An Architecture For Edge Networking Services

Lloyd Brown[1], Emily Marx[1], Dev Bali[1], Emmanuel Amaro[2], Debnil Sur[3], Ezra Kissel[4], Inder Monga[5], Ethan Katz-Bassett[6], Arvind Krishnamurthy[7], James McCauley[8], Tejas N. Narechania[1], Aurojit Panda[9], Scott Shenker[1,10]
[1] UC Berkeley, [2] Microsoft, [3] VMware Research, [4] LBL, [5] ESNet, [6] Columbia University, [7] UWashington, [8] Mount Holyoke College, [9] NYU, [10] ICSI

Abstract

The layered Internet architecture, while far from perfect, has provided a global and neutral platform for the development of a wide range of applications. However, this core architecture has been increasingly augmented with additional in-network functionality that improves the performance, security, and privacy of these applications. These additional in-network functions, which are typically implemented at the network edge, are consistent with the layering of the Internet architecture but deviate from two of the core tenets of the Internet: interconnection and end-to-end simplicity. In this paper, we propose an architecture for these edge networking services called the InterEdge that applies these two Internet tenets in a manner appropriate to edge services while not requiring changes to the underlying Internet architecture or infrastructure.

CCS Concepts

• **Networks → Network architectures**.

Keywords

Internet architecture, Edge networking services

ACM Reference Format:
Lloyd Brown, Emily Marx, Dev Bali, Emmanuel Amaro, Debnil Sur, Ezra Kissel, Inder Monga, Ethan Katz-Bassett, Arvind Krishnamurthy, James McCauley, Tejas N. Narechania, Aurojit Panda, Scott Shenker. 2024. An Architecture For Edge Networking Services. In *ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24), August 4–8, 2024, Sydney, NSW, Australia.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3651890.3672261

1 Introduction

1.1 Context

The Internet's architecture is typically described in terms of its layered design, whose modularity has allowed the Internet to accommodate radical changes in scale, scope, and usage over the past thirty-plus years. However, at a more conceptual level, the architecture is built on two crucial tenets that are neither the consequence of, nor directly imply, this layering: *interconnection* and *end-to-end simplicity* (E2ES).

The notion of interconnection, broadly construed, turned a collection of local packet-switching networks into the Internet of today. This interconnection happened in three logical steps (though, of course, the actual history is more complicated than the simple conceptual picture we describe here). First, the notion of a universal "internetworking protocol" (IP) allowed the interconnection of the many heterogeneous packet networks that had arisen in the 1960s and 1970s. Second, as the single uniform infrastructure broke into separate autonomous systems (ASes), it was necessary to create an interdomain routing protocol, which has now evolved into BGP. Lastly, when Internet Service Providers (ISPs) went commercial, a set of financial peering arrangements arose so that money could recursively flow to the networks carrying traffic. Together, these forms of interconnection allowed the Internet to provide a uniform (via IP), global (via BGP), and economically viable (via peering payments) platform without any individual ISP being global. Note that we do not use the term "global" to imply that everyone has access to the Internet; our point is merely that Internet coverage spans a large portion of the globe and does so without requiring any single ISP to have global coverage.

Turning to the second tenet, by E2ES we mean a design philosophy where the Internet only provides best-effort packet delivery and leaves all additional functionality – particularly application-level functionality – to the communicating endpoints. This simplicity is related to the vaunted End-to-End Principle (E2EP), but the E2EP is far more subtle about the criteria for when in-network functionality is appropriate and is the subject of much commentary (the canonical reference is [57] but see [17] for additional discussion). The early Internet embodied E2ES, and the fact that the early Internet was built around IP on the interdomain dataplane and BGP on the interdomain control plane – two protocols that have no specific application-level considerations in their designs – made interconnection easier.

While the E2EP started out as an engineering design principle, it (and its more simplistic interpretation, E2ES) gave rise to an economic corollary: network neutrality. The tie between the two is best captured in [39]: "One of the most basic arguments in favor of Net Neutrality is that it is needed in order to preserve what is known as the 'end-to-end principle'." Similarly, Tim Wu, one of the originators of the idea of network neutrality, states in [67]: "in networking, the 'end-to-end' principle of network design is also a close cousin, if not the direct ancestor of network neutrality." The reasoning behind these statements is that just as the E2EP/E2ES (we will use this pair of acronyms to refer to the general constellation of end-to-end ideas) keeps the technical aspects of application-level functionality out of the network, network neutrality keeps the economic aspects of applications out of the network. Thus, this corollary to the E2EP/E2ES makes the Internet not just a functionally simple platform but an economically neutral one as well, by requiring that the IP layer not unfairly discriminate between packets based on their higher layers (where the definition of "unfairly" differs between different legal interpretations of network neutrality).

ACM SIGCOMM '24, August 4–8, 2024, Sydney, NSW, Australia

ACM ISBN 979-8-4007-0614-1/24/08
https://doi.org/10.1145/3651890.3672261

This combination of clean separation vertically between the network and end-to-end applications (to keep the network simple and neutral) but interconnection horizontally (to allow global coverage to be created out of local ISPs) was crucial to the Internet's rapid and enduring success. But in the past two decades, the Internet's infrastructure has evolved significantly to meet emerging application requirements, and as we describe below, the resulting changes have not been guided by these two fundamental tenets.

1.2 The End of End-to-End Simplicity

The network infrastructure now applies a variety of in-network "enhancements" that involve more than just packet forwarding, and these additional functions are typically applied at the network edge. We give many examples below, but for specificity, think of caching (as in CDNs), authentication (as in zero-trust network access or ZTNA), or edge routing (as in SD-WANs). Some of these edge networking services are strong violations of E2ES, in that some nodes in the network infrastructure are interposing application-level (L7) functionality between the two endpoints of a connection. Note that this is not a violation of the layered architecture (as happens with transparent firewalls), but rather a departure from the client-server model to something more akin to a client-edge(s)-server model.

This departure was driven by the changing role of the Internet, with the edge networking services typically addressing performance, security, and privacy issues that were not crucial in the early days of the Internet but which have become major pain points now that the Internet plays such a central role in our personal and commercial lives. For instance, "good enough" performance is no longer sufficient because small changes in end-to-end delays have large economic ramifications (*e.g.,* lower response latencies cause users to stay online longer, which in turn produces more ad revenue [65]). Caching (as in CDNs) was the first widespread performance enhancement, but the Internet now has several others, including streaming support, SD-WAN, load balancing, and storage [53].

Similarly, security has become far more important in a highly commercial Internet. Many decry the insecurity of the Internet itself, and it certainly has weaknesses – such as its vulnerability to denial-of-service attacks and route hijacking – but the security-related edge networking services are mostly used to strengthen the weak security of endpoints. Examples of this include ZTNA, in-network next-generation firewalls (NGFWs), and authentication services such as described in [14].

In addition, as the Internet has penetrated ever more deeply into our personal lives, privacy has become far more important. Privacy concerns have led to the ubiquitous use of encryption to protect payloads, and to the deployment of edge networking services such as private relay [5] and oDNS [58] that help hide traffic patterns and metadata from the infrastructure.

The infrastructures offering these enhancements are not required to apply them neutrally, in that there are no equivalents of the network neutrality regulations that apply to these higher-level in-network services. For instance, there is a new class of global private networks operated by hyperscaled cloud and application providers (that we discuss in Appendix A). While these networks have an extensive set of edge networking services (*e.g.,* DDoS protection, load balancing, caching, and flow termination) these are only used for their own traffic or (in some cases) their customers' traffic. In addition, there are commercial providers of edge networking services (we will call them edge service providers, or ESPs), such as Fastly, Cloudflare, and Akamai. However, there is no explicit legal requirement nor historical practice that requires that these ESPs provide services to all their customers on an equal basis (*i.e.,* they would be within their rights to discriminate against some customers by charging more or denying service altogether). As we discuss in Section 2, perhaps the greatest impact of the loss of E2ES is the demise of neutrality.

A quick note about terminology: while ESPs and the private networks are different in how they offer these services to others, they have very similar technical designs. In what follows, we will typically refer only to ESPs, but when the context is about their design, the comments will also typically apply to the private networks of hyperscaled cloud and application providers.

1.3 The End of Interconnection

While these edge networking services are valuable – indeed, they are necessary to handle the current workloads of applications like content delivery – they are not interconnected. By this, we mean that different ESPs do not coordinate with each other to deliver edge services in the same way ISPs coordinate with each other to ensure packet delivery. That is, even though they often peer at the IP level (allowing them to exchange IP traffic), ESPs do not peer at the edge service level and coordinate their handling of service requests. To make this concrete, consider a set of established ESPs (*e.g.,* providing ZTNA) that do not cover region A. If a new ESP emerges to cover region A, it cannot merely "interconnect" with an existing ESP to provide coverage to region A. Instead, the entities arranging for the edge networking service (*i.e.,* the customers of the ESPs) have to individually contract with this new ESP to gain coverage for region A. This is as if the Internet did not have BGP but instead required each AS that is sending traffic to provide source routes to reach their destinations and pay the ISPs along the paths.

This lack of interconnection is not due to an accident or oversight but has happened because none of the necessary ingredients that led to the Internet's interconnection – standards, routing, and peering – are present for these edge networking services. To clarify what we mean by the lack of standards, note that edge networking services typically have to be consistent with application-level protocols used by the endpoints (*e.g.,* TLS or HTTP). However, the implementations of edge networking services do not have to be consistent with each other, and interprovider consistency is necessary for interconnection; *i.e.,* the ESPs cannot coordinate on the delivery of edge networking services unless they agree on the semantics of those services. It is true that for some well-established services, such as CDNs, interconnection standards exist (*e.g.,* CDNI [9], Open Caching [12], and Federated Edge [68] have been developed, though not widely deployed), but in general there is no requirement for edge networking services offered by different ESPs to have the necessary level of compatibility.

Related to the lack of interconnection is the fact that ESPs do not need to allow other ESPs to resell their services; they might choose to do so, but there is no requirement. In contrast, the whole

point of interconnection in the Internet is that when an access ISP A connects to a transit ISP B, A is, by definition, reselling the connectivity of B.

As a result of this lack of interconnection, global ESP coverage requires either that (i) ESP infrastructures be global, or (ii) ESP customers stitch together coverage by several ESPs, tolerating whatever differences in semantics or configuration exist between those ESPs (where semantics refers to the behavior of the service itself, and configuration is how customers specify the parameters of the service to the ESP). Neither of these are attractive solutions: the former creates significant barriers to entry into the ESP market, and the latter imposes a heavy burden on ESP customers. Some customers, such as in the CDN space, do take on the burden of contracting with several ESPs, but our point here is not that the burden is impossible to overcome but that it is undesirable and (as we will argue) unnecessary in a well-designed architecture.

