The Ties that un-Bind: Decoupling IP from web services and sockets for robust addressing agility at CDN-scale

Marwan Fayed[†], Lorenz Bauer[†], Vasileios Giotsas[†], Sami Kerola[†],
Marek Majkowski[†], Pavel Odinstov[†], Jakub Sitnicki[†], Taejoong Chung[⋆],
Dave Levin[‡], Alan Mislove[ø], Christopher A. Wood[†], Nick Sullivan[†]
[†] Cloudflare, Inc. [⋆] Virginia Tech [‡] University of Maryland [ø] Northeastern University

ABSTRACT

The couplings between IP addresses, names of content or services, and socket interfaces, are too tight. This impedes system manageability, growth, and overall provisioning. In turn, large-scale content providers are forced to use staggering numbers of addresses, ultimately leading to address exhaustion (IPv4) and inefficiency (IPv6).

In this paper, we revisit IP bindings, entirely. We attempt to evolve addressing conventions by decoupling IP in DNS and from network sockets. Alongside technologies such as SNI and ECMP, a new architecture emerges that "unbinds" IP from services and servers, thereby returning IP's role to merely that of reachability. The architecture is under evaluation at a major CDN in multiple datacenters. We show that addresses can be generated randomly *per-query*, for 20M+ domains and services, from as few as ~4K addresses, 256 addresses, and even *one* IP address. We explain why this approach is transparent to routing, L4/L7 load-balancers, distributed caching, and all surrounding systems – and is *highly desirable*. Our experience suggests that many network-oriented systems and services (e.g., route leak mitigation, denial of service, measurement) could be improved, and new ones designed, if built with addressing agility.

CCS CONCEPTS

• **Networks** → **Network design principles**; **Network services**; **Network manageability**; **Naming and addressing**.

KEYWORDS

addressing, provisioning, content distribution, programmable sockets

ACM Reference Format:
Marwan Fayed, Lorenz Bauer, Vasileios Giotsas, Sami Kerola, Marek Majkowski, Pavel Odinstov, Jakub Sitnicki, Taejoong Chung, Dave Levin, Alan Mislove, Christopher A. Wood, Nick Sullivan. 2021. The Ties that un-Bind: Decoupling IP from web services and sockets for robust addressing agility at CDN-scale. In *ACM SIGCOMM 2021 Conference (SIGCOMM '21), August 23–28, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3452296.3472922

SIGCOMM '21, August 23–28, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8383-7/21/08…$15.00
https://doi.org/10.1145/3452296.3472922

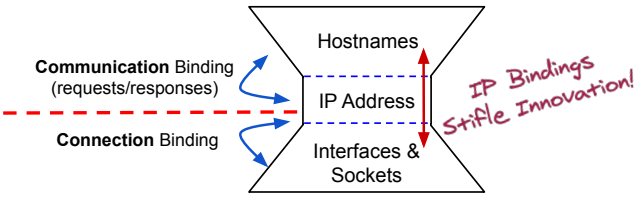


Figure 1: Conventional IP bindings to names, interfaces, and sockets, create transitive relationships between them that are difficult to track and reason about, which hinders changes to any binding without risking others.

1 INTRODUCTION

Unlike with most clients and carriers, limitations exist on hosting- and content-service providers (CDNs) because of decades-old conventions that tie *IP addresses* to *resources*. Hostnames and domains are typically mapped to a set of IP addresses. DNS will lookup and return any IP in the set to load-balance or to geo-select [9, 56] services, but the set itself is static and unchanging. Similarly, network interfaces and sockets are mapped to single IP addresses and address-port pairs, respectively [70]; once assigned to interfaces and sockets, addresses are also typically static and unchanging.

This legacy of IP-to-name and IP-to-server bindings persist in IPv6, and have in turn created a perception that possessing a large number of IP addresses is a *necessity* to operate large-scale CDN services. Indeed, large CDNs have acquired a massive number of IP addresses: At the time of this writing, Cloudflare has 1.7M IPv4 addresses [12], Akamai has 12M [1], and Amazon AWS has over 51M [3]! Corresponding proportions of the IPv6 space arguably exceed one's ability to imagine. This trend might lead one to conclude that many IP addresses are key to scale worldwide, provide reachability, implement sophisticated traffic engineering policies, and ensure consistent server selection across multiple TCP connections. In the absence of address space, a CDN's ability to be flexible, adaptive, and innovative would appear to be inversely proportional to the growth of its software and hardware resources.

Implicitly, IP address bindings also constrain service provisioning. For example, different customers have different SLAs or expectations when it comes to availability and quality of service; client behaviours and connection patterns may also change over time; the service operator may change or release new service offerings. Among the available service provisioning mechanisms, the only representation that assuredly connects these dimensions is an IP address. This makes address changes necessarily slow to plan and costly to execute. The result is an operational bottleneck since changes in any

dimension may have impact on otherwise independent dimensions, a relationship that is shown by Figure 1 and explored further in §2.2.

How, then, can a CDN operate, evolve, or even launch, without ever-growing address space? What is the smallest number of IP addresses required to operate at scale? Can the bottleneck be broken by decoupling IP addresses from resources? To do so, must we resort to clean-slate Internet architectures [23, 29, 35], or are more immediate deployments possible? What if IP usage could be separated into control and data planes in keeping with the networks that rely on it?

In this paper we describe efforts at Cloudflare to improve addressing agility for scale by decoupling IP from names and sockets. This has the effect of transforming addresses from a resource constraint to a resource that can be scheduled, and reduces operational bottlenecks. We note that the emergence of virtual IP- and name-based hosting techniques (§2.3) challenges the need for static IP bindings at all. This observation motivates us to identify and re-architect the binding mechanisms themselves: DNS for clients and sockets for services. Our contributions are as follows:

(1) We re-architect authoritative DNS to match on policies instead of names, then select from an address pool assigned to that policy (§3.1, §3.2).
(2) We design and have open-sourced `sk_lookup`, a programmable sockets mechanism to resolve socket inflexibility – removing limitations on IP+port pair selection, and enabling IP+port re-assignment to existing listening sockets (§3.3).
(3) We evaluate the architecture at scale on live traffic at multiple datacenters and random *per query* address selection for 20M+ hosts and services – more than 15% of all websites [67] – from a pool of ~4K, 256, and even a single IP address (§4, §5).
(4) We state why this works for any service that controls its authoritative DNS and connection termination (§3.4); explain why changes are completely transparent to BGP routing, L4 load-balancing, caching, and other surrounding systems (§4.3,§5.1); alongside implications for future (§4.4, §5.2).
(5) We begin to explore the power of policy-driven IP selection by describing fast route leak detection and mitigation for a global anycast network, and more (§6).

In deployment datacenters, our architecture has been serving *all* production traffic that satisfies the given policy since July 2020. In addition to varying the number of addresses in use (from ~4K to 1), the scale of the deployment show that random *per-query* addressees can be generated at rates of 1000s per second.

The architecture is designed to be transferable. The only requirements are that a service provider controls its own (i) authoritative DNS, *and* (ii) connection termination. DNS is necessary to establish address bindings and communicate changes to clients, while connection termination is needed to ensure that processes can accept connections as intended. This means that our architecture generalizes to, for example, university web services as much as it works for many (but not all) types of CDNs and web services.

Of course, many of the observations we make in designing this new architecture have been made by others [5, 33, 50, 56, 57] (we review additional related work in §7), but, to our knowledge, ours is the first to unbind IP addresses in practice, and to be deployed at large scale to real-world clients. This work raises no ethical issues.

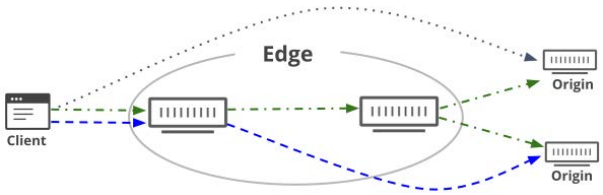


Figure 2: The client-to-origin path can consist of as many as 3 separate connections, with up to 4 out of 6 IP addresses managed by a single CDN.

2 HOW DID WE GET HERE?

In this section we describe address use in CDNs at a high-level, alongside their challenges. We argue that recent advances have provided the foundations to revisit conventional notions of address use.

2.1 The perceived need for address space

In the CDN space, the *server* label is delineated from its instantiations, most often ascribed to their function as on the *edge* or being the *origin*. A server is any machine that terminates a connection. Origin servers hold the ground truth. Finally, edge servers sit on the path between client and origin, typically inserted as reverse proxies [17].

Figure 2 captures the relationship between devices in different CDN and hosting architectures. The conventional end-to-end connection is represented by the top dotted edge, in which client requests are routed directly to origins whose IP addresses must be advertised over BGP. The lower dashed connection represents the edge-service model. Increasingly, edge services are implemented as reverse proxies, i.e., clients connect to an edge service IP address returned by DNS. If the edge service is unable to satisfy the request, a second connection is initiated by the edge service to the origin. Finally, customers of edge services may wish to transit connections to the origin over the edge service's private infrastructure, as indicated by the middle dash-dot line in Figure 2.

In each model represented by Figure 2, origin connections and edge connections that cross the edge boundary need public IP addresses. These IP addresses (at origins and at the edge), returned by DNS, can be dedicated but are most often shared with other services. Two observations can be drawn about connections on today's Internet: The client-to-origin path often differs from the original end-to-end model [54]; and that those origin connections consist of no fewer than three public addresses between two entities.

One obvious way to support additional connections is with additional IP space. Indeed, from data and methodology described by Giotsas et al. [26], we find that ASes that self-identify as Enterprise and Content rely on IP transfers to grow their IP space an order of magnitude more than traditional broadband ASes. On appearance it seems that more services and more interfaces need more IP addresses, a notion this paper suggests can be mitigated.

