) *(FAST'19)*. USENIX Association, USA, 143–157.

[36] Shutian Luo, Huanle Xu, Chengzhi Lu, Kejiang Ye, Guoyao Xu, Liping Zhang, Yu Ding, Jian He, and Chengzhong Xu. 2021. Characterizing Microservice Dependency and Performance: Alibaba Trace Analysis. In *Proceedings of the ACM Symposium on Cloud Computing*. 412–426.

[37] Filipe Manco, Costin Lupu, Florian Schmidt, Jose Mendes, Simon Kuenzer, Sumit Sati, Kenichi Yasukata, Costin Raiciu, and Felipe Huici. 2017. My VM is Lighter (and Safer) than your Container. In *Proceedings of the 26th Symposium on Operating Systems Principles*. 218–233.

[38] Michael Marty, Marc de Kruijf, Jacob Adriaens, Christopher Alfeld, Sean Bauer, Carlo Contavalli, Mike Dalton, Nandita Dukkipati, William C. Evans, Steve Gribble, Nicholas Kidd, Roman Kononov, Gautam Kumar, Carl Mauer, Emily Musick, Lena Olson, Mike Ryan, Erik Rubow, Kevin Springborn, Paul Turner, Valas Valancius, Xi Wang, and Amin Vahdat. 2019. Snap: a Microkernel Approach to Host Networking. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. 399–413.

[39] Alberto Medina, Nina Taft, Kave Salamatian, Supratik Bhattacharyya, and Christophe Diot. 2002. Traffic Matrix Estimation: Existing Techniques and New Directions. *ACM SIGCOMM Computer Communication Review* 32, 4 (2002), 161–174.

[40] Microsoft. 2021. Algorithmic improvements boost TCP performance on the Internet . https://techcommunity.microsoft.com/t5/networking-blog/algorithmic-improvements-boost-tcp-performance-on-the-internet/ba-p/2347061.

[41] Microsoft. 2024. Azure Virtual Network. https://azure.microsoft.com/en-us/products/virtual-network.

[42] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. 2009. PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*. 39–50.

[43] nsnam. 2024. Network Simulator 3. https://www.nsnam.org/.

[44] Tian Pan, Nianbing Yu, Chenhao Jia, Jianwen Pi, Liang Xu, Yisong Qiao, Zhiguo Li, Kun Liu, Jie Lu, Jianyuan Lu, Enge Song, Jiao Zhang, Tao Huang, and Shunmin Zhu. 2021. Sailfish: Accelerating Cloud-Scale Multi-Tenant Multi-Service Gateways with Programmable Switches. In *Proceedings of the ACM SIGCOMM 2021 Conference* (Virtual Event, USA) *(SIGCOMM '21)*. Association for Computing

Machinery, New York, NY, USA, 194–206.

[45] Andrew Putnam, Adrian M. Caulfield, Eric S. Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, Michael Haselman, Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, James Larus, Eric Peterson, Simon Pope, Aaron Smith, Jason Thong, Phillip Yi Xiao, and Doug Burger. 2016. A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services. *Commun. ACM* 59, 11 (oct 2016), 114–122.

[46] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. 2015. Inside the Social Network's (Datacenter) Network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. 123–137.

[47] Alan Shieh, Srikanth Kandula, Albert Greenberg, Changhoon Kim, and Bikas Saha. 2011. Sharing the Data Center Network. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*.

[48] Yu-Wei Eric Sung, Xiaozheng Tie, Starsky HY Wong, and Hongyi Zeng. 2016. Robotron: Top-down Network Management at Facebook Scale. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 426–439.

[49] The Linux Kernel Organization. 2020. ip-sysctl. https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt.

[50] The P4 Language Consortium. 2023. P4$_{16}$ Language Specification. https://staging.p4.org/p4-spec/docs/P4-16-v1.2.4.html.

[51] Tao Wang, Hang Zhu, Fabian Ruffy, Xin Jin, Anirudh Sivaraman, Dan R. K. Ports, and Aurojit Panda. 2020. Multitenancy for Fast and Programmable Networks in the Cloud. In *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*.

[52] Weitao Wang, Xinyu Crystal Wu, Praveen Tammana, Ang Chen, and T. S. Eugene Ng. 2022. Closed-loop Network Performance Monitoring and Diagnosis with SpiderMon. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 267–285.

[53] Chengkun Wei, Xing Li, Ye Yang, Xiaochong Jiang, Tianyu Xu, Bowen Yang, Taotao Wu, Chao Xu, Yilong Lv, Haifeng Gao, Zhentao Zhang, Zikang Chen, Zeke Wang, Zihui Zhang, Shunmin Zhu, and Wenzhi Chen. 2023. Achelous: Enabling Programmability, Elasticity, and Reliability in Hyperscale Cloud Networks. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 769–782.

[54] Timothy Wood, Prashant Shenoy, Alexandre Gerber, Jacobus Van der Merwe, and K.K. Ramakrishnan. 2009. The Case for Enterprise-Ready Virtual Private Clouds. In *Workshop on Hot Topics in Cloud Computing (HotCloud 09)*. USENIX Association, San Diego, CA.

