Adaptive Addresses for Next Generation IP Protocol in Hierarchical Networks

Haoyu Song, Zhaobo Zhang, Yingzhen Qu, James Guichard
Futurewei Technologies
Santa Clara, USA

Abstract*—With the expected explosion in devices connected to the Internet brought about by Internet of Things (IoT) and 5G, IPv4 address exhaustion is unavoidable and the move to IPv6 is considered the cure. However, the transition is not without its challenges and some looming issues deserve investigation. First, while the IPv6 address space satisfies the scale requirement of Internet-addressable devices, the larger packet header caused by 128-bit addresses has a negative impact on communication efficiency for many envisioned applications. Second, IPv6 is a fixed-length address scheme, which inherently lacks extensibility and will require another overhaul if the need for larger or independent address spaces emerge. Third, the potentially larger forwarding tables based on longer and nested IPv6 prefixes will challenge a router's capacity and performance much more than the case of IPv4. To address these issues, we propose to use adaptive IP addresses under a strict hierarchical network structure. The addressing scheme can be realized in a newer generation of IP protocol (*i.e.*, IPvn). It minimizes the communication overhead incurred by IP addresses, enables arbitrary address space extension, simplifies both the network data-plane and control-plane, and supports better network security. More importantly, it allows incremental deployment from the edge of the network and gradual penetration into the core. A clear boundary between IPvn domain and the existing IPv4/IPv6 networks enables transparent cross-domain communication through a simple header translation. The clear evolution path makes pre-standard deployment possible, allowing shorter addresses to be used where needed so that their benefits can be fully enjoyed instantly. In this paper we evaluate the benefit of the adaptive address, design both control plane and data plane to support it, and prototype the routers within and on the edge of an IPvn domain. We open source the project to encourage further investigation and development.

I. Introduction

Internet of Things (IoT) and 5G will introduce to the Internet a huge number of addressable entities, such as sensors, machines, vehicles, and robots. According to a research report by Statista [1], the total connected IoT devices is projected to reach 75.44 billion worldwide by 2025. The majority of these devices will use wireless connections.

As of November 2019, the unallocated IPv4 address blocks are depleted [2]. Apart from the interim remedies of Network Address Translation (NAT) and address recycling, an accelerated transition to IPv6 is expected. While the 128-bit address space of IPv6 was considered large enough and future-proof, the current practice of IPv6 application has several issues. First, the long IP addresses inflate the packet header size. 80% of a basic 40-byte IPv6 header is consumed by addresses. Second, IPv6 addresses are typically allocated with large blocks (*e.g.*, a prefix length of /48 or /56 is recommended [3], so one organization could receive 2^{80} addresses which is 2.8×10^{14} times greater than the entire IPv4 address space). The extravagance may raise another address exhaustion concern eventually. Third, there is a trend toward extending the meaning of the IPv6 address beyond connectivity (*e.g.*, SRv6 network programming [4]) leading to further address usage.

A. Address Overhead

In IoT networks, machine-to-machine or thing-to-thing communication through wireless connections is dominant, which presents several distinct characteristics. (1) The communication tends to be chatty, which include frequent short message exchanges (*e.g.*, industry robots and networked vehicles). (2) The communication is often energy sensitive (*e.g.*, battery-powered sensors). (3) The communication often requires low latency (*e.g.*, industry control). (4) The precious wireless channels demand high bandwidth utilization (*e.g.*, ZigBee, Bluetooth, Wi-Fi, and 5G). These characteristics render a large header overhead unfavorable and even prohibitive, because it wastes bandwidth, storage, and power, and increases the processing time.

The address overhead also takes its toll on Data Center Networks (DCN), especially when large scale containers are deployed and their prevailing communications are comprised of short messages (*e.g.*, key-value pairs) and conducted through virtual switches. The performance hit is hard when packets tunnel through host machines, according to a recent study [5].

On the other hand, in IoT and DCN, most communications happen between adjacent and related entities. It is a good practice to locally confine communication, computing, and storage due to performance, efficiency, and security considerations, as advocated by Edge Computing [6]. This pattern provides an opportunity to overcome the overhead problem by designing and applying a more efficient address scheme.

B. Address Space Extension

What can we do about IPv6 address exhaustion? As we investigate the future, what if we need a parallel address space for the Interplanetary Internet. Such questions are not so far-fetched, and we need to be well prepared for an answer.

A fixed length address scheme only defines one monolithic address space. Each entity[1] is assigned a flat address as its

[1]Throughout the paper, we use the term "entity" to indicate any Internet addressable devices.

global identifier, which is used for communication with other entities. Because of this structure, the only way to combat future address exhaustion is to migrate to a larger address space and reassign longer addresses to the entities. This process introduces service disruption and delay due to necessary application, protocol stack, and global network updates.

A better approach is to allow any entity to keep its base address unchanged even when the address space is expanded. As far as an entity is concerned, nothing needs to be updated and it is business as usual. Address space expansion is transparent to all entities and even to most parts of the network. Fortunately, this is achievable with a simple and flexible address scheme, which can even be made compatible with IPv4 and IPv6.

C. Adaptive Address in Hierarchical Networks

In the era of Internet of Everything, it is natural and even preferred to contain and organize the directly related entities in isolated and hierarchical networks. Doing so not only ensures a securer network environment but also allows more efficient communications.

As a basic fact, although it is necessary for each entity to own a globally unique address to be able to communicate with others, an entity does not need to use the full address to communicate with other entities in the same network segment if they share a common prefix. It is enough to just use a locally unique address (*i.e.*, the global address suffix) for communication. Further, an entity does not even need to know its global address at all. For an entity to communicate with entities outside of its local network, it can still just use its local address as the source address and hand off the necessary address augmentation to the gateway routers connecting to external networks. This simple yet powerful idea allows short addresses for entities and implies a strict hierarchical network structure.

The hierarchical network and address assignment are not unfamiliar for the Internet. The original IPv4 address is designed to be classful [7]. However, due to the coarse granularity of the classes, the address block assignment is inefficient. The Internet has had to resort to Classless Inter-Domain Routing (CIDR) to slow down the address exhaustion [8]. It is no longer possible to maintain a cleanly organized network hierarchy. As a result, the backbone routing table size inflates. The design philosophy applies to IPv6 unchanged.