2.2 The IP bindings bottleneck stifles innovation

Address management is a challenge for any large network, and has long been a subject of research. Among network operators, ISPs, and other traditional consumers of large address space, addresses are associated with points on a path.

Among large CDNs and hosting services, the conventions of IP assignment have exposed an implicit yet long-standing resource-binding architecture, depicted in Figure 1. Its similarity to the seminal 'narrow waist' representation of IP is intentional and unavoidable. Above IP sits a service or website associated with a hostname that is reachable via a designated address. Below IP are sockets that manage client connections over network interfaces. In this manner it is natural to associate an IP address with a software service, or an IP address with hardware resources, or both. The unintended result is a transitive relationship: Since hostnames are related to specific IP addresses and IP addresses are related to specific machines, then hostnames are specifically related to machines.

At CDN-scale this transitive property makes the bindings difficult to reason about, and manifests as an operational bottleneck. Changes are slow and costly because the address bindings impact planning and provisioning, quality of customer (origin) services, client performance, and more. In the CDN context IP addresses can stifle innovation, which is the very opposite of their design [52, 59].

2.3 Enabling technologies: Protocol Multiplexing

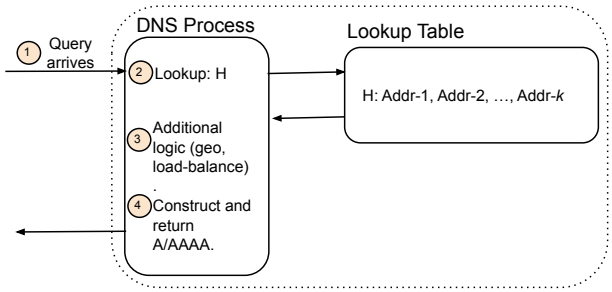
Recent advancements have disentangled some of the conventional bindings between IP addresses, the services they represent, and the physical devices to which they are assigned. The advancements inexorably link to the development of web hosting services and are key enablers for our architecture.

Name-based virtual hosting CDNs used to assign unique IP addresses to each website. This has diminished over time with the adoption of *name-based virtual hosting*. The `Host` header in HTTP enables a single application to provide independent yet identical services. Similarly, the `Server Name Indication` (SNI) field in TLS allows a server to host multiple HTTPS certificates on the same IP+port. This effort was once stymied by slow adoption from clients[1], but client adoption has since changed dramatically. As of 2017, over 99% of TLS connections to major CDNs use the SNI extension [49]. As a result, servers can now safely assume support for SNI. In fact, some services mandate it: many Google services are inaccessible by clients that do not support SNI [32, 60].

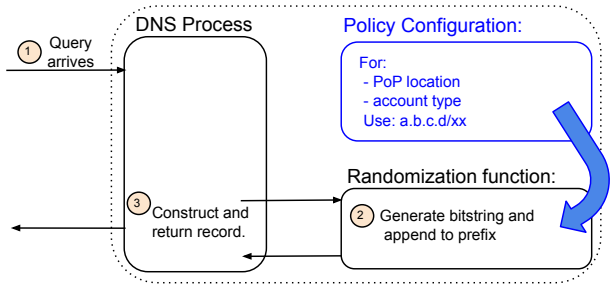
IP-based virtual hosting Recent advancements built atop equal-cost multipath routing (ECMP) allow multiple instances of a single service to sit behind a single public-facing IP, on multiple servers [22]. Though often referred to as a virtual IP (VIP), this model adheres to the IP's original design allowing a single address to interface with different network architectures [52]. It also has the perhaps unintended effect of decoupling addresses from network interfaces, so that the mapping between them is flexible and dynamic.

On the client side, advancements in networking stacks and APIs also encourage applications to move away from IP addresses and towards names. Apple's Network Framework APIs [4], for example, support "connect-by-name" as the primary mechanism for establishing UDP, TCP, and TLS connections. Similar APIs can be found in standard libraries for modern programming languages such as Go.

[1]For years, Akamai limited its use of SNI to remains backwards-compatible with old versions of Windows (e.g., Windows XP) that did not support it.



(a) Conventional DNS uses name to lookup addresses.



(b) Our architecture matches policy without name; a random bit-string is appended to a prefix representing the policy.

Figure 3: Policy-first DNS architecture shifts from matching names to instead matching on attributes that represent a policy; address pools are assigned to policies.

3 HOW DO WE GET OUT OF HERE?

Our overarching goal is to relax the bindings between IP addresses, hostnames, and sockets. In later sections, we show that doing so enables greater flexibility and innovation for network engineering and services. To that end, in this section, we present two complementary designs that, together, completely decouple IP bindings. First, we describe a modification to DNS that separate DNS records from the addresses that populate them. This has the effect of creating independent control and data channels for DNS.

Second, to escape from the rigidity of IP-to-socket bindings we design a scalable and flexible programmable socket architecture, `sk_lookup`. The `sk_lookup` design enables dynamic mapping IP+port pairs in any or all combinations – an otherwise decades-long constraint – to sockets. The implementation with BPF [28] was recently accepted into the mainline Linux kernel [43, 44]. In a spirit similar to the DNS changes, `sk_lookup` has the effect of separating sockets into control and data channels.

3.1 DNS to Decouple IP–name bindings

We assume, for the purpose of presentation, that a service provider originates an IP prefix announcement, and that the IP range is orders of magnitude smaller than the set of hostnames associated with the service. How then do we map or move hostnames to IP addresses?

The fan-out properties of L4 ECMP and consistent hashing load-balancers ([22, 45]) may appear suitable. However, L4 load-balancers are themselves constrained by (virtual) addressing. Alternatively, QUIC anticipates a desire to accept a connection on one IP address, and receive a reply from another [34]. While novel, tangible

progress has yet to be made and use cases are restricted to QUIC, rather than generalized across transport services.

Instead, IP addresses should be mapped where the binding to hostnames occurs – that place is authoritative DNS. From the perspectives of the client, and the connection, the binding between destination hostnames and IP addresses is due solely to DNS. Name-to-IP mappings in DNS are generally static in nature, stored in some form of a record or lookup table.

However, despite the existence of static lookup tables, the binding is only known 'on-query'. In other words the address is returned in response to a DNS query—the last possible moment in the lifetime of a request where a hostname can be bound to an IP address. Figure 3a shows conventional DNS, in which names may be mapped to possibly multiple addresses for the purposes of path redundancy, proximity selection, or load-balancing [9]. For all *practical* uses of that mapping, the address(es)-to-name binding occurs at the moment that DNS generates and returns the response.

This leads to the observation that addresses are bound to names at the moment the response is returned, rather than within any record. The separation is subtle, but important: DNS responses in today's architectures use a name to identify a set of of addresses, from which a subset is selected based on some policy logic. We instead invert this relationship, as represented in Figure 3b. Instead of IP addresses pre-assigned to a name, our architecture begins with a policy that may (or in our case, not) include a name. For example, a policy may be represented by attributes such as location and account type (as in our deployment described in §4). The attributes identify a pool of addresses that represent the policy. The pool itself may be isolated to that policy or have elements shared with other pools and policies.

Note that in this model, all addresses in the pool are equivalent. Any address may be returned without inspecting the DNS query name. Instead, IP addresses are computed and assigned at runtime or query-time. The lifetime of the name-to-IP binding is upper-bounded in time by the larger of connection lifetime and TTL in downstream caches. The binding itself is otherwise ephemeral and can be changed without regard to previous bindings, resolvers, clients, or purpose.

3.2 From Policy to Practice

The address pool can consist of any set addresses. In our design the set of policy attributes is associated with an address pool described by a prefix, `w.x.y.z/b`. From within the pool our design defaults to random selection, as instantiated in §4. We note that IP randomization in and of itself is far from new [9] and is implemented by CDNs, albeit limited to a few addresses [2, 13]. Alongside, domains are increasingly co-hosted; recent studies show 20% of observed domains are co-hosted with more than 1K other domains onto a single IP address, up from 6% in 2007[31]. We view our architecture, which foregoes all notions of fixed IP-to-name ratios, as the next logical step change in this evolution.

The DNS architecture is described by Figure 3b and takes the following approach:

(1) A query arrives for an `A` or `AAAA` record.
(2) Any processing, validation, or logging (e.g., for accounting or debugging), remains unchanged.
(3) Attributes match to a policy (specific examples §4, §5, and §6) that identifies a prefix.
(4) Given a prefix of length b, generate a random bitstring of $32 - b$ (for IPv4) or $128 - b$ (for IPv6).
(5) Respond with the address that is the concatenation of the prefix with the random bitstring.

This approach makes no assumptions about query patterns or contents. For example, consider three hostnames h_i, h_j, h_k. The randomly generated addresses returned for any of (h_i, h_j, h_k) and (h_i, h_i, h_i) are equivalent: IP addresses are *i.i.d* irrespective of ordering or frequency. As a result, *all* hostnames will appear on *all* of the addresses in the pool given a sufficient window of time.

We place no bounds on the number of hostnames that may be mapped onto the IP pool since this mapping is carried by SNI and HTTP `Host` (see §2.3). Nor do we put limits on the prefix length. CDNs and hosting services already bind multiple hostnames to individual IP addresses. SNI enables hostnames to exceed IP addresses by orders of magnitude. Our own evaluations later will show this works at a ratio of 20+ million hostnames to 1 single address.

The flexibility generated by a policy- rather than name-based DNS architecture is instantiated by later sections. However, the inflexibility of sockets is a barrier to a complete decoupling of IP addresses that must first be resolved.

3.3 From listen to lookup: Re-engineering the sockets stack for scale

The inflexibility of standard BSD-style socket implementations is both known and the subject of research [66]. Perhaps less well known is that sockets increasingly are barriers to services at scale. Briefly stated, it is exceptionally challenging to set up network services to listen on hundreds of IP addresses without causing the network stack to buckle, and even break.