1.4 The Goal of This Paper

Summarizing the status quo, the Internet has a time-tested, if imperfect, architecture at L3 built around the tenets of interconnection and E2ES. In recent years, the Internet infrastructure has been augmented with an *ad hoc* collection of interposed networking services at the edge that do not obey either tenet. The purpose of this paper is to identify a coherent architecture for edge networking services that tries to be more consistent with these two guiding tenets of the Internet. We call the resulting design the InterEdge.

Note that the InterEdge does not, in any way, alter the current Internet architecture; it merely provides an organizing structure for the additional edge networking functions that currently have no place in the Internet architecture. In addition, we are not claiming that the world cannot exist without such an architecture; in fact, the world of edge networking services clearly does exist! Nor are we claiming that such an architecture would be easily adopted. Instead, we are merely noting that the research literature has never explored what a clean architecture for these edge networking services might look like, and our goal here is to describe one such architecture.

In the next section, we discuss three topics: related work, why we might need an architecture for edge networking services, and the technical challenges posed by such an architecture. Next, in Section 3, we give an overview of the InterEdge's basic architecture. In Sections 4–6, we describe how the InterEdge can support interposition, interconnection and neutrality, and applications. We consider the potential impact of InterEdge on the Internet ecosystem and the possible incentives for adopting the InterEdge in Section 7.

2 Related Work, Rationale, and Challenges

2.1 Related Work

The research community has produced a wealth of clean-slate designs for new Internet architectures (see [18] for an overview). These design exercises have added to our understanding in many areas, such as quality-of-service [19, 50], content delivery [40, 69], evolvability [22, 37, 45, 54], security [70], mobility [59], and resiliency [3] (where, in each case, we have cited only a few of the many relevant works). In contrast, in this paper, we accept the Internet architecture as given and are merely trying to provide an architectural framework for the many edge networking services that have been deployed within the current Internet. The InterEdge proposal we present here is closely related to our earlier work on the Extensible Internet (EI); EI has been discussed in a previous short editorial [8] and a blog post [60], but no detailed technical description has been published.

But the InterEdge owes far more to real-world developments than to previous research results. There are many commercial deployments at the "edge" (for various definitions of that term) that have transformed how enterprises achieve higher performance and security on the Internet. In particular, there is now a rich ecosystem of ESPs, and many of the hyperscalers have deployed similar infrastructures for their own use (as described in Appendix A). This is a case, as with cloud computing, where the commercial world sprinted far ahead of the research community. In a recent editorial [10] we argued that these recent commercial developments pose a threat to the public Internet, but here our focus is on the technical design of a new architecture for the edge.

Our work only addresses edge deployments that are focused on network-related functionality rather than computation-related functionality, so what is commonly called "Edge Computing" is not within our purview. We use the term networking-related to refer to functionality whose behavior and resource consumption are easily characterized, whereas computation-related functionality offer users a wider range of capabilities and resources. Of course, the distinction between the two is not clearly defined, and as various computational tasks become more standardized (*e.g.,* model serving at the edge) they could eventually be considered networking-related (much as caching has made the transition to a networking-related service). Note that the true distinction here is less *networking versus computation* than *common and easily characterized primitives versus proprietary and/or complicated functionality*. Our goal is to make it easy to deploy these common primitives that in turn will allow the development of more sophisticated and performant applications.

From a conceptual standpoint, the work on which we depend the most is that of Blumenthal and Clark (the latter being one of the coauthors of the original end-to-end paper [57]), who argued in [11] that the E2EP must be reinterpreted for the modern Internet because the current lack of trust, diversity of end devices, and operator-driven requirements are important reasons for overriding the original principle and more aggressively inserting functionality in the network. Given this reinterpretation, how should we think about the current Internet's extensive use of edge networking functionality? Their seminal paper makes two crucial observations.

According to [11], the "ability to support new and unanticipated applications" at scale is the most important criterion when deciding where to place functionality (assuming, of course, that the semantics of the function can be safely implemented in the network, which was the major focus of the original end-to-end paper [57]). When the core forwarding function of the network was mostly implemented in hardware, this criterion called for moving most functions to endpoints, so that (i) hardware implementation of network forwarding remained feasible and (ii) the network's hardware datapath did not become cluttered with a set of complicated and possibly interfering functions. Most current ESPs use distributed software platforms where different services need not interfere with each other nor with traffic that does not need their functionality, so this concern no longer applies.

Blumenthal and Clark also express worries about "investment in closed islands of enhanced service" that may come with in-network functionality. This is a prescient critique of the status quo, with the private networks of the hyperscalers being prime examples of "closed islands." The purpose of the InterEdge is to broaden access to and increase the supply of these in-network services.

To summarize, much of the rationale for the E2EP has been undercut by a new set of concerns, and the simplicity of the early Internet has disappeared as a growing set of edge services have proven essential for better privacy, security, and performance. However, while neither the rationale for nor the reality of the Internet's technical simplicity still holds, the argument for neutrality remains valid. More specifically, even if some higher-level functionality has moved into the network, it is still important to keep the economic aspects of that functionality out of the network, so that applications can compete fairly with each other, rather than being explicitly favored by the infrastructure.

Thus, as mentioned earlier, we think that the most important concern raised by the end of the E2EP/E2ES is that the neutrality of the infrastructure is no longer ensured. There is nothing preventing the ESPs, or the private networks of the hyperscalers, from discriminating against specific customers. We discuss the implications of this in the next subsection, where we discuss why one needs an architecture for edge networking services.

2.2 Why Do We Need An Architecture Like InterEdge?

Any rationale for a new design must start with the question: *What is wrong with the status quo?* In answering this question, we compare the status quo with the alternative where the edge networking services are provided as part of an architecture in which the ESPs are interconnected (which would require standards) and neutral. In such a comparison, we have five main areas of concern: coverage, lock-in, innovation, integration, and long-term evolution.

Coverage: The fact that new ESPs cannot seamlessly interconnect with the existing ESPs means that it is harder to expand coverage to currently underserved areas (see [15] for current CDN coverage). To be clear, the lack of ESP coverage only means that the ESP infrastructure is further away, not that there is no Internet access. However, ESPs incur sizable expenses to provide relatively dense coverage in lucrative markets, so there must be significant customer benefit in doing so. Thus, just as the baseline has moved in the US from Internet access to high-bandwidth Internet access because some applications do not function well without high-bandwidth, we can expect the same will soon hold true for nearby ESP support.

Lock-in: As noted earlier, there are typically no assurances that the semantics or the configurations of the edge networking services of different ESPs are the same. The lack of standards for the basic service semantics (*e.g.,* different pub/sub services give different guarantees) leads to a strong form of lock-in, because an application built on one set of semantics cannot safely move to a weaker set of semantics. Even if the semantics of two edge networking services are the same, the configuration used to specify the desired behavior might not be. This leads to a weaker form of lock-in, where switching between ESPs involves substantial reconfiguration but not rewriting the application. There is currently much concern about similar weak and strong lock-in effects in the cloud computing market [16].

Innovation: It would be hard to argue that the current ESPs are not innovative, since the past ten years have seen a significant increase in the variety of edge networking service deployments. And one might fear that the requirement of standards, which would be necessary for interconnection, would slow this rapid innovation. But both points are more subtle than they first appear, as we now discuss.

While new services have been introduced, they are typically consistent with current application-level protocols (*e.g.,* HTTP) and do not require modifications to client software. The exceptions, such as the deployment of QUIC [42] (which runs on top of UDP) and private relay [5] (which involves an intermediate proxy), are cases where the same entity controlled the client, edge, and back-end deployments. This consistency with current application-level protocols is not because there are no benefits to new application-level protocols (in fact, the exceptional cases above show there are benefits) but because, for fear of lock-in, vendors of end-user software are reluctant to support an edge service requiring client modifications if it is not offered by several ESPs. Creating standards for such services would enable support by multiple ESPs, and therefore encourage the deployment of edge networking services that require client modifications.

However, the presence of a standardization process does not preclude ESPs from offering experimental services that are not yet standardized. If an ESP offers a new nonstandardized service in an open manner – *e.g.,* providing an open-source implementation that is consistent with the overall architecture (that we describe in Section 3) – customers can adopt it knowing that standardization (and therefore broader support) is a likely outcome if the service becomes popular.

Another factor inhibiting innovation is that the current ESPs typically only deploy services that can be easily monetized on their own: *i.e.,* can be charged for individually, and generate enough revenue to justify their deployment. This is not the way to build a generally useful platform, where often one needs a suite of services on which to build applications, not all of which generate enough revenue to justify a deployment infrastructure.

In the cloud ecosystem, open-source innovations in basic primitives (*e.g.,* key-value stores like Redis) can quickly become widely used, without needing to generate a specific revenue stream. This is the kind of innovation we hope to enable in the InterEdge; by encouraging a more dynamic ecosystem of open-source innovations that can more easily achieve wide deployment, the Internet will become a more powerful platform for applications.

Integration: As discussed in the Introduction, there is a wide range of edge networking services, and they are offered by a variety of ESPs. In many cases, it is up to the enterprise (for services like SD-WAN and ZTNA) or application provider (for CDN service and DDoS protection) to figure out how to make the various edge networking services work together (*i.e.,* so that an enterprise can simultaneously have both SD-WAN and ZTNA operating). The presence of an edge networking services architecture would not solve this integration problem completely but would certainly make the integration of network edge services easier for customers.

One might argue that the current edge networking services are in a stage similar to what Minitel [44] and AOL [4] and other early online systems were before the Internet arose. They provided extremely useful functionality, but were standalone systems; the breadth of their effectiveness was largely due to the size of the provider, not the synergy with other systems. That synergy came about only after the Internet provided a single coherent platform that made it easy to deploy new Internet applications. Achieving that synergy for edge networking services is the goal of InterEdge, and it requires that the platform be extensible (allowing the easy deployment of new services) and integrated (so users can avail themselves of several services within the same platform).