[55] Qiao Zhang, Vincent Liu, Hongyi Zeng, and Arvind Krishnamurthy. 2017. High-Resolution Measurement of Data Center Microbursts. In *Proceedings of the 2017 Internet Measurement Conference*. 78–85.

[56] Qianyu Zhang, Gongming Zhao, Hongli Xu, Zhuolong Yu, Liguang Xie, Yangming Zhao, Chunming Qiao, Ying Xiong, and Liusheng Huang. 2022. Zeta: A Scalable and Robust East-West Communication Framework in Large-Scale Clouds. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 1231–1248.

[57] Wenxiao Zhang, Feng Qian, Bo Han, and Pan Hui. 2021. DeepVista: 16K Panoramic Cinema on Your Mobile Device. In *Proceedings of the Web Conference 2021*. 2232–2244.

[58] Hang Zhu, Tao Wang, Yi Hong, Dan R. K. Ports, Anirudh Sivaraman, and Xin Jin. 2022. NetVRM: Virtual Register Memory for Programmable Networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 155–170.

A Appendix

Appendices are supporting material that has not been peer-reviewed.

A.1 Distributed Cache Allocation Model

The utility function is defined as per-packet latency, and our objective is to determine the online optimal solution for the following problem: given the network topology and present traffic distribution, and assuming unchanged routing, what is the ideal cache allocation that achieves the minimum per-packet latency? This pertains to the identification of the V2P mappings to be cached and the corresponding switches on which to execute caching. We take a similar approach to traffic engineering [22] and formulate the problem as an integer linear programming (ILP) optimization.

We start by creating a topology $G = (V_n \cup V_s, E)$, where V_n is the set of routable entities, V_s is the set of switches, and E is the link set. We define $N = |V_n|$, and $S = |V_s|$. We then define the traffic matrix $T = (T_{i,j})$, where $T_{i,j}$ denotes the number of packets transmitted to node $v_j \in V_n$ by node $v_i \in V_n$.[7] Additionally, we assume that the cost of gateway processing is C and that each switch can maintain up to M V2P mappings. We define a binary variable $K_{j,i}$ for each VM, $v_i \in V_n$, and each switch, $v_j \in V_s$. This variable indicates whether the V2P mapping of v_i is stored on v_j.

The latency of each packet, $L_{i,j}$, depends on whether the mapping required is stored on any of the switches along the route to the gateway. Therefore, we can examine the path from v_i to the gateway, i.e., the group of switches. If the bit $K_{s,j}$ is activated on any switch in the path, then the packet latency is the sum of the number of hops to that switch and the number of hops from that switch to v_j. Conversely, if $K_{s,j}$ is not set on any switch along the path, then the packet latency is the sum of the number of hops to the gateway, plus C, and the number of hops from the gateway to v_j.

We can then define the following optimization problem with integer variables:

$$\text{minimize:} \sum_{i,j \in [N]} L_{i,j} T_{i,j} \quad \text{subject to:} \sum_{s \in [S], j \in [N]} K_{j,i} \leq M, K_{i,j} \in \{0,1\}$$

Note that we assume that we have advance knowledge of the path packets will follow to reach the gateway, and that we can generate a precise traffic matrix during runtime.

We employed the Z3 solver [12] and applied the formula to several straightforward topologies and traffic matrices. The optimal solutions lead us to several observations about the practical distributed caching algorithm. This algorithm should pursue two primary objectives: (1) reduce the occurrence of cache misses, which will lead to a reduction in packet latency. (2) "move mappings to the traffic", i.e. relocate the mappings closer to the sender's host so that packets can utilize the cache during their upward journey and avoid the additional hops required to reach the gateway pod altogether.

We also noted the critical importance of placing a mapping in the switches at the intersection of all network paths utilizing it, aiming to minimize entry duplication across switches. In scenarios where every VM sends packets to the same destination, such as in an incast scenario, the relevant intersection would be the gateway's ToR switch.

The solution to the optimization problem may not be practical, as we show in §5.1; however, it still holds value to us, as the insights gleaned from this approach are instrumental in constructing SwitchV2P.

A.2 Centralized Allocation via ILP

We consider the centralized approach to cache allocation by solving the integer program described in Section A.1. We provide the most favorable environment to the Controller and assume unlimited resources to build an exact traffic matrix. The Controller periodically halts the network traffic, collects the statistics from switches, builds the traffic matrix, solves the ILP, and inserts the forwarding rules into the switches. We evaluate two Controller invocation frequencies: every 150 μsec and 300 μsec (in simulation time). These settings are apparently impractical, so the Controller experiment primarily serves us as a sanity check.

Figure 5c shows the results. Some points are missing from the graph as the solver did not finish within 30 hours. For cache sizes below 50% of the address space, the Controller outperforms all other approaches. The Controller has a full picture of the traffic and thus performs better entry placement. It also uses switch memory more efficiently, i.e., avoids entry duplication across switches, exactly when it matters the most. However, as the cache size increases, these benefits are outweighed by the lack of timely information, as the mapping allocation is performed with respect to the *outdated* traffic pattern which rapidly changes. This effect becomes more pronounced when the invocation rate is 300μsec.

[7]This definition deviates slightly from the conventional definition [39], as we are concerned with packet-level counts rather than the overall volume of communication.