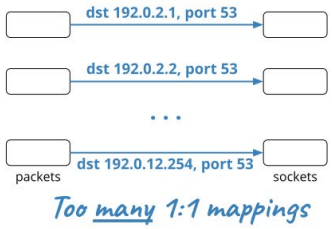
Broadly speaking, the way that sockets are bound to IP+port pairs has three limitations: (i) Each new socket consumes memory, and increases the search time to find the right socket for an arriving packet; (ii) any IP+port pair selection artificially restricts other IP+port pair selections; (iii) once bound, sockets are unchangeable and inexorably tie the software interface to the IP+port.

We propose a new design: *programmable socket lookups*. Our implementation, BPF `sk_lookup`, is open-source and was recently accepted into the mainline Linux kernel [43, 44]. A brief overview is helpful to understand socket limitations.

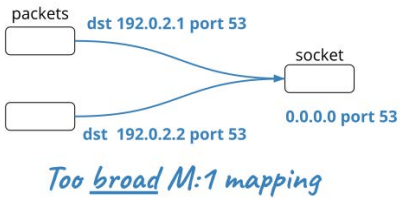
Standard sockets primer and problems The default association between an IP+port and a service is one-to-one. The service application, itself, must open one socket for each transport protocol (be it TCP or UDP) it wants to support. For example, an authoritative DNS service would open up two sockets (`a.b.c.d:53/tcp` and `a.b.c.d:53/udp`). Each socket is given its own file descriptor.

The 1:1 mapping means that CDN services must have at least one listen socket for each IP address in use, as depicted in Figure 4a. For example, 4096 sockets are needed to listen on any single port (e.g., 80) for each IP address in an advertised /20—before doubling in number to accommodate both TCP and UDP.[2] The associated memory and look-up performance penalties do not go unnoticed.

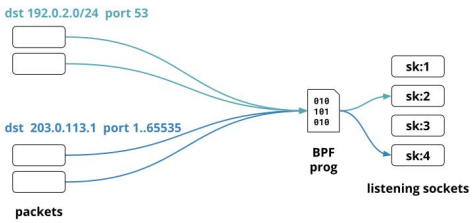
[2] Admittedly this approach, while naïve, has an isolation advantage: a UDP flood attack on any IP in the range has no impact on the receive queues of sockets bound to the remaining IP addresses in the range.



(a) The standard socket model scales linearly in memory and incurs performance penalties with large numbers.



(b) The bind-to-any `INADDR_ANY` catch-all address is inflexible and a security hazard.



(c) An `sk_lookup` example: Packets arriving on 192.0.2.0/24:53 to socket `sk:2`, while traffic on 203.0.113.1 to any port number lands in socket `sk:4`.

Figure 4: Sockets are unscalable, inflexible, and static. `sk_lookup` solves this by decoupling IP+port pairs from listening sockets.

Alternatively, the sockets API comes with an 'any' facility as `INADDR_ANY` or the `0.0.0.0` wildcard address. A socket created with the 'any' facility will respond on all addresses assigned to the server, but specified to one port number. As shown in Figure 4b, compared to the naïve "one address, one socket" model, `INADDR_ANY` provides a single catch-all listening socket for the whole IP range on which connections are accepted. It works by being the last in a chain of lookups for more specific sockets, as shown in Figure 5a.

The `INADDR_ANY` facility deserves closer inspection in a CDN context. One advantage of binding to `0.0.0.0` is that applications become address-agnostic, allowing for addresses to be added or removed after socket binding without need for an application reconfiguration or restart. Conversely, this catch-all creates a security vulnerability: Listening on more addresses than needed may otherwise expose an internal-only service to external traffic that has no firewall or socket filter in place. Furthermore, one socket has one receive queue, so an attack on any IP address will cause legitimate packets on other IP addresses to be dropped[3].

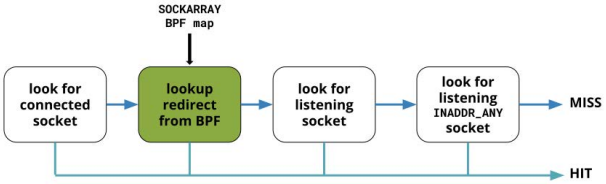
Perhaps the greatest downside, however, is loss of flexibility in services. In particular, what should happen if another service is later bound to a specific address and port pair? In Linux, for example, a service that listens on the wildcard `INADDR_ANY` address claims the port number exclusively for itself. Attempts to listen on a specific IP and a port bound to the wildcard-listening socket will fail.[4]

Unfortunately the sockets API does not allow us to express a setup in which two services share a port and accept requests on disjoint IP ranges. Nor does it offer a facility to listen on *all* ports simultaneously for any IP address or range. Nor does it allow for a socket's IP+port bindings to be altered.

On appearance these may seem like corner cases that can be resolved with a new socket option, but at scale cases such as these are common. Consider, for example, services that share the same port number but otherwise respond to requests on non-overlapping IP ranges. One prominent instance occurs when a recursive DNS resolver runs side-by-side with the authoritative DNS service.

`sk_lookup`: Programmable socket lookup The inflexibility of sockets at scale motivates us to re-evaluate and re-engineer socket bindings. In lieu of 1-to-1 or all-to-1 address to socket bindings, the ideal is a programmable and flexible facility that matches an incoming packet with a listening socket, ignoring the file descriptor to which the socket may have been bound. Our design is captured by Figure 5a, in which a program is executed on packet arrival and identifies the appropriate socket receive queue. One novel aspect of our design is that it leaves sockets untouched. We label the design as `sk_lookup`, and implement the program using BPF [28].

The program itself takes a BPF map structure for reference and is injected onto the TCP socket lookup path. As shown in Figure 5a, its execution is second stage, triggered after the lookup path attempts to match a connected socket (4-tuple) and before looking for a listening socket (2-tuple). The program structure resembles a firewall rule, as shown in Figure 5b, and consists of a set of match statements followed by an action. The map is populated by a socket activation service that receives a file descriptor for any socket created. Upon receipt of a file descriptor the service updates the BPF map.



(a) Programmable `sk_lookup` is injected on the standard socket lookup path after connected sockets are matched, but prioritized ahead of IP+port listeners.

```
SEC("sk_lookup/redir_prefix")
int demo_redir_prefix(struct bpf_sk_lookup *ctx)
{
    struct bpf_sk_sock *sk;
    int err;

    if (ctx->family != AF_INET)          match on 192.0.2.0/8 tcp/80
        return SK_PASS;
    if (ctx->protocol != IPPROTO_TCP)
        return SK_PASS;
    if (ctx->local_port != 80)
        return SK_PASS;
    if ((bpf_ntohl(ctx->local_ip4) >> 8) != (IP4(192, 0, 2, 0) >> 8))
        return SK_PASS;

    sk = bpf_map_lookup_elem(&redir_map, &KEY_SERVER_A);
    if (!sk)
        return SK_PASS;                  map-based socket selection

    err = bpf_sk_assign(ctx, sk, 0);
    bpf_sk_release(sk);
    return err ? SK_DROP : SK_PASS;      action - redirect
}
```

(b) `sk_lookup` program is a set of matches and actions.

Figure 5: BPF `sk_lookup` in the kernel.

[3]The Linux kernel offers protections in the TCP stack, but UDP requires special care.
[4]See https://man7.org/linux/man-pages/man2/listen.2.html#ERRORS.

The Linux kernel patch [43, 44] was accompanied by rigorous performance evaluations required for acceptance into the mainline, also public [62, 63] but omitted for space. Results indicate penalties of ~1-5% from baseline measures of packets per second (~1M TCP and ~2.5M UDP) and CPU usage. This is regarded a *positive result in exchange for the increased flexibility*.

`sk_lookup` is proving to be a powerful construct at CDN-scale. In addition to enabling our efforts to completely decouple IP from names and resources, `sk_lookup` is anticipated to be a crucial tool for multiple streams of future investigation.

3.4 Generalizability, Transferability, and Benefits

One question that arises is whether these mechanisms generalize to other domains. The short answer is yes, as described below.

Generalizability and Transferability This approach generalizes to any *listening* service that uses HTTP, TLS, or other name-based multiplexing, and that meets the following two requirements:

i. Control of authoritative DNS – the service operator must be able to execute its policy-based selection when responding to a query (as in our deployment), or populate authoritative DNS records with policy mappings generated off-line.
ii. Control of connection termination or, at a minimum, the ability to listen on the selected address(es) and ports.

The above pair of requirements defines the transferable domain as being any service operators that manage its own authoritative DNS and connection termination, irrespective of size.

We emphasize that the use of anycast and `sk_lookup` are optional. `sk_lookup` injects flexibility into socket-to-process mappings and efficiency in IP:port pair allocations at scale, as discussed in §3.3. In §6 we describe ways that addressing agility can be combined with anycast to build new features and systems.

Outcomes: Operational and Address Efficiency The initial motivation for this work was to improve address agility, while simultaneously reducing planning and execution costs associated with address management. We observe that large changes in address usage need take only as long as necessary for stakeholders to agree, and minutes or seconds more to execute. Alongside, "more address space is needed" thinking and impulses (as well as their costs) are no longer conventional as the operations grow. For business reasons we are unable to reveal exact measures, but engineer-months or weeks are evolving into hours and minutes. The degree of flexibility, and the reasons this system works, are demonstrated in §4.

In addition, a third *qualitative* observation is unanticipated yet proving to be beneficial: Engineers are learning to ignore IP addresses as constraints, and instead treat them as resources that can be scheduled. When designing new features and systems for managing infrastructure, IP addresses are being treated as an after-thought. The process no longer begins by asking about address availability or usage as a way to limit design possibilities. Instead, algorithms and solutions are designed with generic identifiers, each representing a unique property, attribute, or policy. In the final design, the identifier corresponds to an IP address. The demotion of the IP addresses from a first class object in this manner has enabled us to reason about new systems or improve existing ones, as described in §6.