Long-term evolution: One test of a global platform is whether it will support the deployment of its successor. The telephony infrastructure gave rise to the Internet (by providing point-to-point communication). The Internet then gave rise to the hyperscalers by allowing their customers to access them remotely; the private infrastructure of these hyperscalers now sources the vast majority of Internet traffic reaching end users [41], often bypassing public transit providers to carry the traffic directly into access networks [6, 63]. The question is: can the current Internet, with its hyperscaled private networks and large-scale ESPs, support the next step in this evolution? More specifically, is the current platform general enough to technically support this evolution, and is it neutral enough to economically allow this evolution?

As stated previously, there is currently no requirement that ESPs must be neutral, so they can deny service to or demand additional revenue from customers as they see fit. Such actions could distort competition in application-level markets and potentially thwart innovations that may eventually threaten the dominance of the current incumbents, thereby limiting the possibilities for long-term evolution. In addition, a neutral Internet infrastructure that assumes every flow is between a client and a backend connected with one of the current hyperscaled incumbents is not general enough to support a new set of incumbents; instead, we need a neutral network that can support any-to-any communication rather than just client-to-incumbent communication. Thus, we are not optimistic that the Internet, on its current trajectory, could support or would allow the next generation of incumbents to arise.

2.3 Technical Challenges

The previous discussion described, in general terms, how an edge services architecture that provides interconnection and neutrality might improve on the status quo. Here, we try to identify what technical challenges we must confront in designing such an architecture.

The first class of challenges involves creating an architecture that is flexible, performant, and extensible. This is more about the overall structure of the architecture, and we describe it in Section 3.

The second set of challenges comes from the interposition of third-party edge networking services on connections, where "third parties" mean entities that are not associated with either end of the connection. Many ESPs today are third parties in this sense, so this use case is common today. However, today's application-level protocols do not deal with this case cleanly because trust and privacy are problematic when a third party is interposing on a protocol originally designed for client-server interactions. We address this in Section 4.

The third class of challenges arises from interconnection and neutrality. Interconnection requires standards, coordination (so that services are applied at the appropriate edge), and financial arrangements, so that money flows to the entities delivering services. Neutrality requires nondiscrimination. We discuss these challenges in Section 5.

3 Basic Architecture

In designing the InterEdge, we borrow the basic design patterns from current ESPs (because they have proven effective at scale) and only make changes to support interconnection and neutrality, which are the two new requirements we are placing on edge networking services (and the latter requires very few changes). ESPs use commodity compute clusters relatively near clients to implement their functionality; this positioning is essential to reduce latencies, and the use of general-purpose compute (as opposed to the ASICs in traditional networking gear) is essential to handle application-level functionality. The InterEdge follows this model, with all edge service processing implemented on service nodes (SNs) that are commodity clusters placed near endpoints (*i.e.,* at network edges, such as aggregation points, central offices, or PoPs). SNs are provided by what we will call InterEdge Service Providers (IESPs). Because the InterEdge is a standardized interconnecting architecture (as will become clear below), there will likely be far more IESPs than today's ESPs, and they can span from current ESPs (who might adopt InterEdge for at least some of their offerings) to carriers, cellular providers, and IXPs, and even to others in today's ecosystem that have not entered the current edge networking services business because of its relatively high barriers but who already have a physical presence at various network edges. The sizes of IESPs will vary greatly, from the global scale of today's large ESPs to smaller IESPs that serve communities that are currently underserved today.

In addition, as we describe in Section 5, the edge services provided by the IESPs can be paid for by either (i) the provider of the application being used, (ii) the provider of the content being consumed, or (iii) the owner of a host (where we use the term host in what follows to refer to endpoint devices, whether they be client or server in any particular connection).

We now describe the rest of the InterEdge architecture, starting with the basic components (see Figure 1), a few operational details, and some important properties.

3.1 Components

Interposition-Layer Protocol (ILP): All communication between SNs, and between hosts and SNs, use a new tunneling protocol called ILP. We describe ILP in more detail in Section 4.

Host-SN association: Every host is associated with one or more first-hop SNs; these are the InterEdge's analog of L3's first-hop routers, and handle both outgoing and incoming packets. The first-hop SNs of any IESP can be discovered using a variety of standard techniques (*e.g.,* configuration, anycast, lookup, etc.). The host will use whichever first-hop SN is appropriate for a given connection,

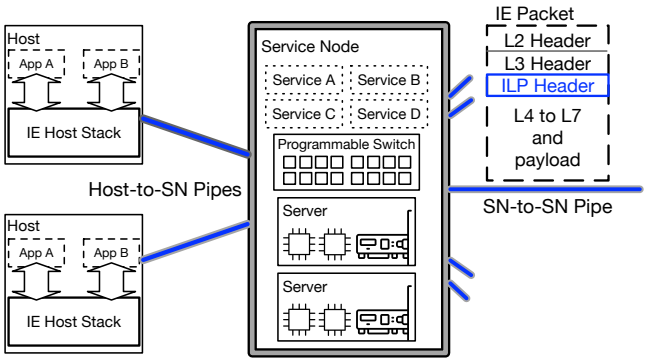

Figure 1: An overview of InterEdge components.

and that choice will often depend on who is paying for the edge services (since they will choose the IESP).

Service model: The InterEdge's service model – that is, the set of services any host accessing the InterEdge can count on – is defined by a set of standardized services implemented in the SNs. These services, examples of which we describe more fully in Section 6, are defined not by detailed written specifications but by evolving open-source software implementations called service modules. These services and their implementations are chosen by some governance body (such as the IETF) and deployed on all SNs, ensuring that the InterEdge's service model is uniformly available.

Execution environment: To make this uniform service deployment practical, all SNs have a common execution environment that supports a few basic primitives (such as sending and receiving packets over ILP, reading and updating configuration, and checkpointing state for fault tolerance) in addition to an extensible set of libraries that service modules can use for common tasks (*e.g.,* cryptography [56], regular expression matching [71], and video-and-audio re-encoding [46]). All service modules are written to this common execution environment, creating a Write-Once-Run-Anywhere (WORA) ecosystem for InterEdge services. We assume that SNs have TPMs that can be used for attestation. We discuss the use of secure enclaves in the SN in Section 6.

Datapath: Packets enter and leave an SN through what we call the pipe-terminus, and then packets can be sent for general processing within the SN's execution environment. One can think of the execution environment's datapath as having three components: a fast path (executed on the pipe-terminus); a slow path, which can provide more general processing on general compute servers; and off-path functions, such as access to persistent storage, that are substantially slower than packet forwarding. In the most basic case, the pipe-terminus sends packets to a server hosting a service module associated with the InterEdge service specified in ILP metadata. The pipe-terminus also has a decision cache that stores match-action pairs, enabling the pipe-terminus to take more general actions (such as forwarding the packet to a specified next-hop SN). This cache is populated by the service modules (either based on their configuration or after processing some initial packets in a connection). SNs can have a variety of hardware accelerators (*e.g.,* programmable switching, SmartNICs, cryptographic engines). These can be used, at the operator's discretion, to accelerate pipe-terminus actions and the execution of various libraries on servers. Service modules can also have alternative versions that directly leverage various accelerators when available, but service modules must have a basic version that only requires general compute support from an SN.

Host support: The InterEdge requires a host component that implements support for ILP. Additionally, the host component is also responsible for implementing client-side support for services – such as pub/sub, anycast and multicast – that require host logic.

Edomains: The InterEdge will consist of many autonomous domains of edge control, which we will call edomains. Each IESP will constitute one or more edomains, with global scale IESPs perhaps using multiple edomains to provide some geographic locality in how other IESPs exchange traffic with them (as discussed below). Edomains will often not line up with the ASes of the Internet, and in particular a host's first hop SN may be in a different AS than the host. In what follows, we will sometimes refer to a host residing in an edomain, but what we mean is that its first-hop SN resides in that edomain.

3.2 Operational Details

Service invocation: Services can be invoked in several ways. First, they can be explicitly invoked by hosts (either by the client or server when establishing a connection), with applications indicating their desired service to the host OS via an extended host network API. This desired service is signalled to the SN via metadata in ILP. There is no composition in such explicit invocations; hosts can only invoke a single service because the InterEdge cannot guarantee that any two arbitrary services meaningfully compose. However, naturally composable services can be combined into "bundles" (*e.g.,* an IP-like service and a caching service) that hosts can invoke, and the invocation may have optional settings (signalled in the metadata) that control various aspects of the service (*e.g.,* whether or not to invoke caching). Note that this bundling is one way InterEdge can integrate several services; more generally (as we discuss later in Section 5), the burden of figuring out how to combine two or more services is taken on by the developers of those services, not the customers of those services.

Second, such services can be invoked by the host out of band (via a control protocol between the host and its first-hop SN) and applied to specified portions of its traffic (*e.g.,* a host asking for last-hop QoS to be applied to portions of its incoming traffic, as discussed in Section 6).

Third, services can be imposed by a third party, which typically would be the operator of the network in which a host resides; for example, an enterprise may impose a firewall service or an SD-WAN service on all traffic entering and leaving its network. In this case, the enterprise would have what we call a "pass-through" SN at its boundary that terminates ILP and executes the operator-imposed services, and then forwards to the next-hop SN where the client-invoked InterEdge services would be implemented (and the client's partial trust relationship discussed in Section 4 is with that next-hop SN). As is true today, it is up to the operator to ensure that such operator-imposed services do not overly restrict host behavior.

Inter-edomain connectivity: To deliver packets to receivers in different edomains, we need to establish connectivity between edomains via ILP. In the InterEdge, we require that every edomain

peers directly with all other edomains via an ILP connection. The economics of these peering arrangements are discussed in Section 5. These peering arrangements between edomains do not change the economic arrangements or the mechanisms used for interdomain routing at the IP layer.

The actual mechanics of inter-edomain forwarding are a bit more involved. For every pair of edomains, basic edomain administration will ensure that (i) there is at least one pair of SNs (one in each edomain) directly connected by a long-lived ILP connection and (ii) each SN has a mapping between each edomain and an SN in their edomain that has a direct connection to that edomain. SNs can route inter-edomain traffic through the appropriate SN in their edomain, or, as an optimization, they can establish, on demand, a connection directly to the destination's associated SN in another edomain. We evaluate the overheads of maintaining many peering connections in Appendix C.

Direct connectivity: The typical communication path (at least for host-to-host interactions) involves a packet going from the source to the SN associated with the source, to the SN associated with the destination, and then to the destination. In some cases, however, communications can go directly between hosts without any intervening SNs, akin to how in IP two hosts in the same subnet can communicate directly without passing through a router. More specifically, if the host implementation of the InterEdge service allows direct connection and detects that the source and destination are within the same subnet (or closer to each other than to the associated SNs), the host service implementations on the source and destination can exchange packets directly over ILP.