However, if we group entities into networks based on criteria such as geographic location, ownership, and logical relationship, and further group these networks into a higher level network (for convenience, we call this higher-level network the super-net of those networks it contains), and so on and so forth, we will end up with a clean network hierarchy and can use addresses as short as possible for communication. For example, if a lowest level network contains less than 256 entities, an 8-bit local address is enough to differentiate between all. When communicating with each other, they can simply use their local addresses.

All these entities attached to the same network share a same super-net prefix, which can be prepended before their local addresses to make them uniquely addressable in the network one level higher. This recursive process can be repeated until the top-level network is reached. A full and unique global address for each entity only reveals at the top level of the hierarchy.

The resulting network architecture deviates from the current network architecture in that an entity only knows its local address in its immediate network, and its global address prefix segments are kept by the hierarchical gateway routers to its higher level super-nets. With such an architecture, it is easy to expand the address space by simply adding a new level of super-net to cover the original top-level network. Now the original top-level network is demoted to a second level network, and other new networks with independent address space can be created alongside. It is critical to see that the existing network hierarchy remains intact and the existing entities are oblivious to the change, except that now global addresses are effectively extended by a new super-net prefix. Moreover, any entity's local address update is confined in the entity's immediate network and any prefix update is confined in the immediate super-net. Such features are ideal for maintaining a stable and evolvable network infrastructure.

However, to make such an idea work, we need to redesign the Internet protocol header (at least for the address part) and the way routers process and forward packets. We also need to consider the compatibility with IPv4 and IPv6, conceive a feasible deployment strategy, and present provable benefits, to overcome the resistance for adoption. In this paper, we provide the system design and P4-based data-plane implementation and evaluation. We demonstrate a clear evolution path for pre-standard incremental deployment of the new scheme, so IoT and DCN can enjoy its benefits instantly.

The remainder of the paper is organized as follows. Section II describes the address scheme. Section III describes the function and operation of routers with different roles. Section IV and V describe the corresponding control plane and data plane designs, respectively. Section VI presents the data-plane prototypes. Section VII evaluates the scheme and implementation. Section VIII summarizes the related works. Finally, Section IX concludes the paper.

II. Adaptive Address Scheme

To begin with, we eliminate the absolute maximum length restriction of a complete address. People might argue that 128-bit address as in IPv6 is more than enough if used with prudence. However, it is better to be flexible in address space size and be prepared for the extensibility in advance when designing new addressing schemes. Consider a scenario that the 128-bit address is used exclusively for entities on earth, and we need to allocate addresses for extraterrestrial entities in different address spaces. We can consider all the terrestrial entities to be in a single 128-bit network. Another unique super-net prefix needs to be appended to this local 128-bit address before a terrestrial entity can communicate with an extraterrestrial entity.

In an expandable hierarchical network, the “local” address of an entity is defined as its assigned address in its immediate network. This address is the shortest possible address for the entity. The entity does not know the super-net prefix for its immediate network. Although an entity does own a unique global address (*i.e.*, by concatenating its local address with all its prefix segments up to the top-level super-net), for two entities in different networks to communicate, they still do not need to know each other’s global address. Their global address suffixes in their lowest-level common super-net are enough for them to identify each other. We name these global address suffixes their relative global addresses. Under this definition, for two entities in the same immediate network, their relative global addresses are simply their local addresses.

In the network hierarchy, each network essentially owns a sub address space. A network can contain multiple entities and multiple other lower level networks. The only constraint is that all the entities and lower level networks in a network must be uniquely addressable.

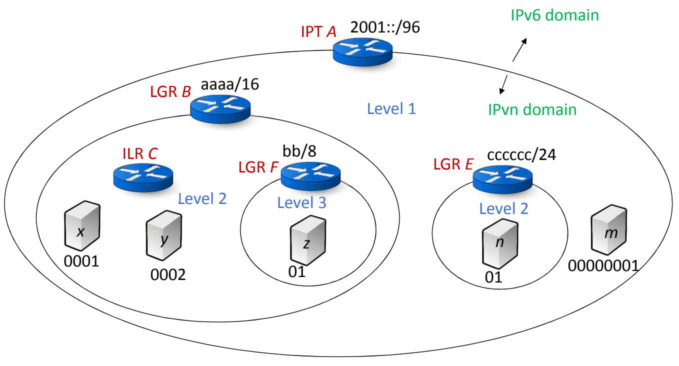

Fig. 1. Hierarchical network and address example. LGR stands for Level Gateway Router; ILR stands for Intra Level Router; IPT stands for IP Translator Router. All entity addresses are labeled in hexadecimal. Each gateway router is labeled with a super-net prefix for the network under it. For a prefix, before the slash is the prefix value in hexadecimal and after it is the prefix length in decimal.

Figure 1 shows a hierarchical network example and the corresponding address assignment. The Level 1 network, having a 32-bit address space, contains two lower level networks and an entity m. One Level 2 network has a 16-bit address space and contains two entities, x and y, and a Level 3 network; the other Level 2 network has an 8-bit address space and contains an entity n. The Level 3 network has an 8-bit address space and contains an entity z. For entities x and y, their immediate Level 2 network is their lowest-level common super-net. However, for entities x and n, the Level 1 network is their lowest level common super-net.

To entity x, entity y’s relative global address is simply its local address, `0x0002`. However, if entity z needs to communicate with x, since z and x’s lowest-level common super-net is x’s immediate network, z’s local address `0x01` must be augmented with its super-net prefix `0xbb` to become the relative global address to x. Similarly, for z to communicate with n, their relative global addresses are their augmented addresses in the Level 1 network. That is, z can be identified by n with the address `0xaaaabb01` and n by z with `0xcccccc01`.