In the next sections we deploy our architecture, evaluate the one-address hypothesis, and then begin to use the architecture to build features that have no obvious alternative designs.

4 AT SCALE: RANDOMIZE 20M TO /24

The previous section described two halves of an architecture that completely decouples IP addresses from names and servers. Together they enable a CDN to treat addresses as a flexible and dynamic resource: Rather than a constraint that must be pondered, addresses form a resource pool that can be scheduled, removed, and re-added.

Here, we pose the question: Are there limits to address usage or rates of change as a CDN or service provider scales? We note that *all* standard measures of our system's performance reduce to sets of before and after evaluations that are indistinguishable. This is expected since the first order evidence of success in our system is the absence of breakage. For this reason, otherwise standard performance measures are uninteresting and omitted for space.

We first present and evaluate the random selection strategy, followed by single-address evaluation in §5.

Randomize IP addresses Rather than bind a service to a set of IP addresses, for all websites and services that match the policy – more than 20 million in our deployment – this solution randomly selects an IP address per-DNS query as described in §3.2. At any point in time, any website or service hosted by a CDN could take on any of the IP addresses. Our operational deployment shows that this has no negative impact on performance, and can enable improvements.

4.1 Production Edge Evaluation Architecture

The Cloudflare network is comprised of points of presence (PoPs) across more than 200 cities in over 100 countries, and direct interconnects with over 9500 distinct networks.

From a networking perspective, the standard CDN service offerings are characterized by two properties. First, the network operates as a reverse proxy on behalf of its customers' origin servers, whose hostnames are registered with Cloudflare's authoritative DNS. This enables Cloudflare to return its own IPs in response to DNS queries, and terminate connections on behalf of customer origins.

In addition, Cloudflare uses anycast—not just for DNS service—but for all of its web services. Accordingly, a DNS query at all PoPs for a customer origin will receive the same IP address in response. While the use of anycast is most commonly associated with reachability, its use for content provides an additional benefit: Identical network configurations are mirrored across all PoPs and datacenters, which facilitates deployment and management.

Each PoP in the network is co-located with a datacenter. The data center architecture and software stack are both architected for simplicity. In particular, the server software stack is uniform by design, as encapsulated by the blue-dashed line in Figure 6. Rather than different servers with different services, each server mirrors a single software stack and offers all services—every server executes distributed denial-of-service, layer-4 load balancers, connection termination, and the full suite of application processes. Every machine also participates in the distributed cache. Servers logically sit between an ECMP router at the ingress, and an origin gateway at egress. The ECMP router doubles as the datacenter's first-pass stateless load

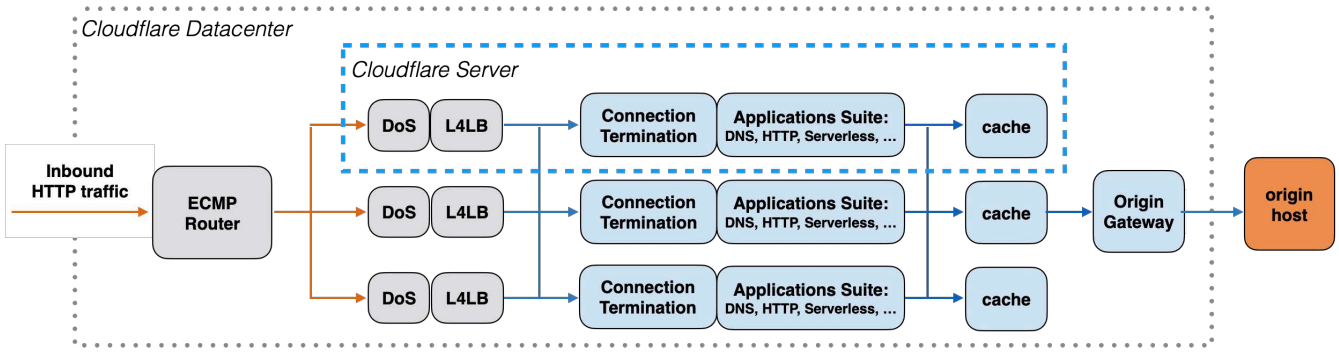


Figure 6: The edge architecture is designed for simplicity: Servers share a uniform software stack, as encapsulated by the dashed line. An ECMP router with consistent hashing fans connections out to servers. An additional L4 load balancer between them precedes connection termination and application suites. All servers participate in the distributed cache. Our changes are entirely transparent. Only authoritative DNS is touched and, separately, `sk_lookup` is installed automatically with the Linux kernel. DoS protections, L4LB, caching, and all surrounding systems are unchanged.

balancer that, similar to previous systems [22, 45, 48], hashes packets in a consistent manner to spread connections between servers.

Specific component details are out of scope and left for future work. Most notably, the datacenter architecture is composed of familiar service architecture components. We emphasize that *no changes were required to surrounding components; our changes are scoped to DNS and otherwise are completely transparent.*

4.2 Experimental Setup with Live Traffic

Our architecture is currently deployed across the whole CDN in keeping with the uniform software stack. It is currently active, at scale, and is expected to remain active beyond final submission of this work while we investigate new systems described in §6. Our system has been running as follows:

- 6 PoPs/DCs at 8 IXPs serving 5 contiguous timezones;
- 100% of DNS responses for 20+ million hostnames;
- ~5-6K DNS queries per second (mean)
- ~35-40K HTTP requests per second (mean);
- active from 2020-07;

at each PoP, the following socket and network configuration:

- One /20 (IPv4) and one /44 (IPv6) address pool;
- Both prefixes advertised from all PoPs (anycast);
- connection termination listening on ports `80`, `443`, and 11 others [16] for the entire /20 and /44 ranges.

The timetable for in-use portion of addresses within the /20:

- 2020-07 to 2021-01, 4096 addresses (the full /20);
- 2021-01 to 2021-05, 256 addresses (/24);
- from 2021-06, on-going, 1 single IPv4 address (/32) in a mid-sized datacenter (see §5).

We emphasize that our architecture is a drop-in software modification to existing production architecture and systems. The deployment uses the complete set of production servers and infrastructure, and all surrounding systems are both unchanged and unaffected—*all existing measures of performance were found to be indistinguishable.*

The breadth and duration of the deployment allay any concerns about scale or feasibility – for more than 1 year, clients have initiated ~500 million DNS queries per day, followed by ~3-4 billion HTTP requests, to addresses selected per-query purely at random. For comparison, the same hostnames at all remaining 200+ data centers were mapped across 18 /20s. The reduction in address usage is 94.4% for the /20, and 99.7% for the /24. A /24 is the minimum permissible address range in BGP. In §5 we reduce IP usage to /32, and describe ways that the 'leftover' addresses from the required /24 might be used for services and resilience in §6.

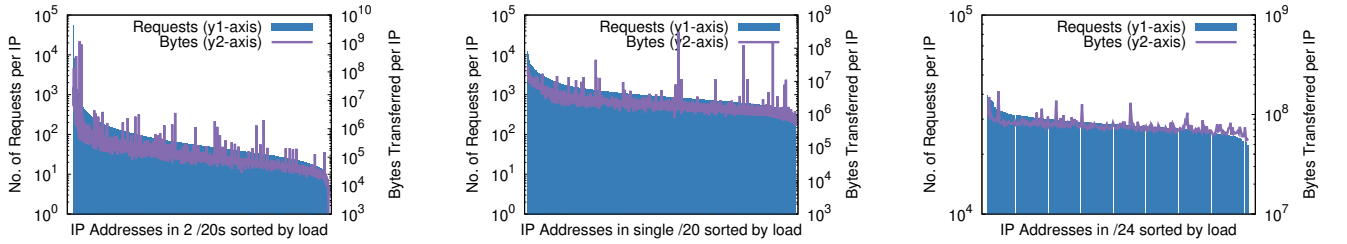
A comment on the operational benefits. The reduced address space is evident. However, before describing why this works, one particular observation may be of value to the wider community. The per-query *rate of change* of IP addresses made evident by this work is changing address management internally. From an operational perspective, address management is decreasingly concerned with hostnames and servers. In return, network and address management is increasingly focussed on the use of IP addresses for their intended purposes—routing, reachability, and quality-of-service— resulting in improvements on engineering time, flexibility, and ingenuity.

4.3 Why does this work, and why do it?

We first remind our reader that our changes to DNS are entirely transparent. The surrounding systems, software stack, and configurations are untouched. In addition, *all performance metrics are identical and indistinguishable*, and omitted accordingly. Here, we explain the reasons that randomization works, and its benefits.

Routing is unchanged. Reachability between autonomous systems (ASes) is evaluated over prefixes advertised by the AS that either owns the addresses or is permitted to advertise them. Furthermore, forwarding is decided by longest-prefix match. Since BGP routing succeeds at the granularity of IP prefixes, the semantics of a prefix used for randomized addresses is identical to prefixes used for statically bound IPs.

Smaller prefixes are harder to leak or hijack. To mitigate against routing table inflation, a best-practice convention among network



(a) Pre-agility: two (of 18) busiest /20s. **(b) Per-query random addressing in a /20.** **(c) Per-query random addressing in a /24.**

Figure 7: Load across IP addresses at a medium-popularity facility, serving 20M+ hostnames, approximately 10% of websites [67]. (a) In the two busiest (out of 18) /20 prefixes *without* addressing agility; (b) Randomizing over a single /20 *with* addressing agility; and (c) Randomizing over a single /24 *with* addressing agility. Data is comprised of 1% of all requests taken over 24 hours. The x-axis is IP addresses in descending order of number of requests each IP address receives. This shows that, in today's CDNs, per-IP address load is far from uniform – but can be made uniform by decoupling IP addresses from names.

operators is to refrain from advertising narrower than a /20 prefixes without good reason (/48 in IPv6 [72]). However, larger prefix advertisements increase the likelihood, and decrease the visibility, of narrower sub-prefixes that leak or are hijacked. It is for this reason that some prefixes are advertised as /24s, e.g., Cloudflare's public DNS resolver [14]—by advertising the narrowest prefix permitted by BGP, the resolvers are resilient to hijacks. We revisit route leak detection and mitigation in §6.