Name services: Different services can be based on different name and address spaces (*e.g.,* pub/sub is based on topics, multicast is based on groups). However, we assume for point-to-point services that the appropriate name resolution returns not just the service-specific address but also one or more SNs associated with the destination host.

3.3 Important Properties

Flexible, performant, and extensible: These are the properties we promised in the Introduction, and we now justify them here. All services can avail themselves of general compute, so there is complete flexibility in the semantics of the services. The performance comes from carefully separating service functions into fast-path, slow-path, and off-path components. The extensibility is due to the common execution environment and the deployment model. When a new service is standardized, its service module is made available, and after some delay to allow for sufficient testing (say, on the order of several months), it is required that all SNs support this service. At that point, hosts that have applications or operating systems that are aware of this new service can invoke it, and hosts that are unaware can continue without change. The InterEdge is thus fully extensible, and we expect that its set of services will grow over time. We expect these services to be basic primitives (such as pub/sub or attestation) or use-case-specific (such as support for streaming), but not application-specific (*i.e.,* they are not designed to support specific proprietary applications).

Backwards compatible: The InterEdge is backwards-compatible, in that InterEdge-unaware endpoints can continue interacting with each other without change, because nothing in the existing IP infrastructure has changed.

Resilient: Interposition makes resiliency harder to achieve, a problem already raised by currently deployed edge networking services. While a more thorough answer to this question is outside the scope of this paper, we observe that (i) for stateless services, SN failures are like router failures and can be easily recovered from, and (ii) for stateful services, one can use host-driven state reconstruction techniques (as briefly mentioned for pub/sub in Section 6) for correctness and standby-replication for performance.

4 Supporting Interposition

Since interposition is a necessary feature of the InterEdge, we think it is important to revisit how best to support interposition. Today's implementations of interposition have two major drawbacks. First, they are computationally inefficient (see [72] for an exploration of interposition overheads in service meshes and other settings) because they need to terminate a connection (since they must operate on data that might be encrypted) and then forward it on another connection. Second, many of today's implementations, at least those in the hyperscaled private networks, are built on the assumption that the interposing point belongs in the same trust domain as the backend server (so that assuming access to the connection payload does not violate trust assumptions). This limits where interposition can be performed and is therefore antithetical to our goal of providing a common set of widely deployed edge networking services implemented using potentially third-party interposition.

We designed ILP to address these problems. To start with, we assume a different trust model than the one assumed by existing interposition solutions. Our trust model starts with what is assumed by applications (both those running over the Internet and inside datacenters) that do not use interposed services; we assume that application developers and users fully trust the devices and servers that they control, and that they do not trust network switches or links (as evidenced by the wider adoption of TLS, QUIC, and other encrypted transport protocols). We then extend that model so that, in the InterEdge, we assume that applications only *partially trust* SNs. The partial trust in SNs reflects the fact that an SN must necessarily learn more information about a packet than what is carried in current layer 2–4 headers, because (in lieu of secure enclaves, which we discuss in Section 6) they can observe which service processes the packet, and the type of processing applied by the service. However, applications (and users) can control what additional information is revealed by choosing the service they use and the inputs they provide in the ILP header.

In addition to adopting the trust model above, we designed ILP to meet three additional goals:

- **Generality:** It should not restrict what services can be implemented.
- **Efficiency:** It should have minimal impact on packet latency and time to establish a connection, beyond the overheads imposed by the service itself.
- **Cacheability:** It should allow services to use a decision cache at the pipe-terminus to reduce processing overheads.

To fit within our trust model, ILP requires each packet's ILP headers (Figure 2) to be encrypted using a key shared between the

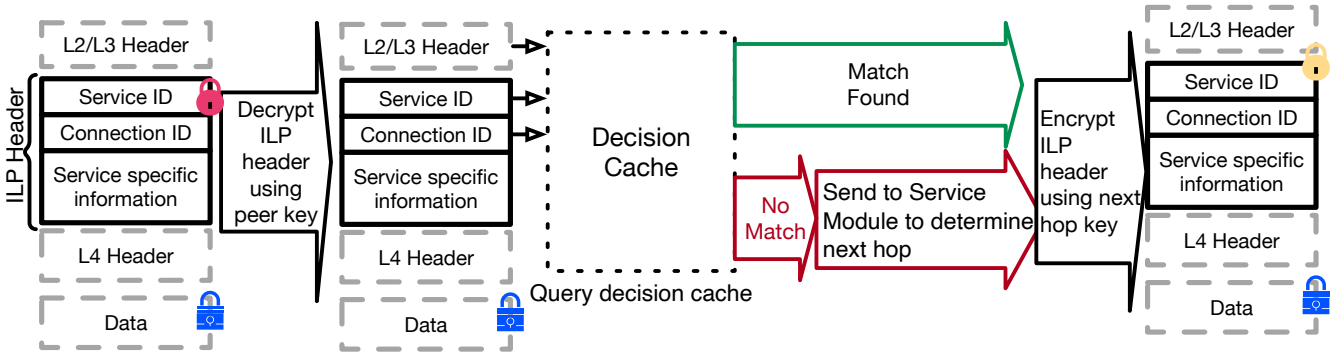

Figure 2: ILP header and processing at a SN

ILP packet's sender and its receiver (either a host and an SN or two SNs), and must be decrypted before it can be processed. As is standard, we assume that this shared key is created when the sender and the receiver first connect with each other: *i.e.,* when a host first associates with an SN or when two SNs establish a pipe between each other. This shared key is only used to encrypt the ILP header, and we assume that application-level data is protected using a different key that is only known to the connection endpoints. Using a single shared key to encrypt ILP headers for multiple connections ensures that ILP adds no additional latency when establishing a connection or when sending individual packets. To reduce cryptographic overheads and avoid imposing ordering or reliability requirements on the network, we plan to use PSP [34], a recently proposed encryption protocol that is designed so it can be easily offloaded to NIC ASICs and can operate on individual packets, even when they arrive out of order.

As we described previously, all packets at an SN are first received by the pipe-terminus, which is responsible for decrypting the ILP header using the shared key associated with the packet's source address (carried in the Layer 3 header). Next, the pipe-terminus uses the packet's L3 header, service ID, and connection ID to query the decision cache. If a match is found, it indicates whether and to whom the SN should forward the packet (the decision can specify multiple forwarding destinations, in which case a copy of the packet is forwarded to each destination). If no match is found, the pipe-terminus forwards the packet's L3 header and decrypted ILP header to the appropriate service (determined using the service identifier in the ILP header). Services can use an InterEdge-provided API and information from the L3 header (*i.e.,* the source address) and ILP header (the connection ID) to add rules to the decision cache, if desired. Either the decision cache or the service provides the pipe-terminus with a (possibly empty) list of forwarding destinations. The pipe-terminus looks up each destination's key, encrypts the ILP header, adds the appropriate network headers, and forwards the packet (Figure 2).

To ensure generality, other than requiring that the initial portion of the ILP header contain a service ID and connection ID, we place no limits on the length (beyond those imposed by the network MTU) or contents of a packet's ILP header. Furthermore, we allow services to require different ILP headers (while keeping service and connection IDs the same) for different packets in a connection. This allows services to encode an arbitrary amount of information in the ILP header (across packets). Thus, our design imposes no limitation on the size or type of inputs used by a service.

The use of separate encryption keys for the ILP header and application data is necessary, given our trust assumptions and requirements. While some prior work [36, 49, 61] has shown how cryptographic techniques (such as homomorphic encryption) that allow algorithms to compute on encrypted data can be used to build interposition software that does not have complete visibility into application content, these approaches come at the cost of generality (since they limit what computation can be performed), cacheability, and efficiency. To the best of our knowledge, no existing approach meets our three requirements while also obeying our trust assumptions. See Appendix B for more details on how ILP can be used by services.

5 Supporting Interconnection and Neutrality

Interconnection and neutrality are core requirements of the InterEdge. As discussed, interconnection requires standards for the services SNs deploy, coordination among the SNs to appropriately respond to service requests, and financial arrangements among IESPs. We have already discussed the standardization of InterEdge services, but here we note that this includes standardizing how those services are configured, so that a customer can seamlessly move between IESPs without having to reconfigure their services. The standards thus not only allow interconnection but also provide portability.

However, the coordination of InterEdge services is more complicated than that of packet delivery (though we are not saying that BGP is simple!) and depends on the nature of the particular service. When an enterprise has arranged for an SD-WAN service, the associated SN for outgoing packets goes through the enterprise's first-hop SN arranged for by the enterprise. When a residential customer has arranged for last-hop QoS (a service described in Section 6), the associated SN for incoming packets goes through an SN arranged for by the resident. However, when an application provider has arranged for a CDN service from some IESP, then the application packets going to some client come from an SN associated with the application provider's IESP. For more complicated services that involve interactions between multiple clients, such as

the pub/sub service we describe in Section 6, there are more complicated interactions between SNs. These interactions are driven by the service's standard implementation and, thus, are typically service-specific and worked out by the designers of the service.

The last aspect of interconnection is the financial arrangement of peering between IESPs. In the InterEdge, we require that in terms of ILP connections, every edomain engages in settlement-free peering with all other edomains; that is, when an SN in one edomain sends packets via ILP to an SN in another edomain, no money changes hands. The settlement-free requirement is natural – neither edomain is offering transport, and each is being paid directly by their respective customers – and necessary to avoid issues of network neutrality (see [38] for further discussion of this topic, in a somewhat different context). These InterEdge arrangements do not change the economic arrangements or the mechanisms used for interdomain routing at the IP layer.

While we have covered the basics of achieving interconnection, three thorny problems remain.

First, since there can be different ways SNs are associated with a host (these associations can be based on the user, the enterprise, the application, and/or the content provider), how are they coordinated? We don't think there is a magic answer, but instead hope that this is something for which a workable set of guidelines will emerge as the ecosystem matures; as a parallel, recall that BGP has oscillatory modes in theory, but a set of reasonable business practices provably avoid them [27]. As a starting point for such guidelines, note that there are services paid for by enterprises (such as last-hop QoS) that apply to all traffic and can be administered by an SN very close to the enterprise, while other services that are more application-specific (such as caching) can be applied by a different SN that need not be located as close. The coordination rules for these two kinds of services would be that the client's request for content would travel to its own first-hop SN (dictated by the enterprise's InterEdge configuration), then to the first-hop SN run by the IESP hired by the application provider. The return path would be the reverse, with the cached content going from the SN paid for by the application provider to the SN paid for by the enterprise and then to the client itself.