Consider z needs to send a packet to n. z must first acquire n’s relative global address, presumably through the domain name service. Meanwhile, z only knows its own local address `0x01`. Carrying these addresses, the packet is forwarded to the LGR F. F needs to augment the source address with the prefix `0xbb` and continues to forward the packet to the LGR B. B further augments the source address with the prefix `0xaaaa`. Now since the source address and the destination address have the same length, the two entities must be in their lowest level common super-net. The packet is forwarded in this network and eventually reaches the LGR E. Before forwarding the packet into the Level 2 network where n resides, E will first prune the prefix `0xcccccc` off from the destination address because it is no longer needed and the remaining address `0x01` is enough to identify n in this network.

This example shows that variable-length addresses are involved in communications. Current IP protocols only use fixed-length addresses. The header of the next generation IP protocol (*e.g.*, IPvn) therefore needs to be amended to support variable-length addresses. In addition to the Source Address (SA) and the Destination Address (DA), it also needs to include the corresponding Source Address Length (SAL) and the Destination Address Length (DAL). One possible field layout is shown in Figure 2.

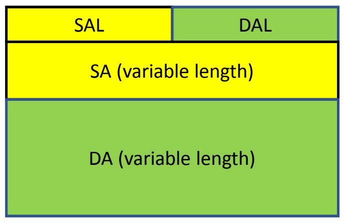

Fig. 2. A possible layout of address related fields in IPvn header.

The SAL and DAL are fixed length. To simplify the implementation, SA and DA are conventionally byte-aligned. It is possible to define the length of address in the unit of byte, nibble, or bit. Each has its own pros and cons. The unit of byte can help reduce the size of the SAL/DAL but results in coarse network granularity which might be inefficient in address allocation. For example, a 4-bit SAL or DAL is enough to encode 16 possible address lengths (up to 16 bytes) for networks by using the code’s equivalent decimal value (e.g., `0b0011` means three bytes; specifically, `0b0000` is considered as 16). In this design, each super-net is at least 256 times greater than a lower level network under it. On the other extreme, the unit of bit allows fine network granularity but requires more space for SAL/DAL. For example, 8-bit SAL and DAL can support an address length up to 256 bits and a super-net is only twice larger than a lower level network under it. Note that, with a few bits, it is possible to design a more sophisticated encoding scheme that supports variable address length steps and adapts to the ideal network sizes at different levels. We leave this topic for future study.

By any means, the header overhead saving due to the new address scheme is substantial. In Figure 1, for communication between x and y, the total address-related header overhead is at most 6 bytes, in contrast to 8 bytes for IPv4 and 32 bytes for IPv6. Consider the majority communications would happen locally for IoT, the saving can directly translate to resource, energy, and latency gains.

For communication crossing networks, the saving is still sizable due to the in-network address manipulations. Next we show the new scheme introduces minimum extra work for the gateway routers (*i.e.*, LGR and IPT). In general, the new scheme simplifies the entire network design from both hardware and software perspectives.

III. Router Role and Function

The networks can be nested in multiple levels. An entity only knows its local address in its immediate network. Each network has one or more Level Gateway Routers (LGR) which are responsible for forwarding packets in or out of this network. The LGRs are the only interface between a network and its super-net.

A network can be in a single L2 domain, which means all the entities in this network (excluding those in lower level networks) and all the network devices (including the LGRs to the super-net and the LGRs to the lower level networks) are L2 reachable through L2 switches. In this case, the network architecture makes some interesting difference on L2 address resolution as discussed in Section IV-A3.

A network can also be a pure L3 network, which means no L2 device is allowed. Each entity in a network is directly connected to either an LGR or some Intra-Level Router (ILR) which is solely responsible for packet forwarding within this network. In this case, the entities need to partially participate in the routing process (*e.g.*, advertising its address to the router it connects). If for some reason one decides to group some entities into a L2 network, then these entities must be put into a lower level network and deploy an LGR to serve it.

The scale of an intra-level network can be used to guide the L2/L3 selection. Small networks prefer the L2-based solution and large networks prefer the L3-based solution. In the higher level networks, since the number of entities is usually small, it is free to choose between L2 or L3-based solution. The lowest level network is usually L2-based.

We defer the discussion on entity address assignment and resolution to Section IV. For now, let's focus on the packet forwarding process, assuming SA and DA are known.

Unlike IPv4 and IPv6, the address related fields in IPvn header can be modified in network by LGRs. An LGR of a network keeps a prefix that can augment the SAs from this network to an address in the super-net. If an LGR needs to forward an internal packet outside (*i.e.*, DAL>SAL), it augments the packet's SA and updates its SAL accordingly. Reversely, if an LGR receives a packet destined for the lower level network it serves from the super-net (*i.e.*, the super-net prefix matches the DA's prefix), it strips off the super-net prefix from the packet's DA and updates its DAL accordingly.

In contrast, in an L3-based level network, ILRs do not modify the address fields. An ILR can decide the packet forwarding direction by simply examining the DAL. If DAL>SAL, the packet needs to be forwarded to an LGR of this network; otherwise, the packet needs to be forwarded within the current network, and possibly into a lower level network.

We use Figure 1 to illustrate some packet forwarding examples. We assume the address related field layout shown in Figure 2. We allocate generous 8 bits to encode the address length in bytes. Figure 3 shows that entity x sends a packet to y. Figure 4 shows that entity x sends a packet to n. In the figures, we only show the address-related fields and highlight their transformation at LGRs.

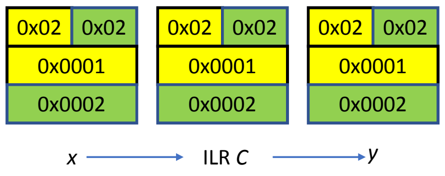

Fig. 3. x sends a packet to y, both in the same L3-based network.

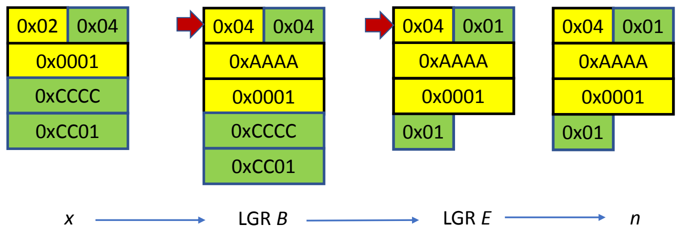

Fig. 4. x sends a packet to n in a different network. The Level 1 network is their lowest-level common super-net.