Since a single /20 can be used to reach limitless hostnames and services, then larger or more blocks may be unnecessary. Our evaluations show that a /24 (256 or fewer addresses) is equivalent to a /20 in this respect. The implication of our deployment is that a CDN could operate with fewer addresses and simultaneously, as if for free, increase its resilience to route leaks and hijacks – without inflating Internet route and forwarding tables.

ECMP and consistent hashing are unaffected An ECMP group connects a single virtual IP to multiple servers. Alongside, consistent hashing ensures that all packets for a connection are forwarded to a single server. At scale in operational settings, the number of ECMP groups can grow to tens of thousands. Such numbers are difficult to manage, and also exposed limits of router software [38]. The flexible address assignment enabled by our architecture may facilitate ECMP changes. However, our architecture exists independently from ECMP and consistent hashing, for which complexities are dominated by numbers of servers and not IP addresses.

Distributed caches and filesystems are unaffected Our architecture and its addressing are isolated from cache systems. In the CDN architecture, as indicated by Figure 6, every server participates in the distributed cache. Both internal addressing schemes, and distributed filesystems are untouched.

DNS changes are trivial; opportunities and challenges shift to policies. Our experience shows that necessary changes to DNS are relatively small, and otherwise eclipsed by planning and execution costs associated with changes to hostname-IP bindings, sockets, ECMP, quality of service routing and resourcing. In addition, policies can be designed and executed independent and irrespective of IP addresses. This contrasts with the convention of designating addresses in DNS records and systems as proxies for a desired policy. In our evaluation deployment the policy is expressed as datacenter locations and account type. A match on both attributes then triggers randomized addressing—hostnames are completely ignored. Queries that do not match are resolved as normal. This approach enabled code changes to be deployed globally, so that a single codebase could be maintained during evaluation instead of two.

One question that has emerged is how best to design and allow more expressive policies? Safe and verifiable policy expression and processing is left for future work.

"Shared ~~fate~~ benefit, shared load" Our architecture also facilitates provisioning. Different hostnames will incur different loads that may change over time and are hard to predict a priori. Since IP addresses are shared between hostnames and services, and IP addresses can be associated with physical resources, the choice of address can affect wider services. Given any set of candidate addresses, our system's answer to "which IP addresses should be assigned?" is *all of them.*

This idea is borne out of one non-standard performance metric—measures of load per IP address equalize. The effect is shown by Figure 7, which plots dual measures of requests-per-IP ($y1$-axes) and bytes-per-IP ($y2$-axes) in a medium-sized datacenter before and after our architecture is deployed. Notably, the equalization emerges without a priori engineering or post-analysis. This is an artefact of the address randomization, itself.

Figure 7a shows dual measures of per-IP load before our system was deployed. Measurements are drawn from 1% samples taken over the two most loaded prefixes (out of 18) in a medium-sized datacenter over a 24 hour period in June 2020. The 8192 individual IP addresses are sorted on the x-axis from most- to least-loaded by number of requests per address indicated by the $y1$-axis. Corresponding values for bytes transferred over each address are plotted in place, and indicated by the $y2$-axis. The differences between most and least loaded addresses are ~4–6 orders of magnitude. Excluded from Figure 7a is the observation that these differences only increases when including the remaining 16 prefixes allocated to hostnames that fall under our test policy.

The same measurements captured during our deployment reveal a sharp contrast. Recall from §3.2 that our deployment responds to each DNS query, independent of the hostname, with an address selected at random from the address pool. Figure 7b shows that random addressing within a /20 reduces the load gaps to less than

2 and 3 orders of magnitude in the same day of week 24hr period. Figure 7c shows that random addressing from a smaller /24 closes the load gaps to near uniform levels of magnitude, and to factor of less than 2 in absolute terms.

Absent randomization, reliable *a priori* predictions about the popularity or usage of hostnames is difficult, if not impossible. This has consequences on most aspects of provisioning. The use of all addresses in a set of any size precludes the need to plan or provision on the basis of popularity. Instead, a natural and effective load balancer emerges by simultaneously yet dynamically binding all hostnames to all available addresses in uniformly distributed manner, irrespective of request patterns and demands.

4.4 Implications and Potential Pitfalls

For the sake of completeness we raise the following implications of IP randomization, identify concerns, and resolve most. Later we show that placing services behind a single address works, resolves any potential pitfalls, and may improve application performance.

TTL Values and Individual Behaviors Any dependency on DNS responses to somehow shape ensuing request patterns should be avoided. This is because DNS responses are cached both at recursive and local client stub resolvers. The subsequent traffic generated by any single authoritative response is determined by the number and behaviour of downstream resolvers and clients. For well-behaved resolvers, TTLs provide some granularity of control. Recent evidence suggests, however, that resolvers commonly modify TTL values [46, 47]. Even so, agile IPs do open opportunities to investigate resolver behaviors by changing TTL at the authoritative DNS and comparing addresses that are returned with addresses used to connect. We revisit this observation in §6.

Denying Denial of Service (DoS) Denial of service attacks can target IP addresses (layer-4 attack) or hostnames (layer-7 attack). Identifying the layer of the attack may, on appearance, seem more difficult if randomizing IPs. We argue the root of the challenges lay, not with randomization, but with the multiplexing of many hosts onto individual IPs that exist independently of our scheme.

We also posit that an agile addressing architecture is a natural fit for DoS mitigation for two reasons. First, the total load for all affected services is immediately shared equally by all reachable servers. Since our architecture operates at layer-3, separately and isolated from L4 and L7 balancers, the impact of attacks is less pronounced at higher protocol layers.

In addition, many DoS mitigations at CDN scale work by temporarily changing hostnames to address assignments, and dropping or directing packets destined to the affected IP addresses elsewhere. In this way, DoS mitigations need the ability to quickly assign different IP addresses to names and servers. We exploit this observation in §6 to sketch a DoS mitigation mechanism built using addressing agility to quickly identify DoS targets and mitigate attacks.

HTTP Connection Coalescing HTTP/2 [7] supports connection reuse, and permits resource requests on domains that differ from the domain used to connect. There are two conditions under which connection reuse for a target resource is allowed:

(1) The target resource URI authority matches that of the certificate associated with the connection.

(2) The URI host resolves to the same IP address as the given connection.[5]

IP address randomization may prohibit the second condition. There are several reasons why this may not be a problem in practice. First, browsers and operating system stacks are increasingly narrowing the context in which connection state may be shared. Connection reuse is often limited to a single process and, depending on application characteristics, within certain contexts of that process (e.g., tab-isolation). In §5.2 we show opportunities potentially missed by randomization across many addresses are instead exploited by use of one address. Note that HTTP/3 [8] does not require IP address matching, so randomization would not affect coalescing those connections.

Non-TLS or HTTP based Services Applying our architecture more broadly than to TLS and HTTP(S) presents a potentially separate yet surmountable set of challenges. One service that might be adversely affected by randomized IPs is `ssh`, which maintains a `known_hosts` file that stores the hostname-to-IP address mapping, and issues a warning when the IP address used to connect is different than is stored in the file. This association, while understandable, is outdated and already broken given that many DNS records presently return more than one IP address.

5 ONE ADDRESS TO SERVE THEM ALL

If millions of names and services can be hosted behind ~4K addresses, and 256 addresses, then why not 1 address? We conducted one additional experiment to evaluate feasibility.

Map *everything* to a *single* IP address Our deployment indicates that as more services and machines reuse the same IP address, the benefits continue to improve. We show that extending this to its logical conclusion—a single IP address for the *entire* set of services—is feasible, perhaps even preferable.

In early 2021 we parameterized a 24-hour trial at the datacenter used in Figure 7 to return a "random" IP address from a set of size 1 (i.e., a /32). Its success has motivated an extended trial—from June 2021 and scheduled to continue after final submission, this medium-sized datacenter has been serving the same 20+M hostnames and services with a randomized address set of 1.

We stress that all available system and performance metrics remained *unchanged*. This experiment suggests that, services and hostnames being equal, an autonomous system's content and hosting services can be bound to a single IP address. The result is newfound levels of flexibility and granularity of control that enable entirely new systems and visibility, as described in §6. Below we argue that one-address also amplifies all benefits that are ascribed to randomization over a larger set, and additionally resolves potential pitfalls of randomization identified in §4.4.

5.1 Why one address works, too

The explanations in §4.3 equally apply to one address. It is instructive to restate those reasons in this context, and clarify differences.

[5]Not all browsers implement this check the same. Some browsers require the IP address set for the new host to contain the IP address used for the existing connection, whereas others assume transitive properties among IP address sets that have intersecting addresses between them.

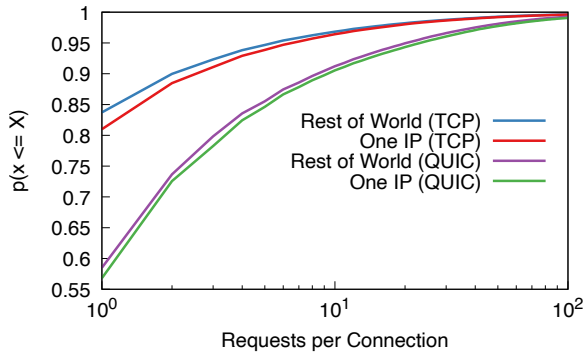


Figure 8: Preliminary evidence of connection coalescing: Placing all services on one IP address increases requests per connection relative to address to name bindings used in the rest of the CDN.

Routing and traffic engineering are unchanged, IP space is liberated This argument is identical to IP randomization. One in-use IP address has no effect on inter-domain routing, which admits a maximum of /24 and /56 prefix lengths. Once inside the autonomous system, a packet is correctly forwarded to its destination. One-address also makes it easier to reason about IP space and traffic engineering—an otherwise challenging endeavours at CDN-scale.