Second, how do we ensure neutrality? The most important goal here is to prevent an IESP from discriminating against specific application providers; this would ensure that application-level markets are not distorted by the interests of the IESPs, which is the essence of network neutrality. To combat this, we propose that each IESP be forced to publish their standard rates and make their services available to all on nondiscriminatory terms (subject, of course, to external regulatory concerns). These prices might depend on the volume and location of service, but cannot vary based on the customer. More specifically, there can be no discrimination based on the user's identity aside from the type of service requested and the amount they are paying. However, there are other societal goals that come under the umbrella of neutrality (see [2] for a discussion of neutrality in caching). These additional notions of neutrality will vary on a service-by-service basis, and should be addressed by the body standardizing these services.

Third, there are a variety of ways customers pay IESPs, and how do we support them all? This interaction is simple when the IESP is being paid by the owner of the host, since each owner can choose their IESP. The situation is more complicated when the payment comes from an application or content provider (for, say, CDN or transcoding services) where a single entity wants to arrange for first-hop SNs near *all* potential clients. They have a choice of going with an IESP that has global coverage or going with a set of smaller IESPs that together cover the desired areas. We believe that most customers will not want to stitch together coverage, yet we also do not want the IESP market dominated by an oligopoly of global providers. However, with standard rates being published openly, we believe that a set of "brokers" will arise that can do the stitching on behalf of customers. Such brokers are common in other industries (*e.g.,* travel), and could help allow collections of smaller IESPs compete with the global ones; in particular, carriers could more easily become IESPs (deploying at the edge is easy for them, but attaining global coverage is hard).

6 Supporting Applications

6.1 Context

The intent of this section is to describe how the InterEdge can help support applications through the deployment of edge networking services. The InterEdge architecture described in Section 3 can implement a wide range of edge networking services. Its performance is not its competitive advantage, as we do not expect the InterEdge implementations of common services such as CDNs to match the finely-tuned implementations used by the larger ESPs. The fact that the InterEdge architecture provides a platform on which many different services can run is a step forward,[1] but the presence of a common execution environment and standardized open-source service modules is the more radical departure. It will not change what *could* be deployed at the edge, but it can change what *might* be deployed by lowering the deployment barriers and enabling interconnection. This lowering of deployment barriers is because (i) the common execution environment enables deployment of a single implementation in all SNs, and (ii) the presence of an open-source infrastructure and set of services means that one can become an ESP without a sizable development team.

In this section, we do not list the commonly offered services like caching or ZTNA, but instead, we offer a few examples of services that might not individually generate significant revenue, but together would create a better platform for applications and users. These services fall into five categories: privacy, security, last-hop QoS, multipoint delivery, and specialty services.

6.2 Examples of Possible InterEdge Services

Privacy: The essence of interposition is that intermediate nodes can access some application (*i.e.,* payload) state, which inherently lessens client privacy. To counteract this, we propose the following for privacy-sensitive services: (i) SNs perform their interposed packet processing in secure enclaves (the use of enclaves will be specified when the service is standardized), and (ii) ILP is always encrypted. Thus, unless an attacker has penetrated the secure enclave, the non-enclave portions of SNs are only aware of what other SNs they are communicating with (since they can see the destination address in the ILP header) but know nothing about the content. The

[1] A designer of one of the leading ESPs, who we obviously cannot name, said to us: "We need a platform like this. Our current development practice is too fragmented."

use of enclaves makes it simpler to implement oDNS, private relays, ToR-like mixnet infrastructures, and other privacy-aware services. There are other approaches to privacy in middleboxes (such as mbTLS [48], ZKMB [36], and BlindBox [61]), but these typically require changes to client protocols. Note that InterEdge's privacy guarantees are somewhat different from these three examples, and that InterEdge relies on enclaves whose security models are evolving. In addition, packet processing in secure enclaves incurs some overhead, which we characterize in Appendix C.

Security: InterEdge services, and the applications built thereon, will use the usual suite of cryptographic protocols (suitably modified for InterEdge) to ensure the privacy and integrity of communications. Beyond these traditional measures, we describe one area where InterEdge could substantially improve security: prefix hijacking. Such attacks have long been a concern for network and cloud service providers, and recent prefix hijacking attacks have targeted cloud providers [30, 32], online services [33], and cellular providers [31]. The Resource Public Key Infrastructure (RPKI) has been proposed to eliminate these problems. Unfortunately, prior work [29, 43] has shown that these RPKI-based protections against prefix hijacking are not yet widely effective. In particular, while there has been an increase in the number of ASes that use route origin authorization (ROA) (*i.e.,* they provide cryptographically signed information attesting to path validity), relatively few perform route origin validation (ROV) (*i.e.,* few check the correctness of this data). A recent (2022) survey by Cloudflare [55] found that fewer than 261 million users (about 6.5% of Internet users) are protected by these techniques. While subsequent work [1] says that a larger number of users might be protected (because of validation by transit ASes), there is agreement that prefix attacks are a concern until we get widespread (near universal) deployment of ROA and ROV. This is a problem that InterEdge can address: our design assumes that any pair of InterEdge SNs are connected directly over an encrypted and authenticated tunnel. This trivially prevents hijacking attacks that redirect traffic to malicious hosts, as was the case with recent attacks on Amazon [30, 32] and Twitter [33]. While this approach cannot restrict what geographic regions data transits when traveling between the two endpoints [31], the use of encryption limits what information can be garnered by an adversary.

Last-hop QoS: While the general topic of QoS has resulted in many proposals (*e.g.,* [7, 52]) but little deployment, here we target the more limited, but still useful, notion of QoS where the main source of degraded service comes at the user's own edge (*i.e.,* their Internet access point). InterEdge could provide a last-hop QoS service to receivers where they specify to their first-hop SN (which is presumably on the other side of their congested network access link) the total bandwidth that their access link can handle and a set of weights or priorities (for weighted-fair-queueing and/or priority scheduling) for various traffic streams (identified by source prefixes). This approach would allow a household to give high priority to gaming traffic (which requires low latency), while still preserving enough bandwidth for streaming movies (see [13] for a similar service).

Multipoint delivery: The InterEdge could easily support interconnected versions of three popular multipoint paradigms: anycast packet delivery, multicast packet delivery, and pub/sub message delivery. To give a brief idea of how this might be implemented, we describe how to handle the core design challenge of knowing where the various members of a group are. We assume that IANA or some other organization provides a durable and scalable lookup service that associates each address with the public key of the owner of that address, and contains additional information described below. Receivers join anycast or multicast groups by sending a join message to their first-hop SN, and these messages must have a signature from the owner authorizing them to join. Some groups will be open to all, and the owner can post a signed statement in the lookup service, allowing all receivers to validate their join messages. To achieve better scalability, we change the semantics of anycast and multicast so that before a host can send to a group it must first inform its first-hop SN of its intention to do so; *i.e.,* it must register as a sender to the group. We assume that edomains use SDN-like network management tools with a persistent and scalable store that we refer to as the core (which will be used in anycast, multicast, and pub/sub). Whenever an SN receives a join message for a group for which it does not currently have a member, it sends a notice to the edomain's core, which keeps track of which of the edomain's SNs have members of each group. If the edomain does not currently have a member, the core forwards this message to the IANA lookup service, which keeps track of which edomains have members of each group. The same is done when registering as a sender (first to SN, then perhaps to core, and then perhaps to lookup service), with the extra step that the SN reads from the core the set of other internal SNs that have members (and puts a watch on this list so the core will send updates). When the core registers a sender with the lookup service, it reads from the lookup service the list of edomains with members (and puts a watch on that list so the lookup service will send updates). As a result of these actions, each SN knows, for each group for which it has a sender, all SNs in its edomain that have associated members. Each SN also knows all group memberships of its associated hosts. Each edomain core knows the group memberships of each of its SNs, and for those groups where there is at least one sender in the edomain, the core knows which other edomains have members in that group. The rest of the mechanisms for anycast, multicast, and pub/sub are relatively straightforward, so we do not describe them here, but we have an implementation of pub/sub running on our prototype.

Specialty services: The previous services are all familiar generic primitives, but the InterEdge could also provide services that are more specialized to certain use cases. For instance, message queues such as Kafka [47] are a core component of many distributed applications that process streaming data, but most implementations are typically optimized for environments where these requests are made by processes that run in the same datacenter. Recently, however, Cloudflare Queues [20] has tried to address this change in workloads by proposing a geo-distributed message queuing service running on its edge. The InterEdge could provide such a service in an interconnected manner.

Bulk data delivery is a form of multipoint delivery but focuses on large data transfers rather than single packets or messages. The InterEdge could incorporate an interconnected version of this, and we are currently building such a service for possible use for large experimental datasets in the scientific community.

Similarly, consistency services such as Spanner [21] and other follow-on works [28, 35] have shown that using fine-grained time synchronization for message ordering can reduce coordination overheads, and thus improve performance for databases and enable new use cases. If InterEdge requires that SNs be equipped with GPS receivers, it could offer a high-latency (owing to variations in latency) and low-throughput (due to limits on time precision) but ordered message delivery system. While such a system cannot guarantee atomicity (since we cannot assume bounds on message latencies), the prior works we cited above show that even ordering in the absence of atomicity can reduce coordination overheads for applications.

Lastly, the InterEdge could easily support a generic VPN service that provides a customer with a publicly reachable address, redirects incoming traffic to a customer-specified authentication service, and only allows in traffic that has been duly authenticated.

6.3 Discussion

To reiterate our opening comments in this section, the point of the InterEdge is not that it is technically superior in capabilities to what ESPs have today but that its use of a common open-source platform and its focus on interconnection (which allows services to arise that connect clients using different IESPs), makes it easier to deploy services that might not, on their own, generate enough revenue to warrant deployment. But taken together, these less lucrative services would make the Internet a far better platform for applications. For instance, security would be improved if prefix hijacking were eliminated; users would be happier if they could resolve congestion on their last mile by choosing the relative priorities of incoming connections, and if their privacy was always protected by mechanisms such as private relay, oDNS, and (when desired) mixnets; and application development would be eased by the presence of global-scale multipoint delivery and other primitives like message queues. Of course, we have only listed a few of the possible services that could be integrated into InterEdge, and we expect many more possibilities to emerge as new use cases arise.