The type of communication shown in Figure 3 may account for most of the traffic for IoT or DCN. The size of the address-related fields is kept to the minimum and there is no address transformation involved. The example in Figure 4 is more complicated and interesting. Since DAL>SAL, the packet is forwarded to LGR B, at which SA of the packet is augmented and the packet enters the Level 1 network. Now since SAL=DAL, the packet is forwarded in the current network and eventually reaches LGR E. LGR E prunes DA of the packet and forwards the packet into the Level 2 network and eventually to n. n can directly talk back to x because from the packet, n can acquire x's relative global address.

A. Interface with Existing IPv4/IPv6 Networks

One reason it is difficult to introduce new architectures and protocols into today's Internet is due to the lack of a backwards-compatible incremental deployment strategy [9]. An end-to-end overhaul is economically unjustifiable. To make the new address scheme acceptable and deployable, we need to consider how we can gracefully introduce it to the Internet and how to make it seamlessly work with existing IPv4 or IPv6-based networks. Fortunately, it is straightforward for the new address scheme to be compatible with IPv4/IPv6 addresses and for IPvn to interface with existing IPv4/IPv6 networks.

We discuss two scenarios. First, the whole IPv4 (IPv6) network space is considered a 32-bit (128-bit) lowest level network in IPvn (*i.e.*, IPv4/IPv6 in IPvn). Within the IPv4 (IPv6) network, the communications continue to use the IPv4 (IPv6) protocol. However, if an entity in the IPv4 (IPv6) network needs to communicate with outside entities, they need to switch to use the IPvn protocol at the gateway between the two types of networks. This idea is not new. The current IPv6 address allocation makes IPv4 a sub-net of IPv6 and allows the translation from IPv4 address to IPv6 address by augmenting the IPv4 address with the prefix `::ffff/96` [10]. This scenario adopts a similar approach, which enables the conventional entities with IPv4 or IPv6 addresses to retain their addresses and communicate with the entities in networks running IPvn. This approach keeps the current IPv4 and IPv6-based networks intact and grows the IPvn-based networks alongside.

The second scenario (*i.e.*, IPvn in IPv4/IPv6) allows the support of IPvn within the IPv4 (IPv6) address space. In this scenario, the IPv4 (IPv6) network is considered the top level super-net and the 32-bit (128-bit) addresses cover the entire address space, in which we can construct hierarchical lower level networks to support IPvn. This scenario contains two sub cases.

Case 1: Each top-level IPvn network is assigned a unique prefix to extend the local addresses of the entities in it to full IPv4/IPv6 addresses. This means that each IPvn network occupies a sub address space and each entity in the IPvn network owns a full IPv4/IPv6 address. The LGR of the top-level IPvn network (named IP Translator or IPT) is responsible for protocol translating between IPvn and IPv4/IPv6. Inside of the network, IPvn is used; outside of the network, IPv4/IPv6 is used. Figure 1 illustrates an example of this case. The top-level IPvn network is a 32-bit subnet of the IPv6 network. IPT A is the interface between the IPvn domain and the IPv6 domain. A unique 96-bit prefix `2001::/96` is assigned to the top-level IPvn network.

Case 2: Each top level IPvn network is assigned one or more public IPv4/IPv6 addresses. The entities in the IPvn network are assigned with private IPvn addresses. In this case, LGR of the top-level IPvn networks serves as not only a protocol translator but also a Network Address Translation (NAT) server.

While both scenarios make IPvn compatible with IPv4/IPv6, Case 1 of the second scenario is especially attractive because it supports a simple, straightforward, and incremental deployment strategy, that helps to introduce IPvn into the current Internet with the least resistance and the most benefits. The edge networks with arbitrary scale can immediately transfer to IPvn while the core networks still run IPv4 or IPv6. The penetration to the core can take a slower pace based on the standard maturity and the demand curve.

For successful protocol translation, the IPvn header needs to maintain other information than the address-related fields to match the IPv4/IPv6 header. The specification of the IPvn header is beyond the scope of this paper, although in our prototype discussed in Section VI, we do propose a preliminary IPvn header format supporting IPv4/IPv6 translation.

IV. Control Plane Design

It is conceivable that all the control plane functions and protocols need to be modified or redesigned due to the hierarchical network architecture of IPvn. Fortunately, the updates are often incremental and the results are not more complex and often simpler than their counterparts in IPv4 and IPv6. We cannot cover all the control plane functions and protocols in one paper. Instead, we discuss a few essential ones that are sufficient for us to implement a working prototype or show interesting and unique new properties.

A. Address Configuration and Resolution

1) DHCP: The first issue is how an entity acquires its own address after booting. An entity's address can be manually configured or dynamically acquired. Each network can contain a Dynamic Host Configuration Protocol (DHCP) server which is responsible for assigning addresses for the entities in the same network. The DHCP server maintains a pool of assignable addresses for its local address space. The protocol is almost identical to that for IPv4 and IPv6, except that the assigned address length is adaptive to the network size.

2) DNS: The second issue is how an entity acquires the address of a peer entity in order to initiate a communication. Domain Name System (DNS) is still a prominent approach for this but with a new service model. Any network can provide name service. Each entity is configured with the address of the closest DNS server. The hierarchical network architecture allows a scoped domain name service. That is, a name registered in a network can only bind to an entity covered by the network. Also, the name-address binding is only valid and accessible to the entities covered by the network. It is possible that a same name is registered in two networks and one network is the other's super-net. Such name conflict is not a bug but a feature. It allows name reuse. The name queries process can naturally resolve the conflict.

Each network may contain a DNS server[2]. Each DNS server knows the nearest DNS server in a higher level network and the nearest DNS servers in lower level networks. This essentially organizes the DNS servers in the network hierarchy into a tree structure rooted at the top-level network. Each named entity in a network is registered with the DNS server that covers its scope, which is basically a subtree.

Figure 5 shows an example of DNS server tree, in which each node represents a DNS server in a network. Assume a name $x.y.z$ binds to an entity in the network whose DNS server is f and the name's scope is as shown in the figure.