The reachability of innumerable services behind one-address also has implications for smaller or non-CDN operators: (i) Few addresses are required to function, even one; irrespective, (ii) standard sockets are sufficient without the use of `sk_lookup` (see §3.4).

IPv6 Applicability The abundance of IPv6 prefixes and addresses makes reasoning about its bindings difficult. One argument against binding to a single IPv6 address is that there is no need, i.e., address exhaustion is unlikely. This is a pre-CIDR position that, we claim, is benign at best and irresponsible at worst. In lieu of asking why use a single IPv6 address, we should be asking why not? and reject any answer that reduces to, "because there is no need."

Connection Coalescing The potential scale of service under scrutiny notwithstanding, we note that one-address preserves any semantics ascribed to IP addresses such as SSH's `known_hosts`.

We expect connection coalescing will increase when all services sit behind one address (assuming target URIs match certificates). The evidence for coalescing is preliminary, yet favourable. Figure 8 shows requests per connection for services on one address, and standard addressing used by the CDN in the rest of the world.

Measurements were conducted over a 7-day window in June 2021, and consist of a 1% sample of all connections over all TCP and QUIC. For this reason, samples at the 'One IP' datacenter are skewed by connections not on the one IP address, as well as connections from HTTP/1 and older browsers that do not support connection reuse. Despite the additional noise, a 2-sample Anderson–Darling test [55] suggests a significant difference. We tested the hypothesis that the observations from the 'One IP' data center, and the "Rest of World" data centers can be modelled as coming from a single population. According to our results, the hypothesis can be rejected with 99.9% confidence since the returned test value $AD = 3532.4$ is higher than the critical value $AD_{crit} = 6.546$ for significance level of 0.001.

5.2 Upstream Implications and Opportunities

One-address is functionally equivalent to both conventional addressing and IP randomization. Even so, and in contrast to IP randomization, one-address does raise the possibility of upstream effects that deserve consideration. None have presented in our evaluations so far, nor in communications with upstream networks.

Upstream Routing Errors are Immediate and Total If all traffic arrives on a single address (or prefix), then upstream routing errors affects all content equally. This criticism has two responses. First, this is an address diversity problem that exists independent of the way addresses are used or the scale of operation, as are resolutions. For example, one solution is to allocate two addresses in DNS records from non-overlapping prefixes [15].

Alongside, our architecture provides fast mitigation. The per-query rate of address change (see §4) means that addresses can be changed en masse by changing affected address pool attached to any policy. The changes will be immediate for new queries, and cached records will update in a time that is upper-bounded by TTL.

Upstream DoS Protections The concentration of prefix- or many prefixes-wide traffic to a single address might be mistakenly interpreted by an upstream observer as a denial of service attack. An ISP could conceivably respond by triggering a well-meaning protection in the form of rate-limiting, or block. This can be prevented by means of communication and community engagement.

Port exhaustion in IPv4 NAT is accelerated, as is a push to IPv6 From the client-side, the number of permissible concurrent connections to one-address is upper-bounded by the size of a transport protocol's port field. For TCP this is no longer an issue [21, 42].

In UDP (QUIC), however, the only way to reuse ports is with `SO_REUSEPORT`. This could cause carrier-grade NATs to exhaust available UDP ports, or lose the IP+port pair uniqueness that is required for address translation. One option may be to incorporate other transport-protocol headers into the NAT binding function, such as QUIC connection identifiers, but these are decreasingly available as encrypted transport becomes increasingly dominant.

To the best of our knowledge this is *the only drawback to one-address, and is also immediately resolved by migrating to IPv6*.

Opportunities opened by one-address at the Client Running a CDN on a single IP address may enable the fascinating new space of *client-side* optimizations that arise as a result. Standard tasks like DNS lookups and establishing TCP connections can comprise large fraction of page load times (7% and 53%, respectively) [65]. When all content is served from the same IP address, a client can potentially avoid these performance hits.

For example, one-address could be used to reduce stresses on DNS. CDNs commonly use low DNS TTLs to permit rapid load rebalancing. Under one-address, a CDN can adopt long-lived expiries akin to root DNS servers, thereby extending cache duration and reducing frequency of client DNS requests. An interesting area of future work is to preemptively inform clients that the one-address CDN hosts a particular domain. One possibility is to explore a new form of HTTP resource hint that merely indicates the one-address CDN host (making a DNS lookup unnecessary). Another possibility is to use compact filters to represent all of the hostnames hosted by the CDN [41]. We leave such optimizations for future work.

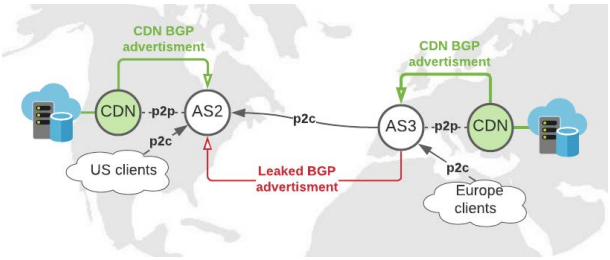


Figure 9: An example BGP leak of an anycasted prefix, based on an actual incident that affected DNS performance. CDN originates an anycasted prefix from multiple PoPs to regional peers. *AS*3, preferring customer routes, leaks the prefix to *AS*2. Performance degrades for US clients routed to Europe, but the leak goes undetected. A policy to map each PoP's DNS responses to a unique address in the range would enable every CDN location to monitor requests on unexpected IPs in that range, and flag potential route leaks. Services can be immediately moved to a different prefix with agile addressing.

TCP connection establishment and longevity is another area of possible investigation. When all hostnames are multiplexed over the same IP address, is it necessary to maintain more than one long-lived TCP connection? Instead, would it not be possible for each client to simply open a *single* TCP connection to a one-address CDN and use that connection to issue all of its requests? This could have upside in terms of simplicity and performance: the general understanding is that maintaining a single connection yields better performance than opening multiple, short-lived ones. Moreover, there are ongoing efforts to be able to negotiate distinct TLS sessions over a given TCP connection. However, there are some potential concerns: TCP connections offer a convenient unit of isolation across different customers of the CDN. An interesting area of future work would be to provide stronger isolation guarantees above the transport layer.

6 DISCUSSION: LOOKING AHEAD

Perhaps the most exciting aspect of our architecture is that it opens pathways to new agile systems designed for added visibility into outside network effects and behaviours, as well as for granularity of control. In this section we describe initial ideas that, while concrete, abstract away many details and are without implementation. All are currently being pursued or under investigation, and expected to appear as future work. Each has helped to elucidate our architecture's merit – and the value of decoupling IPs from names and sockets.

For ease of presentation we sketch each system assuming one-address datacenters, unless stated otherwise. We also emphasize that in each of these designs *a single-address set for some policy is functionally equivalent to a random selection from any disjoint set*. For example, an IPv4 /32 may be comfortably substituted with a /28.

Route Leak Detection and Mitigation for Anycast Figure 9 describes a route leak on an anycast system based on actual events. The timely detection and mitigation of traffic misdirection is currently one of the hardest issues in network operations. In an anycast network, addressing agility offers a foundation for fast route leak detection and mitigation, and starts with the observation that different PoPs can return distinct IPs within a single anycast prefix.

We illustrate by example. Assume all PoPs in the network advertise and accept connections on the same /24 (as shown in §4.3). A policy can be expressed in DNS so that each PoP expects to receive traffic on a unique address. Say the policy returns `*.*.*.25` in response to all queries routed to PoP-A, similarly `*.*.*.26` for all queries routed to nearby PoP-B, and `*.*.*.78` for PoP-X that is far from both A and B. As a consequence all or most of the ensuing request traffic at each PoP should arrive on its corresponding IP. PoP-A may see a small amount of traffic arrive on `*.26`, but should see none on `*.78`. Conversely, PoP-X should expect little-to-no traffic on `*.25` or `*.26`. A violation of these expectations in either direction is an indication of traffic misdirection. We expect network issues to be visible at DNS TTL timescales.

In this setup, our architecture makes mitigation trivial: Keep the policy, but change the prefix. Observe that the policy in this case is unchanged, since only the `*.*.*` portion is different. If the mitigation prefix is already advertised and known to the Internet, then mitigation is complete also at DNS TTL timescales.

Denial of Service (DoS) Mitigation at the Speed of TTLs An important ability in DoS mitigation is to be able to distinguish the attack vector as layer 3/4 (e.g., UDP or SYN floods) or layer 7 (named services). One of the tools for diagnosis, as well as mitigation, is to re-bind a named service to a different IP address.

Addressing agility enables an otherwise naïve $k-$ary search as a fast, powerful foundation for mitigation. For ease of presentation, start with all n services sit behind one address (or narrow prefix) in a range. For example, `a.b.c.1/32` from `a.b.c/24`, then proceed:

(1) An attack is detected; set DNS TTL to small value, t.

Begin an $k-$ary search:

(2) Partition n affected services randomly into k disjoint sets, each of size $\lceil \frac{n}{k} \rceil$;

(3) Map each set to the i^{th} address in the range where $i \in \{0..k-1\}$, e.g., for $k = 32$, place $\frac{n}{k}$ services on each of `a.b.c.1`, `a.b.c.2`, ..., `a.b.c.32`.

If the attack follows a slice then there is a named target; repeat from (2) on the affected slice. Otherwise the attack continues on the starting address, meaning that it is layer 3/4. Assuming DNS caches respect TTL values, then the worst case time to isolate the attack from services is TTL+$t(\log_k n)$.

Measurement Opportunities Initially, our live deployment was limited to a single medium datacenter, `DC1` for testing and debugging. To ensure reachability in the event of a failure, the BGP prefix p used in our test was also advertised from a datacenter, `DC2`, 600km away. Accordingly, web services at `DC2` were configured to accept connections on p addresses, but DNS at `DC2` was unaltered and returned addresses in accordance with the rest of the world. In other words, `DC2` had little expectation of ingress traffic via p addresses.