We have developed an InterEdge prototype and have tested InterEdge-enabled hosts and the following services (some of which we did not have space to mention here) in our lab, on CloudLab [24], and on Fabric [25]: mixnets, pub/sub, oDNS, cluster interconnection, DDoS protection, mobility lookup service, and support for attestation. In building our initial prototype, we focused on flexibility, not performance. Specifically, we designed it so that we can rapidly add services, prototype new features, and deploy them in virtualized environments. We used IPC to send and receive data from services which obviously adds overhead, but this approach makes it trivial to prototype services. There are well-known solutions to address these and other performance bottlenecks in our prototype, but we will only begin focusing on performance once we have explored more of the service design space.

7 Ecosystem Considerations

The current edge networking services ecosystem has two notable characteristics: a predominant focus on a few highly lucrative services (with CDN, SD-WAN, and security features probably leading the way), and two categories of networking service providers: the hyperscaled cloud and application providers who apply edge networking services to their own traffic (and potentially to those of their cloud customers), and a broader set of commercial ESPs. At the structural level, the edge networking services ecosystem has clearly been shaped by seeking revenues, rather than starting with a clean architecture. The reverse is true of the Internet, where the architecture preceded commercialization. More generally, the lack of an architecture occurs in many technology ecosystems, but it is rarer in communication ecosystems where the need for interconnection (between providers), portability (devices moving between providers), and uniformity (so devices can talk to each other) has typically driven the adoption of architectural standards (*e.g.,* cellular, telephony, wifi, bluetooth, etc.).

The question, then, is how *might* we "retrofit" an architecture onto an already existing ecosystem? The operative word is "might" because such a massive change in a large ecosystem is, at best, a low-probability event. We approach this as two separate issues: (i) once the InterEdge has some traction how might it spread more widely, and (ii) how might the InterEdge get its initial traction.

Turning to the first issue, we ask the following question: if a production-quality InterEdge implementation achieved some level of adoption, how might the ecosystem evolve in response to its presence? In the short term, we envision that the private networks of the hyperscalers would remain unchanged, as they would be unwilling to risk their core business by adopting an emerging infrastructure. In contrast, we think some of the larger ESPs might adopt the InterEdge (while still offering some of their own proprietary services) because their size (in terms of coverage) offers such an advantage that using a standardized service might help rather than hurt (*i.e.,* customers would have no worry about lock-in, but would still seek out the largest providers). The probability of this depends greatly on the extent of early adoption on which this question is predicated; the larger ESPs would likely only adopt if the InterEdge had sufficient traction so that (i) adopting it did not scare off current customers and (ii) there were enough novel services (requiring changes to client protocols) so that ESPs were willing to abandon their narrower proprietary strategy in favor of offering a wider variety of functionality.

This positive outcome will require four significant ecosystem advances. First is the development of a new set of services that are not compatible with current client protocols and that complete the path from "interesting open-source offering" to "InterEdge-adopted standard" to "widely used InterEdge service." These won't necessarily be highly lucrative services, but services that prove useful in various use cases. This will help establish the benefit of not remaining tied to current client protocols. Second, the InterEdge community (developers, providers, and customers) provides some overall architectural oversight in terms of how to address the thorny problems we raise in Section 5. Third, a new generation of IESPs emerges; because the InterEdge software infrastructure is open-source, these new providers need not have large development teams, but only need to have edge deployments where it would be easy to add InterEdge SNs (and these edge deployments need not be global in scope, because of interconnection). Examples of these new IESPs could include ISPs, cellular carriers, IXPs, and others. These new entrants to the ESP business would provide the impetus for larger ESPs to adopt, since through interconnection these new

entrants – who collectively would have far more sites than even the largest existing ESPs – could create a credible threat to the larger ESPs. Fourth, the combination of these new IESPs and the ability to interconnect means that coverage greatly improves.

If all this comes to pass, the InterEdge could become so widespread that it essentially defines a new and extensible service model of the Internet. At that point, the hyperscalers might abandon their proprietary private networks in favor of the InterEdge, given that the InterEdge would provide more features and more coverage, at potentially less cost and little fear of lock-in. These private networks are not profit centers for the hyperscalers, they are merely a means to an end: customers who stay online longer. If the InterEdge makes it easier and cheaper (by providing an open-source infrastructure and potentially third-party providers) to achieve that goal, it might prove to be the more attractive option.

Given this possible trajectory from initial adoption towards wider InterEdge deployment, we turn to the second issue of initial adoption and ask the following question: who might be an early adopter of the InterEdge? One possibility is the large ESPs. If you view them as dominant in the ESP ecosystem, the incentives are weak without the InterEdge gaining prior traction. However, if you view the hyperscaled clouds and application providers as the true apex predators of the broader ecosystem, and the ESPs as merely nibbling at the corners of the market (which, given that the relative market caps differ by more than an order of magnitude, is not an unreasonable perspective), then the incentives for the large ESPs to adopt become stronger. That is, the large ESPs might adopt InterEdge to avoid being absorbed by the hyperscalers.

However, there are indications that the large ESPs do not see the cloud and application providers as competitors. There is good reason for this view, because the hyperscalers and the ESPs have very different business models (*e.g.,* different margins), so it is unlikely that the hyperscalers would go after the ESP market. In that case, our search for early adopters should focus on the ESP ecosystem. Since the large ESPs dominate in this market, there is little reason for them to precipitously jump to the InterEdge. However, a key aspect of the InterEdge is, as we mentioned above, that it would enable collections of smaller ESPs to compete more effectively with the larger ones. Thus, the most likely first adopters would be the smaller ESPs and companies that are not in the ESP business but have physical infrastructures at the edge that could easily host SNs.

But that leaves open the question of how these groups could take the first step towards creating the InterEdge. The ARPANET provides an instructive example of one such possible first step. The ARPANET provided a pre-commercial platform on which the ideas that led to the Internet could be tested, and then demonstrated, and then used by applications. Recall that the ARPANET did not just prove that what we now call the Internet architecture could deliver packets, but also that such a standardized packet-delivery platform could be used by a wide range of applications.

Similarly, what might be needed to encourage the smaller ESPs and the not-yet-ESPs to adopt the InterEdge is an ongoing pre-commercial deployment that proves not only that the InterEdge can support services, but also that some of the more novel services (that current ESPs are not supporting) could gain traction in the marketplace. This is the path we are currently pursuing, by seeking to operate our prototype as an open platform on which the research community can explore new edge services and new applications that exploit those and other edge services.

Recall that the purpose of this section was to describe a plausible path to adoption, and point out some of the necessary precursors. Admittedly, at this point success seems far-fetched, but as the saying goes "Pessimists are usually right and optimists are usually wrong but all the great changes have been accomplished by optimists" [26]. On the other hand, as Macbeth said, this could be a "tale told by an idiot, full of sound and fury, signifying nothing." Neither the authors nor the readers of this paper can predict the future; all we can do in this section is describe some of the incentives that may affect the outcome.

8 Conclusion

The previous section speculated on how the InterEdge *might* change the ecosystem, but such predictions – both our optimistic ones and our critics' pessimistic ones – are hardly reliable, and our optimism is not the contribution of this paper. Instead, the contribution of this paper is to describe, for the first time, what an architecture for edge networking services might look like. These edge networking services are here to stay, and we think the role of our research community is to provide insight into how we might guide the ecosystem towards a future that best serves society's need by providing a global, extensible, and neutral platform for application deployment. This paper is our initial attempt to provide some of that guidance, and we hope that its publication will spur a broader discussion of this topic.

What our design shows is that an interconnected and neutral architecture for edge networking services could be easily built, and plausibly deployable through current ecosystem incentives. Moreover, the main benefit of this architecture is not the development of radical new services that *cannot be built* today, but rather the development and deployment of a range of widely useful, but perhaps not sufficiently backwards-compatible and/or lucrative, services that *would not be deployed* today.

Ethics Statement. While this paper addresses ethical issues relating to what designs might be best for society, the methodology does not raise any ethical concerns.

Acknowledgments

We thank our shepherd Katerina Argyraki and the SIGCOMM reviewers for their comments. We also thank Marwan Fayed, David D. Clark, Mark Nottingham, Michael Schapira, and members of the Berkeley NetSys lab for their comments. This work was funded in part by gifts from Intel, Broadcom (through VMware Research) and Protocol Labs; by grants from NSF (CNS-2242502, CNS-2242503, CNS-2148275, Eager-2137220, OAC-2201489, and CNS-2213387); and by the Department of Energy's Office of Advanced Scientific Computing Research (ASCR) under Contract No. DE-AC02-05CH1123.

References

[1] 2023. Exploring the Latest RPKI ROV Adoption Numbers. https://www.kentik.com/blog/exploring-the-latest-rpki-rov-adoption-numbers/.

[2] Muhammad Abdullah, Pavlos Nikolopoulos, and Katerina Argyraki. 2023. Caching and Neutrality. In *HotNets*. Association for Computing Machinery.

[3] Tom Anderson, Ken Birman, Robert Broberg, Matthew Caesar, Douglas Comer, Chase Cotton, Michael J. Freedman, Andreas Haeberlen, Zachary G. Ives, Arvind Krishnamurthy, William Lehr, Boon Thau Loo, David Mazières, Antonio Nicolosi, Jonathan M. Smith, Ion Stoica, Robbert van Renesse, Michael Walfish, Hakim Weatherspoon, and Christopher S. Yoo. 2014. A Brief Overview of the NEBULA Future Internet Architecture. *SIGCOMM CCR* (2014).

[4] AOL. 2024. https://www.aol.com/.

[5] Apple. 2024. About iCloud Private Relay. https://support.apple.com/en-us/HT212614.

[6] Todd Arnold, Jia He, Weifan Jiang, Matt Calder, Italo Cunha, Vasileios Giotsas, and Ethan Katz-Bassett. 2020. Cloud Provider Connectivity in the Flat Internet. In *IMC*.

[7] Fred Baker, David L. Black, Kathleen Nichols, and Steven L. Blake. 1998. *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*. RFC 2474.