We have several methods to populate the name to support the scoped name queries. 1) Register the name in all the DNS servers in its scope (*i.e.*, all the subtree nodes). In this case,

[2]The notation is only logical. The actual implementation may follow the same hierarchical and distributed architecture of today's DNS.

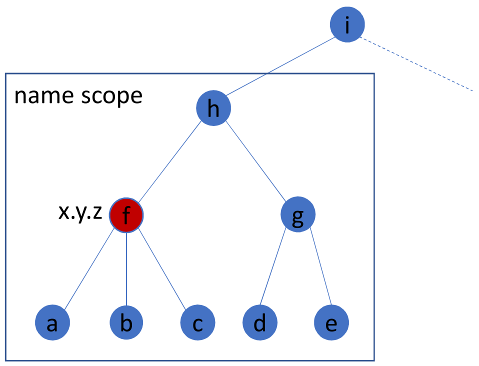

Fig. 5. DNS server tree and scoped name query in hierarchical networks.

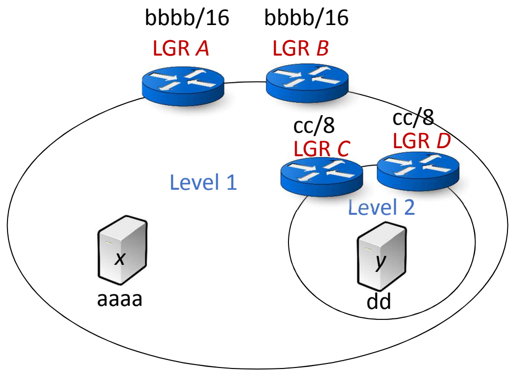

Fig. 6. A L2-based network with multi-homing gateway routers.

each entity only needs to query its configured DNS server to resolve a name. If the name is not there, it is safe to conclude that the name is not valid in this network. This method is fast but introduces redundancy due to name replication. 2) Recursively register the name in every parent DNS server until the scope root (*i.e.*, f and h). In this case, each entity needs to recursively query the DNS servers along the path to the tree root to resolve the name. The search stops when a server returns an answer, or the tree root is reached with no resolution. The query and answer are relayed between the adjacent DNS servers. A valid answer can be cached to ease future queries for the same name from other entities. 3) Register the name only in the DNS server in its scope root (*i.e.*, h). The query process is similar to the second case. This method scales the best. The query performance can also be improved by caching the results on the returning path.

The address for a name returned by a DNS server is also on a "need-to-know" basis. That is, in a network, if the address's prefix matches it super-net prefix, the prefix is removed. This can be easily done by the original or the relay DNS servers. If a query crosses the IPvn domain and enters into the IPv4/IPv6 domain, the protocol translation is also needed.

3) ARP: For a link or in a L2-based network, Address Resolution Protocol (ARP) or Neighbor Discovery Protocol (NDP) are needed to resolve addresses between L2 and L3. The operation of these protocols is almost identical to those for IPv4 and IPv6.

If a network is L2-based, each immediate entity in it should be configured with a default gateway address to its super-net. If no default gateway is configured, a network LGR should be configured as an ARP proxy to respond to all internal ARP requests for addresses out of the network. Similarly, the LGRs of any lower level network in this network are also needed to be configured as ARP proxy to response all ARP requests for addresses in the lower level network.

As shown in Figure 6, the Level 1 network has a two-byte address space and its super-net prefix is `0xbbbb`; the Level 2 network has a one-byte address space and its super-net prefix is `0xcc`. Each network has two LGRs. Entity x's default gateway could be either LGR A or LGR B. One or both of the LGRs can be configured as ARP proxy for addresses outside of the Level 1 network. Similarly, LGR C and/or LGR D should be configured as ARP proxy for ARP requests for addresses that have the prefix `0xcc`. Due to the multi-homing gateway routers, an ARP request may receive multiple responses. It is up to the requester to determine which one to cache.

B. Routing Protocol

The address aggregation due to the hierarchical network architecture also benefits the routing protocols.

A lower level L3-based network may belong to a single organization or Autonomous System (AS), so the interior gateway routing protocols (IGP) such as OSPF and IS-IS (modified to run under IPvn) can be used. Other lower level networks in this network can be considered OSPF stub areas or IS-IS levels. A simpler way is that each network run an independent instance of OSPF or IS-IS. In this, an LGR runs two OSPF/IS-IS instances: one for the super-net and the other for the lower level network.

On the other hand, a higher level L3-based network may contain multiple ASes and forms the backbone of the lower level networks. Like today's Internet, the ASes could come from multiple Internet Service Providers (ISP) with peering relationship. Each lower level network becomes a stub AS. In this case, the exterior gateway routing protocol (EGP) such as BGP (modified to run under IPvn) can be used.

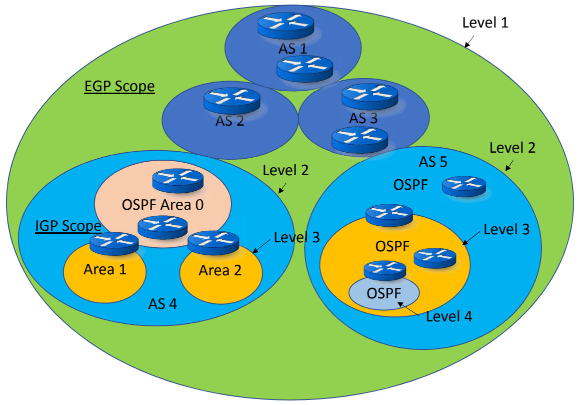

Fig. 7. Routing in hierarchical networks.

Figure 7 shows a network with four levels of L3-based networks. The Level 1 network contains five ASes. Two of the stub ASes are the Level 2 networks; the other three ASes provides connections for the Level 1 network and the outside networks. An EGP instance runs in the Level 1 network. Each Level 2 network contains some Level 3 networks and runs an IGP instance, such as OSPF. Particularly, in the left Level 2 network runs only one OSPF instance. Each Level 3 network is an OSPF stub area and the remaining network forms the backbone area. In contrast, in the right Level 2 network and each lower level network within it runs an independent OSPF instance.