Despite `DC2`'s intended purpose as a failover, `DC2` received significant legitimate traffic on the IP addresses that could only be learned via DNS queries to `DC1`. We hypothesize that this happens because the DNS queries of some clients closest to `DC2` are handled by ISP resolvers that are closest to `DC1`. Surprisingly, the proportion of affected traffic was substantially higher for IPv6 than for IPv4.

This opens new and otherwise infeasible measurement directions, for example, elucidating Internet routing paths and maps. These are opportunities are left for future investigation.

Traffic tuning across anycast datacenters The route leak detection and mitigation mechanism described above gives evidence that addressing agility could be used to build route leak detection and mitigation for anycast networks. We posit that similar mechanisms await that can shift traffic between datacenters in an anycast network using map-colouring [27]. In this use case a colour is equivalent to a BGP prefix announcement, such that each datacenter in an anycast network advertises only one colour (or prefix) from the set. Aforementioned measurements may help to identify the smallest number of colours needed to achieve some property, for example, region isolation or traffic tuning zones with nearby datacenters.

7 RELATED WORK

Decoupling Content Names from Hostnames Our proposal that merges multiple hostnames into a single IP address leveraging anycast is very similar to a wide range of work that seeks to decouple *what* content is from *where* it is hosted.

One broad initiative to this end was the set of clean-slate network architectures from the early 2000s to address concerns such as the impending shortage of IPv4 addresses and poor IP-based support for mobility, replicated content, and middleboxes. DONA [40], TRIAD [29], IPNL [23], FARA [11], HIP [36], DOA [6, 69], *i3* [64], and SNF [37] are but a handful of example systems that demonstrated it was possible to decouple hostnames from content names. Most of these architectures require extensive changes to the existing network, some even taking a clean-slate approach. Conversely, we sought to achieve many of the same goals in today's Internet, without any additional deployments at clients or routers.

A more recent initiative that decouples content from location is Information-Centric Networking (ICN) [18, 19, 24, 35]. Prominent examples are Name Data Networking (NDN) [71], Network of Information (NetInf) [20], and the eXpressive Internet Architecture (XIA) [30]. These clean-state network architectures route based on the name of the content by completely removing host identifiers.

Today's CDNs can be viewed as a sort of ICN [35]: by employing aggressive caching, geo-replication, and traffic engineering, virtually any of a CDN's PoPs can be used to service a user's request. However, none decouple names and addresses to the extent that ICN architectures do; CDNs still use a large number of IP addresses, effectively resulting in a few content names mapping to any given IP address. While there are various designs that facilitate the deployment of ICN systems (such as the use of overlays), they are still not compatible enough with the current existing Internet architecture to have met our operational needs.

We believe our system can be viewed as an evolutionary step towards ICN that is (surprisingly) deployable today; our design would comprise only one IP address per ASN but can still provide service and content regardless of which hosts end-users communicate with. Of course, the scope of our problem is also considerably smaller than these drastic redesigns of the Internet. Where they sought to address naming concerns for *all* hosts and *all* protocols, we seek the far more modest goal of serving our customers' content on machines *we* control. Moreover, we seek only to operate over a small set of protocols (HTTP and HTTPS) that already permit transmitting content names (through Host headers and SNI). As a result, we did not have to design new packet headers or architectural components to translate between namespaces like many of these prior works [23, 29, 40, 64]: standard HTTP(S) clients already include them.

One trade-off prior architectures faced was between human-understandable names but less efficient to route (e.g., FQDNs), and being opaque (e.g., the flat namespace of a DHT is unhelpful to users) but more efficient to route [10, 25, 39, 64, 68]. Our system makes no such tradeoff; we maintain the namespace familiar to users (FQDNs) and still achieve highly optimized performance. This, too, is because our system need not provide an Internet-wide solution.

IP Randomization and Re-use One study shows surprising variability in IPv4 usage and activity varies across measures of reachability and load, as well as inferring evidence of restructuring [53]. There exist multiple attempts to improve multiplexing over IP, or re-imagining addressing all together. One example explores combinations of NAT, DNS, and DHCP within datacenters to multiplex a large number of execution environments onto a smaller number of IP addresses [61]. A more radical proposition reimagines addressing entirely with variable length addresses [51] or adds an additional layer on top of the network layer to remove IP bindings [23, 29, 40, 58]. Our design is more immediately deployable and, as above, is made possible by the fact that we are able to make several simplifying assumptions as compared to work that seeks to rearchitect the entire Internet. That said, our hope is that by demonstrating that even one significant step towards these loftier goals can be accomplished in a performance-sensitive production environment will breathe renewed interest into these fascinating endeavors.

8 CONCLUDING REMARKS

In this paper, we have presented the design, implementation, and experiences from an operational deployment of an architecture that drastically improves addressing agility of CDN services at scale. We use the architecture to bind addresses at query time, in response to attributes rather than names, and change bindings with every query. Our architecture is reminiscent of more audacious redesigns of Internet naming in that it decouples IP addresses (location) from hostnames (content), but because our architecture need not apply to the entirety of the Internet nor to all protocols, we were able to leverage existing application-layer mechanisms (like SNI), facilitating immediate deployment. Our year-long deployment shows that few IP addresses are needed and that, by disassociating addresses from names and sockets, just a few addresses can lead to exciting new applications like route leak detection and greater resilience to attack.

ACKNOWLEDGMENTS

We thank the anonymous reviewers and our shepherd, Anja Feldmann, for their helpful feedback. Special thanks are due to Tom Arnfeld, Wesley Evans, Jérôme Fleury, John Graham-Cumming, Ólafur Guðmundsson, Martin Levy, Kari Linder, Justin Paine, Ethan Park, Edo Ryker, Alissa Starzak, Tom Strickx, Luke Valenta, David Wragg, and more, whose collective input enabled or informed this work. This work was supported in part by NSF grants CNS-1900879, CNS-1901325, CNS-2051166, and CNS-2053363.

REFERENCES

[1] Akamai IP Ranges. https://bgp.he.net/search?search%5Bsearch%5D=Akamai&commit=Search.

[2] AWS: Multivalue Answer Routing. https://aws.amazon.com/premiumsupport/knowledge-center/multivalue-versus-simple-policies/. Last access: 2020/06.

[3] Amazon. AWS IP address ranges. https://docs.aws.amazon.com/general/latest/gr/aws-ip-ranges.html.

[4] Apple, Inc. Network Framework. https://developer.apple.com/documentation/network/, June 2020.

[5] J. T. Araujo. Addressing IPv6: A CDN Perspective. https://ripe74.ripe.net/presentations/presentation-archive/, May 2017.

[6] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish. A Layered Naming Architecture for the Internet. In *ACM SIGCOMM*, 2004.

[7] M. Belshe, R. Peon, and M. Thomson. Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540, May 2015.

[8] M. Bishop. Hypertext Transfer Protocol Version 3 (HTTP/3). Internet-Draft draft-ietf-quic-http-29, Internet Engineering Task Force, June 2020. Work in Progress.

[9] B. Briscoe. RFC1794: DNS Support for Load Balancing. https://tools.ietf.org/html/rfc1794, Apr 1995.

[10] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, I. Stoica, and S. Shenker. ROFL: Routing on Flat Labels. In *ACM SIGCOMM*, 2006.

[11] D. Clark, R. Braden, A. Falk, and V. Pingali. FARA: Reorganizing the Addressing Architecture. In *ACM SIGCOMM Workshop on Future Directions in Network Architectures (FDNA)*, 2003.

[12] Cloudflare. IP Ranges. https://www.cloudflare.com/ips/.

[13] Cloudflare Blog. What is round-robin DNS? https://www.cloudflare.com/learning/dns/glossary/round-robin-dns/. Last access: 2020/06.

[14] Cloudflare Blog. Fixing reachability to 1.1.1.1, globally. https://blog.cloudflare.com/fixing-reachability-to-1-1-1-1-globally/, 2018.

[15] Cloudflare Blog. The Technical Challenges of Building Cloudflare WARP. https://blog.cloudflare.com/warp-technical-challenges/, October 2019.

[16] Cloudflare Technical Documentation. Network Ports Compatible with Cloudflare. https://support.cloudflare.com/hc/en-us/articles/200169156-Identifying-network-ports-compatible-with-Cloudflare-s-proxy.

[17] Cloudflare Technical Documentation. What Is A Reverse Proxy? Proxy Servers Explained. https://www.cloudflare.com/en-gb/learning/cdn/glossary/reverse-proxy/.

[18] M. D'Ambrosio, C. Dannewitz, H. Karl, and V. Vercellone. MDHT: A Hierarchical Name Resolution Service for Information-centric Networks. In *ACM SIGCOMM Workshop on Information-Centric Networking (ICN)*, 2011.

[19] M. D'Ambrosio, P. Fasano, M. Marchisio, V. Vercellone, and M. Ullio. Providing Data Dissemination Services in the Future Internet. In *IEEE Global Communications Conference (GLOBECOM)*, 2008.

[20] C. Dannewitz, D. Kutscher, B. Ohlman, S. Farrell, B. Ahlgren, and H. Karl. Network of information (netinf)–an information-centric networking architecture. *Computer Communications*, 36(7):721–735, 2013.

[21] E. Dumazet and D. S. Miller. inet: add IP_BIND_ADDRESS_NO_PORT to overcome bind(0) limitations. https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=90c337da1524863838658078ec34241f45d8394d, June 2015.

[22] D. E. Eisenbud, C. Yi, C. Contavalli, C. Smith, R. Kononov, E. Mann-Hielscher, A. Cilingiroglu, B. Cheyney, W. Shang, and J. D. Hosein. Maglev: A Fast and Reliable Software Network Load Balancer. In *Symposium on Networked Systems Design and Implementation (NSDI)*, 2016.