[8] Hari Balakrishnan, Sujata Banerjee, Israel Cidon, David E. Culler, Deborah Estrin, Ethan Katz-Bassett, Arvind Krishnamurthy, James Murphy McCauley, Nick McKeown, Aurojit Panda, Sylvia Ratnasamy, Jennifer Rexford, Michael Schapira, Scott Shenker, Ion Stoica, David L. Tennenhouse, Amin Vahdat, and Ellen W. Zegura. 2021. Revitalizing the public internet by making it extensible. *Comput. Commun. Rev.* 51, 2 (2021), 18–24. https://doi.org/10.1145/3464994.3464998

[9] Gilles Bertrand, Stephan Emile, Trevor Burbridge, Philip Eardley, Kevin J. Ma, and Grant Watson. 2012. Use Cases for Content Delivery Network Interconnection. RFC 6770. https://doi.org/10.17487/RFC6770

[10] Marjory Blumenthal, Ramesh Govindan, Ethan Katz-Bassett, Arvind Krishnamurthy, James McCauley, Nick Merrill, Tejas Narechania, Aurojit Panda, and Scott Shenker. 2024. Can We Save the Public Internet? *SIGCOMM Comput. Commun. Rev.* 53, 3 (feb 2024), 18–22. https://doi.org/10.1145/3649171.3649175

[11] Marjory S. Blumenthal and David D. Clark. 2001. Rethinking the Design of the Internet: The End-to-End Arguments vs. the Brave New World. *ACM Trans. Internet Technol.* (2001).

[12] Open Caching. 2021. SVTA2007: Open Cache Request Routing Functional Specification. https://www.svta.org/product/svta2007/.

[13] Frank Cangialosi, Akshay Narayan, Prateesh Goyal, Radhika Mittal, Mohammad Alizadeh, and Hari Balakrishnan. 2021. Site-to-Site Internet Traffic Control *(EuroSys)*.

[14] Karen Casella, Travis Nelson, and Sunny Singh. 2021. Edge Authentication and Token-Agnostic Identity Propagation. https://netflixtechblog.com/edge-authentication-and-token-agnostic-identity-propagation-514e47e0b602.

[15] CDN Planet. 2024. Content Delivery Networks by country. https://www.cdnplanet.com/cdns-by-country/.

[16] Sarah Chasins, Alvin Cheung, Natacha Crooks, Ali Ghodsi, Ken Goldberg, Joseph E Gonzalez, Joseph M Hellerstein, Michael I Jordan, Anthony D Joseph, Michael W Mahoney, et al. 2022. The sky above the clouds. *arXiv preprint arXiv:2205.07147* (2022).

[17] T.M. Chen and A.W. Jackson. 1998. Commentaries on "Active networking and end-to-end arguments". *IEEE Network* 12, 3 (1998), 66–71. https://doi.org/10.1109/65.690972

[18] David D. Clark. 2018. *Designing an Internet* (1st ed.). The MIT Press.

[19] David D. Clark, Scott Shenker, and Lixia Zhang. 1992. Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism. In *Proceedings of the Conference on Communications Architecture & Protocols, SIGCOMM 1992, Baltimore, Maryland, USA, August 17-20, 1992*, Deepinder P. Sidhu and David Oran (Eds.). ACM.

[20] Cloudflare. 2023. Cloudflare Queues. https://developers.cloudflare.com/queues/.

[21] James C Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, Jeffrey John Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, et al. 2013. Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems (TOCS)* 31, 3 (2013), 1–22.

[22] Jon Crowcroft, Steven Hand, Richard Mortier, Timothy Roscoe, and Andrew Warfield. 2003. Plutarch: An Argument for Network Pluralism. *SIGCOMM Comput. Commun. Rev.* (2003).

[23] Jason A Donenfeld. 2017. Wireguard: next generation kernel network tunnel.. In *NDSS*.

[24] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The Design and Operation of CloudLab. In *ATC*. 1–14. https://www.flux.utah.edu/paper/duplyakin-atc19

[25] fabric [n. d.]. FABRIC Testbed. https://fabric-testbed.net.

[26] T.L. Friedman. 2009. *Hot, Flat, and Crowded 2.0: Why We Need a Green Revolution–and How It Can Renew America*. Picador. https://books.google.com/books?id=BpkALHFTnhUC

[27] Lixin Gao and Jennifer Rexford. 2001. Stable internet routing without global coordination. *IEEE/ACM Trans. Netw.* 9, 6 (2001), 681–692. https://doi.org/10.1109/90.974523

[28] Ahmad Ghalayini, Jinkun Geng, Vighnesh Sachidananda, Vinay Sriram, Yilong Geng, Balaji Prabhakar, Mendel Rosenblum, and Anirudh Sivaraman. 2021. CloudEx: a fair-access financial exchange in the cloud. In *HotOS*. 96–103.

[29] Yossi Gilad, Avichai Cohen, Amir Herzberg, Michael Schapira, and Haya Shulman. 2017. Are We There Yet? On RPKI's Deployment and Security. In *NDSS*.

[30] Dan Goodin. 2018. Suspicious event hijacks Amazon traffic for 2 hours, steals cryptocurrency. https://arstechnica.com/information-technology/2018/04/suspicious-event-hijacks-amazon-traffic-for-2-hours-steals-cryptocurrency/.

[31] Dan Goodin. 2019. BGP event sends European mobile traffic through China Telecom for 2 hours. https://arstechnica.com/information-technology/2019/06/bgp-mishap-sends-european-mobile-traffic-through-china-telecom-for-2-hours/.

[32] Dan Goodin. 2022. How 3 hours of inaction from Amazon cost cryptocurrency holders $235,000. https://arstechnica.com/information-technology/2022/09/how-3-hours-of-inaction-from-amazon-cost-cryptocurrency-holders-235000/.

[33] Dan Goodin. 2022. Some Twitter traffic briefly funneled through Russian ISP, thanks to BGP mishap. https://arstechnica.com/information-technology/2022/03/absence-of-malice-russian-isps-hijacking-of-twitter-ips-appears-to-be-a-goof/.

[34] Google. 2022. PSP Architecture Specification. https://raw.githubusercontent.com/google/psp/main/doc/PSP_Arch_Spec.pdf.

[35] Prateesh Goyal, Ilias Marinos, Eashan Gupta, Chaitanya Bandi, Alan Ross, and Ranveer Chandra. 2022. Rethinking cloud-hosted financial exchanges for response time fairness. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*. 108–114.

[36] Paul Grubbs, Arasu Arun, Ye Zhang, Joseph Bonneau, and Michael Walfish. 2022. Zero-Knowledge Middleboxes. In *31st USENIX Security Symposium (USENIX Security 22)*. 4255–4272.

[37] Dongsu Han, Ashok Anand, Fahad Dogar, Boyan Li, Hyeontaek Lim, Michel Machado, Arvind Mukundan, Wenfei Wu, Aditya Akella, David G. Andersen, John W. Byers, Srinivasan Seshan, and Peter Steenkiste. 2012. XIA: Efficient Support for Evolvable Internetworking. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI)*.

[38] Yotam Harchol, Dirk Bergmann, Nick Feamster, Eric Friedman, Arvind Krishnamurthy, Aurojit Panda, Sylvia Ratnasamy, Michael Schapira, and Scott Shenker. 2020. *A Public Option for the Core*. In *SIGCOMM*.

[39] Tatiana Iskandar, Lee Semien, and Daniel Vinegrad. 2010. The End-to-End Principle. https://cs.stanford.edu/people/eroberts/cs201/projects/2010-11/NetNeutrality/Articles/Proponents.html.

[40] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. 2007. A data-oriented (and beyond) network architecture. In *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (Kyoto, Japan) *(SIGCOMM '07)*. Association for Computing Machinery, New York, NY, USA, 181–192. https://doi.org/10.1145/1282380.1282402

[41] Craig Labovitz. 2019. Internet Traffic 2009-2019. https://pc.nanog.org/static/published/meetings/NANOG76/1972/20190610_Labovitz_Internet_Traffic_2009-2019_v1.pdf. Presentation at NANOG 76.

[42] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. 2017. The quic transport protocol: Design and internet-scale deployment. In *Proceedings of the conference of the ACM special interest group on data communication*. 183–196.

[43] Robert Lychev, Sharon Goldberg, and Michael Schapira. 2013. BGP security in partial deployment: Is the juice worth the squeeze?. In *SIGCOMM*.

[44] Julien Mailland and Kevin Driscoll. 2017. Minitel: The online world France built before the web. *IEEE Spectrum* (2017).

[45] James McCauley, Yotam Harchol, Aurojit Panda, Barath Raghavan, and Scott Shenker. 2019. Enabling a permanent revolution in internet architecture. In *Proceedings of the ACM Special Interest Group on Data Communication* (Beijing, China) *(SIGCOMM '19)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3341302.3342075

[46] Tatsuji Moriyoshi, Fumiyo Takano, and Yuichi Nakamura. 2011. GPU acceleration of H. 264/MPEG-4 AVC software video encoder. *APSIPA* (2011).

[47] Neha Narkhede, Gwen Shapira, and Todd Palino. 2017. *Kafka: The Definitive Guide Real-Time Data and Stream Processing at Scale* (1st ed.). O'Reilly Media, Inc.

[48] David Naylor, Richard Li, Christos Gkantsidis, Thomas Karagiannis, and Peter Steenkiste. 2017. And then there were more: Secure communication for more than two parties. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*. 88–100.

[49] David Naylor, Kyle Schomp, Matteo Varvello, Ilias Leontiadis, Jeremy Blackburn, Diego R López, Konstantina Papagiannaki, Pablo Rodriguez Rodriguez, and Peter

Steenkiste. 2015. Multi-context TLS (mcTLS) enabling secure in-network functionality in TLS. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 199–212.

[50] Abhay Kumar Parekh. 1992. *A generalized processor sharing approach to flow control in integrated services networks.* Ph. D. Dissertation. Massachusetts Institute of Technology.

[51] Rishabh Poddar, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. 2018. {SafeBricks}: Shielding Network Functions in the Cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 201–216.

[52] S. Shenker R. Braden, D. Clark. 1994. *Integrated Services in the Internet Architecture: an Overview.* RFC 1633.

[53] r2 2023. Cloudflare R2. https://developers.cloudflare.com/r2/.

[54] Sylvia Ratnasamy, Scott Shenker, and Steven McCanne. 2005. Towards an Evolvable Internet Architecture. In *Proc. of SIGCOMM*.

[55] Carlos Rodrigues and Vasilis Giotsas. 2022. Helping build a safer Internet by measuring BGP RPKI Route Origin Validation. https://blog.cloudflare.com/rpki-updates-data/.

[56] Jeffery Rott. 2012. Intel Advanced Encryption Standard Instructions (AES-NI). Intel Developer Zone https://goo.gl/0VvQ8G, retrieved 07/17/2015.

[57] Jerome H Saltzer, David P Reed, and David D Clark. 1984. End-to-end arguments in system design. *ACM Transactions on Computer Systems (TOCS)* 2, 4 (1984), 277–288.