The hierarchical architecture solves the routing protocol scalability issue, and simplifies the protocol implementation by removing unnecessary features. The clean routing scope helps to secure the infrastructure and troubleshoot the networks.

V. Data Plane Design

A. *IPvn Socket for End Entities*

Enabling IPvn as another network layer (*i.e.*, L3) protocol in end entities is straightforward. We need to add the IPvn protocol implementation in the OS Kernel and allow applications to invoke the socket API using the new address family parameter `AF_INETN`. The existing L4-L7 protocol stack and the application logic do not need to change. This is important to allow the entities in IPvn domain to communicate with entities in IPv4/IPv6 domain.

B. *Forwarding Table Lookups in Networks*

The adaptive address scheme significantly simplifies the router forwarding table structure in every L3-based network. As a result, the forwarding chip is simpler and the packet forwarding performance is improved on both throughput and latency.

In each network, the forwarding table only contains the addresses to local entities and the prefixes to the lower level networks (note that the default route to the super-net LGR is not maintained in the forwarding table. The choice of using it solely depends on the packet DAL). It is easy to see that the forwarding table in each ILR of a network contains no nested prefixes. That is, in the corresponding prefix trie, only the leaf nodes are valid prefixes. This effectively nullifies the need of Longest Prefix Matching (LPM) which is especially challenging for large scale IPv6 prefix tables [11].

More important, the number of unique prefix lengths is small. In the lowest level networks, only exact-length addresses are present. Therefore, the prefixes can be grouped into a few sets based on their lengths. Each set can be implemented as a hash table. For a forwarding table lookup, all the hash tables are searched in parallel. Only one table can return a positive match which is the lookup result. This design avoids the use of expensive TCAM or other complex trie-based algorithms [12], [13]. Instead, optimizations such as Bloom Filters [14] or Fast Hash Table [15] can be applied to improve the hardware-based hash table lookup performance.

An LGR has two types of interfaces: one faces the lower level network it serves, and the other faces the super-net. It is possible for one LGR to serve more than one lower level network. Hence, logically, an LGR may contain multiple forwarding tables, with each for a network (including the super-net). For a packet, once its target network is determined and the address related fields are processed, the proper forwarding table is searched.

VI. Implementation

In order to prototype the system, we draft an IPvn header format as shown in Figure 8. The header contains enough information to allow the IP header translation to and from IPv4 and IPv6. We use P4 [16] for prototyping.

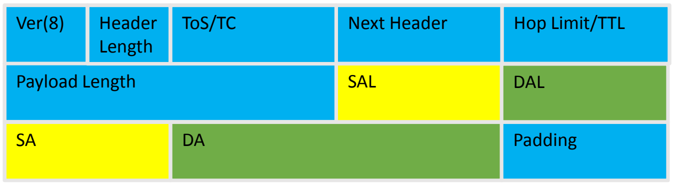

Fig. 8. A proposed header format of IPvn prototype.

Compared to IPv4 and IPv6, the variable-length addresses in IPvn pose the main challenge in the implementation, especially considering the limited support of variable-length header in P4 language. One dynamic-size data type offered by P4 is “varbit”, but most operations like addition, subtraction, arithmetic shift are not applicable to it, which limits its usability. Alternatively, we use the header stack, an array of fixed-length headers [17]. The dynamic address and padding are split into multiple 8-bit elements, which can be pushed and popped in a header stack, and the complete address can be recovered by concatenating those multiple 8-bit elements. The header parsing graph is shown on the left of Figure 9. A metadata is used to track the remaining bytes from SAL/DAL and the address is recursively extracted. On the right of Figure 9, an enumerate way of implementation is illustrated instead of recursion. Different lengths supported in the router are all declared in the header (*e.g.*, one-/two-/four-byte addresses), so an address extraction takes just one state transition.

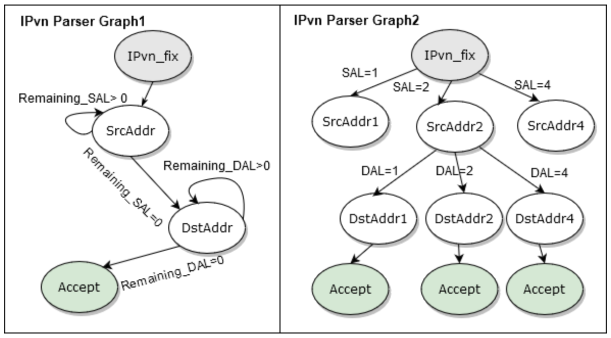

Fig. 9. Two options of IPvn header parser graph implementation in P4.

Algorithm 1: ILR forwarding flow

```
Input: pkt, level.length
1  if $pkt.dal < level.length \mid\mid pkt.sal < level.length$ then
       /* exception                                    */
2      drop pkt;
3  if $pkt.dal == level.length$ then
       /* forward within the network                   */
4      search the level network forwarding table;
5      forward pkt;
6  else
       /* forward out of the network                   */
7      forward pkt towards the level network LGW;
```

The computation overhead is related with P4 program's features and parameters, *e.g.*, the number of header fields, branches in parse graph, tables, and header adds/removes for packet modification. More detailed metrics can be found in [18]. Since header parsing is the most important change in router functions, both parser graphs are evaluated. We find the second graph is more efficient.

The implementation framework for our P4-based prototype is shown in Figure 10. The routers, including ILR, LGR, and IPT, are written in P4-14 language [16]. The packet processing logic of ILR, LGR, and IPT is shown in Algorithm 1, 2, and 3, respectively. Through the P4 compiler, *p4c*, the programs are compiled and installed to the BMv2 software switch. Forwarding tables and switch parameters can be pre-loaded or dynamically changed using *Simple_Switch_CLI* during run-time. The customized switches run on a configured network topology in the Mininet emulation environment [19]. The whole environment runs as a docker container from the open-source tool p4app [20], which is easy to launch and repeat.