[23] P. Francis and R. Gummadi. IPNL: A NAT-Extended Internet Architecture. In *ACM SIGCOMM*, 2001.

[24] A. Ghodsi, T. Koponen, B. Raghavan, S. Shenker, A. Singla, and J. Wilcox. Information-Centric Networking: Seeing the Forest for the Trees. In *Workshop on Hot Topics in Networks (HotNets)*, 2011.

[25] A. Ghodsi, T. Koponen, J. Rajahalme, P. Sarolahti, and S. Shenker. Naming in Content-Oriented Architectures. In *ACM SIGCOMM Workshop on Information-Centric Networking (ICN)*, 2011.

[26] V. Giotsas, I. Livadariu, and P. Gigis. A First Look at the Misuse and Abuse of the IPv4 Transfer Market. In *Passive and Active Measurement (PAM)*. Springer International Publishing, 2020.

[27] G. Gonthier. Formal proof–the four-color theorem. *Notices of the AMS*, 55(11):1382–1393, 2008.

[28] B. Gregg. Performance Superpowers with Enhanced BPF. Santa Clara, CA, July 2017. USENIX Association.

[29] M. Gritter and D. R. Cheriton. An Architecture for Content Routing Support in the Internet. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, USITS'01, USA, 2001. USENIX Association.

[30] D. Han, A. Anand, F. Dogar, B. Li, H. Lim, M. Machado, A. Mukundan, W. Wu, A. Akella, D. G. Andersen, et al. {XIA}: Efficient Support for Evolvable Internetworking. In *Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*, 2012.

[31] N. P. Hoang, A. A. Niaki, M. Polychronakis, and P. Gill. The Web is Still Small after More than a Decade. *ACM SIGCOMM Computer Communication Review (CCR)*, 50(2):24–31, May 2020.

[32] R. Holz, J. Hiller, J. Amann, A. Razaghpanah, T. Jost, N. Vallina-Rodriguez, and O. Hohlfeld. Tracking the Deployment of TLS 1.3 on the Web: A Story of Experimentation and Centralization. *SIGCOMM Comput. Commun. Rev.*, 50(3):3–15, July 2020.

[33] G. Huston. The Architecture of the Internet or Waist Watching in IP. APNIC Labs Presentation Archive, https://labs.apnic.net/presentations/store/2004-05-04-waistwatching.pdf, May 2004.

[34] J. Iyengar and M. Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. https://tools.ietf.org/html/draft-ietf-quic-transport-28#section-9.6, May 2020.

[35] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking Named Content. In *Proceedings of the ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '09, New York, NY, USA, 2009. Association for Computing Machinery.

[36] P. Jokela, P. Nikander, J. Melen, J. Ylitalo, and J. Wall. Host Identity Protocol - Extended Abstract. In *Wireless World Research Forum*, 2004.

[37] A. Jonsson, M. Folke, and B. Ahlgren. The Split Naming/Forwarding Network Architecture. In *Swedish National Computer Networking Workshop*, 2003.

[38] Juniper Networks. What's New in Release 19.2R2: Routing Protocols. https://www.juniper.net/documentation/en_US/junos/information-products/topic-collections/release-notes/19.2/topic-147567.html#rn-junos-qfx-new-and-changed-features, May 2020.

[39] C. Kim, M. Caesar, and J. Rexford. Floodless in SEATTLE: A Scalable Ethernet Architecture for Large Enterprises. In *ACM SIGCOMM*, 2008.

[40] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A Data-Oriented (and Beyond) Network Architecture. In *ACM SIGCOMM*, 2007.

[41] J. Larisch, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson. CRLite: A Scalable System for Pushing All TLS Revocations to All Browsers. In *IEEE Symposium on Security and Privacy*, 2017.

[42] Linux Foundation. *ip(7) — Linux manual page, see entry for IP_BIND_ADDRESS_NO_PORT*. Linux Programmer's Manual, 2021.

[43] BPF `sk_lookup` merge commit. Linux kernel 5.9, https://github.com/torvalds/linux/commit/e57892f50a07953053dcb1e0c9431197e569c258, 2020.

[44] Support for running bpf programs on socket lookups. Linux kernel 5.9, https://kernelnewbies.org/Linux_5.9#Support_for_running_BPF_programs_on_socket_lookups, December 2020.

[45] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu. SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs. In *ACM SIGCOMM*, 2017.

[46] G. Moura. DNS TTL Violations in the Wild - Measured with RIPE Atlas. https://labs.ripe.net/Members/giovane_moura/dns-ttl-violations-in-the-wild-with-ripe-atlas-2, Dec 2017.

[47] G. C. M. Moura, J. Heidemann, R. d. O. Schmidt, and W. Hardaker. Cache Me If You Can: Effects of DNS Time-to-Live. In *ACM Internet Measurement Conference (IMC)*, 2019.

[48] U. Naseer, L. Niccolini, U. Pant, A. Frindell, R. Dasineni, and T. A. Benson. Zero Downtime Release: Disruption-Free Load Balancing of a Multi-Billion User Website. In *ACM SIGGCOMM*, 2020.

[49] E. Nygren. Reaching Toward Universal TLS SNI. https://blogs.akamai.com/2017/03/reaching-toward-universal-tls-sni.html, 2017.

[50] T. Pauly, B. Trammell, A. Brunstrom, G. Fairhurst, C. Perkins, P. S. Tiesel, and C. A. Wood. An Architecture for Transport Services. Internet-Draft draft-ietf-taps-arch-10, Internet Engineering Task Force, Apr. 2021. Work in Progress.

[51] S. Ren, D. Yu, G. Li, S. Hu, Y. Tian, X. Gong, and R. Moskowitz. Routing and Addressing with Length Variable IP Address. In *Proceedings of the ACM SIGCOMM 2019 Workshop on Networking for Emerging Applications and Technologies*, NEAT'19, pages 43–48, New York, NY, USA, 2019. Association for Computing Machinery.

[52] RFC 791: Internet Protocol. https://datatracker.ietf.org/doc/html/rfc791, Sept. 1981.

[53] P. Richter, G. Smaragdakis, D. Plonka, and A. Berger. Beyond Counting: New Perspectives on the Active IPv4 Address Space. In *ACM Internet Measurement Conference (IMC)*, 2016.

[54] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2(4):277–288, Nov. 1984.

[55] F. W. Scholz and M. A. Stephens. K-sample Anderson–Darling tests. *Journal of the American Statistical Association*, 82(399):918–924, 1987.

[56] K. Schomp and R. Al-Dalky. Partitioning the Internet Using Anycast Catchments. *SIGCOMM Comput. Commun. Rev.*, 50(4):3–9, Oct. 2020.

[57] K. Schomp, O. Bhardwaj, E. Kurdoglu, M. Muhaimen, and R. K. Sitaraman. Akamai DNS: Providing Authoritative Answers to the World's Queries. In *ACM SIGCOMM*, 2020.

[58] S. Sevilla and J. J. Garcia-Luna-Aceves. Freeing the IP Internet Architecture from Fixed IP Addresses. In *IEEE International Conference on Network Protocols (ICNP)*, 2015.

[59] J. F. Shoch. A note on Inter-Network Naming, Addressing, and Routing. IEN 19, 1978.

[60] M. Silverlock and G. Redner. Bringing Modern Transport Security to Google Cloud with TLS 1.3. https://cloud.google.com/blog/products/networking/tls-1-3-is-now-on-by-default-for-google-cloud-services, June 2020.

[61] R. P. Singh, T. Brecht, and S. Keshav. IP Address Multiplexing for VEEs. *SIGCOMM Comput. Commun. Rev.*, 44(2):36–43, Apr. 2014.

[62] J. Sitnicki. BPF sk_lookup - TCP SYN and UDP 0-len flood benchmarks. https://lore.kernel.org/bpf/87lficrm2v.fsf@cloudflare.com/, Aug 2020.

[63] J. Sitnicki. Run a BPF program on socket lookup. https://lore.kernel.org/bpf/20200506125514.1020829-1-jakub@cloudflare.com/, May 2020.

[64] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet Indirection Infrastructure. In *ACM SIGCOMM*, 2002.

[65] S. Sundaresan, N. Feamster, R. Teixeira, and N. Magharei. Measuring and Mitigating Web Performance Bottlenecks in Broadband Access Networks. In *ACM Internet Measurement Conference (IMC)*, 2013.

[66] B. Trammell, C. Perkins, T. Pauly, M. Kühlewind, and C. A. Wood. Post Sockets, An Abstract Programming Interface for the Transport Layer. Internet-Draft draft-trammell-taps-post-sockets-03, Internet Engineering Task Force, Oct. 2017. Work in Progress.

[67] W3Techs Web Technology Surveys. Usage statistics and market share of cloudflare. https://w3techs.com/technologies/details/cn-cloudflare last accessed 06/2021.

[68] M. Walfish, H. Balakrishnan, and S. Shenker. Untangling the Web from DNS. In *Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.

[69] M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, and S. Shenker. Middleboxes No Longer Considered Harmful. In *Symposium on Operating Systems Design and Implementation (OSDI)*, 2004.

[70] J. M. Winett. Definition of a socket. RFC 147, https://rfc-editor.org/rfc/rfc147.txt, May 1971.

[71] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, k. claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang. Named Data Networking. *ACM SIGCOMM Computer Communication Review (CCR)*, 44(3):66–73, July 2014.

[72] J. Žorž, S. Steffann, P. Dražumerič, M. Townsley, A. Alston, G. Doering, J. Palet, J. Linkova, L. Balbinot, K. Meynell, and L. Howard. Best Current Operational Practice for Operators: IPv6 prefix assignment for end-users - persistent vs non-persistent, and what size to choose. https://www.ripe.net/publications/docs/ripe-690, October 2017.