[58] Paul Schmitt, Anne Edmundson, Allison Mankin, and Nick Feamster. 2019. Oblivious DNS: practical privacy for DNS queries: published in PoPETS 2019. In *Proceedings of the Applied Networking Research Workshop* (Montreal, Quebec, Canada) *(ANRW '19)*. Association for Computing Machinery, New York, NY, USA, 17–19. https://doi.org/10.1145/3340301.3341128

[59] Ivan Seskar, Kiran Nagaraja, Sam Nelson, and Dipankar Raychaudhuri. 2011. MobilityFirst Future Internet Architecture Project. In *AINTEC*.

[60] Scott Shenker. 2022. Creating an Extensible Internet. https://blog.apnic.net/2022/04/14/creating-an-extensible-internet/

[61] Justine Sherry, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. 2015. Blindbox: Deep packet inspection over encrypted traffic. In *Proceedings of the 2015 ACM conference on special interest group on data communication*. 213–226.

[62] Amin Vahdat and Soheil Hassas Yeganeh. 2022. Announcing PSP's cryptographic hardware offload at scale is now open source. https://cloud.google.com/blog/products/identity-security/announcing-psp-security-protocol-is-now-open-source.

[63] Kevin Vermeulen, Loqman Salamatian, Sang Hoon Kim, Matt Calder, and Ethan Katz-Bassett. 2023. The Central Problem with Distributed Content: Common CDN Deployments Centralize Traffic In A Risky Way. In *HotNets*.

[64] Kevin Vermeulen, Loqman Salamatian, Sang Hoon Kim, Matt Calder, and Ethan Katz-Bassett. 2023. The Central Problem with Distributed Content: Common CDN Deployments Centralize Traffic In A Risky Way. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks* (, Cambridge, MA, USA,) *(HotNets '23)*. Association for Computing Machinery, New York, NY, USA, 70–78. https://doi.org/10.1145/3626111.3628213

[65] Gustav Ernberg von Heijne, Jonathon Imperiosi, Rob Hazan, and Beng Eu. 2019. Fast Ads Matter. https://web.dev/articles/fast-ads-matter.

[66] Tao Wang, Xiangrui Yang, Gianni Antichi, Anirudh Sivaraman, and Aurojit Panda. 2022. Isolation mechanisms for {High-Speed}{Packet-Processing} pipelines. In *NSDI*.

[67] Tim Wu. [n. d.]. Network Neutrality FAQ. http://www.timwu.org/network_neutrality.html.

[68] Zadara. [n. d.]. Federated Edge. https://www.zadara.com/federated-edge/.

[69] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, kc claffy, Patrick Crowley, Christos Papadopoulos, Lan Wang, and Beichuan Zhang. 2014. Named data networking. *SIGCOMM Comput. Commun. Rev.* 44, 3 (jul 2014), 66–73. https://doi.org/10.1145/2656877.2656887

[70] Xin Zhang, Hsu-Chun Hsiao, Geoffrey Hasker, Haowen Chan, Adrian Perrig, and David G. Andersen. 2011. SCION: Scalability, Control, and Isolation on Next-Generation Networks. In *IEEE S&P*.

[71] Zhipeng Zhao. 2021. *Pigasus: Efficient Handling of Input-Dependent Streaming on FPGAs.* Ph. D. Dissertation. Carnegie Mellon University.

[72] Xiangfeng Zhu, Guozhen She, Bowen Xue, Yu Zhang, Yongsu Zhang, Xuan Kelvin Zou, Xiongchun Duan, Peng He, Arvind Krishnamurthy, Matthew Lentz, et al. 2022. Dissecting Service Mesh Overheads. *arXiv preprint arXiv:2207.00592* (2022).

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

A Private Hyperscaler Networks and ESPs

We now describe some features of the private networks run by the hyperscaled cloud and application providers, and then describe some features of ESPs. We are not privy to the technical details (size, speeds, workloads, etc.) of the private networks run by hyperscalers, but from what we can gather these networks share some basic common aspects that we describe here. Of course, there are some specialized aspects, such as gaming support, that are specific to certain hyperscalers and we do not mention them here.

These private networks have roughly four distinct functional components, described below. We want to clearly distinguish those components (even though there are significant overlaps in the infrastructure used to support them), to clarify that only the last two in our list support the kind of edge networking services we refer to. The first two are primarily for internal traffic, which is not our concern in this paper.

- Enterprise and datacenter networks: These are purely internal networks.
- Backbone networks: These are high-speed interconnections between their datacenters. They often have specialized mechanisms to maximize utilization and handle failures gracefully. These are primarily for internal traffic.
- "Off-net" edge networking services [64]: These are installations at PoPs often within carrier networks (*i.e.,* with addresses within the carrier's address space).
- "On-net" edge networking services: These are edge networking services placed in PoPs near clients that are connected to the rest of the hyperscaler's network via private dedicated bandwidth, and typically have addresses within the hyperscaler's address space.

Typically ESPs have some combination of On-net and Off-net installations, but do not necessarily have the set of hyperscaled datacenters connected by a high-speed backbone.

B Using ILP in Services

We provided an overview of ILP (§4) earlier in the paper. Here we provide some details about how InterEdge services can use the decision cache, ILP, and offloading capabilities at the pipe-terminus. This offloading to a pipe-terminus is different from offloading capabilities on individual servers (used to accelerate cryptography, video and audio re-encoding, regular expression matching, etc.), in which services access various accelerators by using InterEdge-provided library functions. We start by describing the InterEdge decision cache, and our envisioned pipe-terminus implementation, before describing how services incorporate with this design.

B.1 The Decision Cache and pipe-terminus

We allow implementations of the decision cache to arbitrarily evict entries, even when the connections they are associated with are active. Providing flexibility in when decision cache entries can be evicted is necessary to ensure that the SN remains correct even as the number of connections grow beyond the cache's capacity, and also provides flexibility in how and where the decision cache is implemented. However, this decision means that a service module needs to be able to make forwarding decisions not just for the first few packets in a connection, but for any arbitrary packet in the connection.

Our design expects that decision cache is implemented in the pipe-terminus. The pipe-terminus itself can be implemented in software running on a server (indeed, this is what we do at present, but we expect that in the long-term most SNs will use switching ASICs to implement their pipe-terminus). This is feasible: existing hardware [62] can already encrypt and decrypt ILP headers, and implementing the decision cache merely requires using exact-match tables that are a part of every switch ASIC. While we do not require that switches implementing a pipe-terminus use programmable ASICs, our design allows services to offload functionality to the pipe-terminus if a programmable ASIC with an appropriate isolation mechanism (*e.g.,* using Menshen [66]) is used. We do not discuss this further, beyond noting that the precise decision of what services can be offloaded and how they are expressed depends on the ASIC architecture used.

B.2 Using the Decision Cache

The main challenge that a service module faces when using the decision cache is that it must be able to recompute decisions for any packet in the connection. This is easiest when a service requires that all packets in a connection carry the same information in their ILP header, since in this case the service can reuse the same logic it used when the connection was first established to route any packet. For such services, using the decision cache merely improves performance, and requires no further consideration.

However, in general not all services can use the same ILP headers for all packets in a connection. Some services (*e.g.,* ZTNA security services that check software version information when establishing a connection) require a substantial amount of information to make forwarding decisions. In the best case this information can be a significant portion of the MTU (thus reducing goodput), and in the worse case it might not even fit in a single packet. Therefore, we neither require nor expect all services to use the same ILP header for packets sent while establishing a connection, and packets sent once the connection has been established. Instead, we require that such services maintain an internal cache of their forwarding decisions, thus allowing services to use domain-specific information to implement connection scaling, and determine the duration for which connection information has to be maintained. We allow these service to use the decision cache as above, and to allow them visibility into whether or not a connection is still active. We also provide an API that services can use to determine whether or not a decision cache entry has been recently used. Implementing this API merely requires retrieving the hit-count for an entry, and is supported by most switching ASICs and software routers today.

C Benchmarks

There are two design choices we made whose impact on performance deserves exploration: the use of secure enclaves and direct peering between all domains.

Secure enclaves: One of the novel aspects of InterEdge is its reliance on packet processing in secure enclaves for ensuring privacy.

Microbenchmark	Enclave?	Throughput (PPS)	Latency (us)
No-service	No	377420.1	12.4
No-service	Yes	372882.9	13.1
Null-service	No	120018.5	33.0
Null-service	Yes	110627.1	35.5

Table 1: No-service and null-service performance comparison with and without AMD SEV enclaves in an AMD EPYC 7B12 CPU.

Enclaves are not commonly used for network processing (though see [51]), so we measured their impact on forwarding performance. To assess the resulting overheads on the basic datapath we used a null-service and a no-service microbenchmark. In the null-service case, the packet arrives on an ingress pipe to the pipe-terminus, then is sent to a service module (via IPC) which immediately returns the packet to the pipe-terminus, which then sends it to an egress pipe. The no-service case is where the packet is merely received by the pipe-terminus and then forwarded out the egress pipe, with no invocation of a service via IPC (*e.g.,* as if we implemented service communication through shared memory rings).

The third row of Table 1 shows the packets per second and the latency of an SN using a null-service. Our current prototype is able to handle 120k packets per second using 2 cores (one for the pipe-terminus, and one for the service) and 64 outstanding packets. The unloaded median latency is 33us per packet. The first row in Table 1 shows the performance when we do not have to communicate with a service through IPC. In this case, we can see a single core could handle 377k packets per second with a median latency of 12.4us per packet.

To see what the impact of the enclave is, we repeat the experiments by running them inside a secure enclave. As the second and fourth row of Table 1 show, using an enclave reduces throughput by up to 9% and increases latency by up to 8%. This suggests that it is feasible to use secure enclaves in InterEdge. Note that since enclaves typically have little computational overhead, but do have I/O overhead, these overhead numbers on a no-service and a null-service are likely to be the worst-case overheads.

Direct peering: The direct interdomain peering described in Section 3 requires that some SNs might need to maintain a large number of tunnels to peers in other domains, with encryption as described in 4. To evaluate the feasibility of this scale, we benchmark Wireguard [23], a widely used VPN tunnel. A commodity (16-core) server could easily maintain 98,000 simultaneous tunnels, each doing symmetric key rotation every three minutes. In terms of compute, this consumed less than half a core, and in terms of bandwidth it consumed roughly 3.4 Mbps. Thus, we do not expect the number of directly peered domains to constrain interdomain traffic; instead, the likely bottleneck is the total traffic being handled by any SN, which can be load-balanced by proactive domain management.