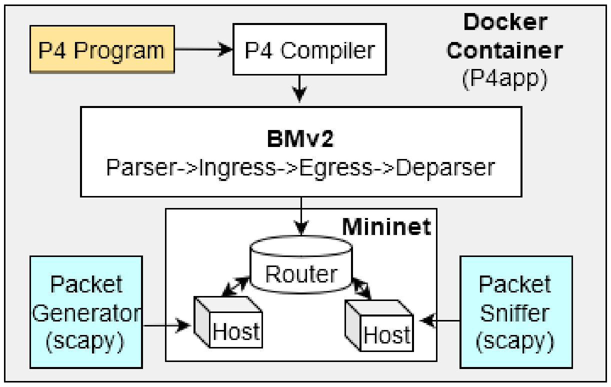

Fig. 10. Implementation and simulation environment.

We open source the entire project [21]. Currently, it contains the network data-plane prototype and simulation environment. We are working on the new L3 support in end entities, hardware-based prototypes, and control-plane implementations, which will be included in the project eventually.

VII. Evaluation

Evaluation runs on an Ubuntu 18.04 server with 10-core CPU at 2.2GHZ and 64GB RAM. Since BMv2 is not meant

Algorithm 2: LGR forwarding flow

```
Input: pkt, level.length, supernet.prefix, supernet.length
1  if $pkt.dal < level.length \mid\mid pkt.sal < level.length$ then
       /* exception                                    */
2      drop pkt;
3  if $pkt.sal < pkt.dal$ && $pkt.sal < supernet.length$ then
       /* source address processing                    */
4      augment pkt.sa with supernet.prefix;
5      pkt.sal = supernet.length;
6      update pkt.header_length if needed;
7  else if $pkt.dal == supernet.length$ && $pkt.da.prefix ==$
     $supernet.prefix$ then
       /* destination address processing               */
8      remove supernet.prefix from pkt.da;
9      pkt.dal = level.length;
10     update pkt.header_length if needed;
11 if $pkt.dal == level.length$ then
       /* forward in lower-level network               */
12     search the level network forwarding table;
13     forward pkt;
14 else if $pkt.dal == supernet.length$ then
       /* forward in the super-net                     */
15     search the supernet forwarding table;
16     forward pkt;
17 else
       /* forward out of the super-net                 */
18     forward pkt towards the supernet LGW;
```

Algorithm 3: IPT forwarding flow

```
Input: pkt, level.length, supernet.prefix
1  if $pkt.ver == 8$ then
2      if $pkt.dal < level.length \mid\mid pkt.sal < level.length$ then
3          drop pkt;
4      if $pkt.sal < pkt.dal$ then
5          augment pkt.sa with supernet.prefix;
6          covert header to IPv6;
7          forward packet in IPv6 network;
8      else
9          forward packet in IPv8 network;
10 else if $pkt.ver == 6$ then
11     if $pkt.da.prefix == supernet.prefix$ then
12         remove supernet.prefix from pkt.da;
13         convert header to IPv8;
14         forward packet in IPv8 network;
15     else
16         forward packet in IPv6 network;
```

to be a production-grade software switch and the whole network simulation runs as a docker process, the absolute time calculated in the experiments is not our focus. We aim to prototype the new protocol and explore the relative performance difference under different implementations.

Scapy [22] is used to generate and sniff packets. New protocols can be easily defined using the *Packet* class provided by Scapy, and the function *sendp* is used to send layer 2 packets. The performance of sending and receiving packets with Scapy varies significantly depending on how it is used. The sending and receiving scripts need to be carefully configured to eliminate the noise in the evaluation of forwarding performance in P4 switches. We document our finding and experience in the project website [21].

A. Software Switch Forwarding Performance

To evaluate the forwarding performance, two methods are available to calculate processing time per packet in current simulation environment. The first one uses the timestamps on the interfaces from the Pcap Tool, which records the time packets enter/leave the interfaces; the second one uses the timestamps in the standard metadata from BMv2 architecture, often used for in-band telemetry (INT) [23], which records the time packets enter the ingress/egress processing logic. Both timestamps are in microsecond precision and can reflect the packet processing complexity. The time calculated by the first method is longer than the second one, since it covers more logic. The comparison of these two methods for three types of packets in one switch is shown Fig. 11. One hundred packets of IPv4, IPv6 and IPvn were sent accordingly through three *sendp* functions in one script. We observe that a roughly constant time difference about 140 μs exists between these two metrics. For the rest of experiments, to be less intrusive, the timestamps on interfaces are used to calculate packet processing time. In Fig. 11, the first packet's processing time is significant longer than the average is due to the start of python packet sending script.

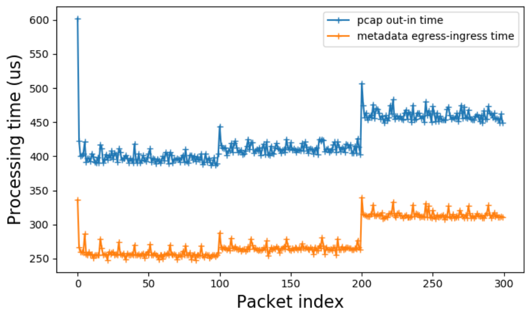

Fig. 11. Comparison of two metrics for packet processing time.

We evaluate the two different IPvn header parser implementations shown in Fig. 9 for ILR under four different address lengths. Two unassigned EtherType numbers (`0x8881`,`0x8882`) are used to select an active parser in an ILR. Packets are transmitted between two hosts connected to the ILR. Both hosts can send and receive packets with one-/two-/four-/eight-byte length of source and destination addresses. 1,000 packets are transmitted at each configuration. The address length of ILR can be modified with runtime *Simple_Switch_CLI*. The processing time per packet is shown in Fig. 12. For Parser1, the parsing step increases as the length of the address goes up, so does the processing time. The median processing time varies from 588μs to 967μs. Parser2 has a fixed steps of parsing, so its median processing time only varies from 503μs to 514μs. It scales better when the address length increases. Another interesting observation is that the processing of one-byte address is not faster than the two-byte address. This is because one-byte address triggers an extra padding parsing state, based on the header format shown in Fig.8. Since the second parser implementation is more efficient and scalable, we adopt parser2 for the rest of the experiments. In the later experiments we remove the logic of the first parser from ILR, so the ILR's processing time for a 2-byte address is improved to about 405μs.

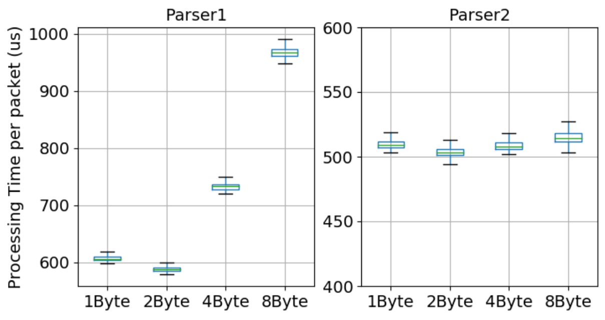

Fig. 12. Comparison of two parser implementation.

The forwarding performance of ILR, LGR, and IPT is evaluated and processing time per packet is shown in Fig. 13. There are six columns in the boxplot, the first one is ILR forwarding within a network; the second one is ILR forwarding toward super-net (this is faster because no table lookup is needed); the third one is LGR forwarding for packets entering a lower level network; the fourth one is LGR forwarding for packets entering a super-net; the fifth one is IPT for IPv6-to-IPvn conversion; the sixth one is IPT for IPvn-to-IPv conversion. According to the Algorithm 1, 2, and 3, LGR requires the most processing, including header parsing, modification, and forwarding table selection and matching. The evaluation results match the analysis. For IPT, IPvn-to-IPv6 is slower than the reverse conversion due to the extra parsing steps for IPvn header.

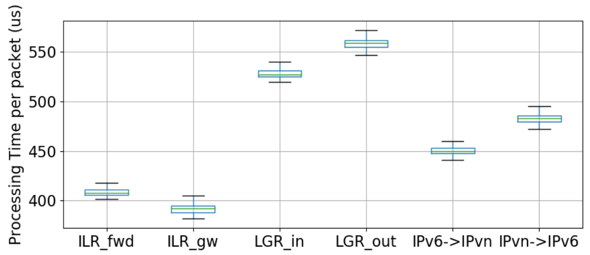

Fig. 13. Processing time per packet of ILR, LGR and IPT

Lastly, we compare the processing time of IPvn with the basic IPv4 and IPv6 header (*i.e.*, no option and extension header). 1000 packets of IPv4, IPv6 and IPvn are forwarded separately. Forwarding table is shown in Table I. Forwarding performance is consistent with what is shown in Fig. 11. The median of the processing time among the 1000 packets for IPv4, IPv6, IPvn is 344 μs, 368 μs and 410 μs separately. The slightly longer processing time of IPvn is due to the lack of efficient support on variable-length header fields in P4. The future revision of P4 should consider to amend this. In addition, our experiments use a small forwarding table with a few entries, so the table lookup cost is negligible. In reality, complex data structure and lookup algorithms are needed for large forwarding tables in IPv4 or IPv6, and the table lookup performance is usually the performance bottleneck in software switches. Fortunately, there is no prefix overlap in IPvn and the number of prefix lengths in a network is small. Therefore exact match (EM) can be used for table lookup instead of longest prefix match (LPM), which is simpler and faster. This can offset the cost of header processing and contribute to a much higher overall forwarding performance. In our experiments, processing time per packet with largely overlapped 256 prefix entries in a LPM table can be slowed down by 40 μs, compared to that with a small 2-entry LPM table. In contrast, a small (fewer than 10 entries) size EM table is about 10 μs faster than a same size LPM table. To make fair lookup comparison and obtain consistent results, it is better to create multiple headers and apply different tables to different headers, instead of selecting tables based on header fields with *if-else* branches in a P4 program.

TABLE I
SIMPLIFIED FORWARDING TABLE FOR IPV4, IPV6, IPVN

Simple Switch Command					
table_add	ipv8_ilr_lpm	fwd	0x0001/16	=>	1
table_add	ipv8_ilr_lpm	fwd	0x0002/16	=>	2
table_add	ipv6_lpm	fwd	2001::aaaa:0001/128	=>	1
table_add	ipv6_lpm	fwd	2001::aaaa:0002/128	=>	2
table_add	ipv4_lpm	fwd	0xaaaa0001/32	=>	1
table_add	ipv4_lpm	fwd	0xaaaa0002/32	=>	2

The above evaluation only applies to the software switch and router. In the more prevailing hardware, the extra header processing in ILR and LGR only needs a few extra pipeline stages which slightly increase the forwarding latency but have no influence on throughput. The chip die size saved due to the compact forwarding table size and efficient forwarding table implementation for IPvn can directly translate into higher I/O bandwidth and/or deeper buffer.

B. Overhead

The packet overhead saving by using the adaptive address is substantial. Compared with IPv6, the address-related overhead saving is from 87.5% to 68.8% when the network size is from 1 Byte (*i.e.*, up to 256 entities) to 4 Bytes (*i.e.*, up to 4 billion entities). If the entire IP header is considered, the overhead saving is between 70% and 60%, as shown in Figure 14.

C. Power

Networking is responsible for more than 80% of the total power consumption for wireless IoT devices due to the radio and processing [24], [25]. It is difficult to measure the actual networking power consumption without a production environment. However, it is well established that, at a bandwidth C, the networking power consumption is, $P(C) = P_i + E_bC$, in

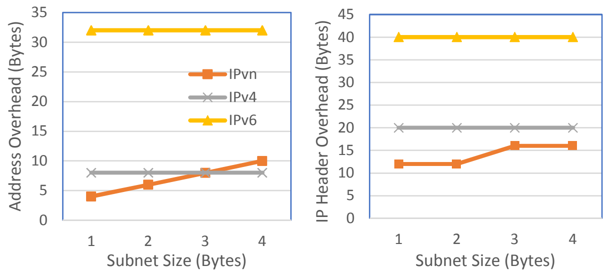

Fig. 14. Overhead comparison: address fields only and IP header.

which P_i is the idle power and E_b is the energy for per-bit transmission [26]. P_i usually consumes less than 10% of the power [25] and for a highly shared IoT networking device, P_i is negligible [26].

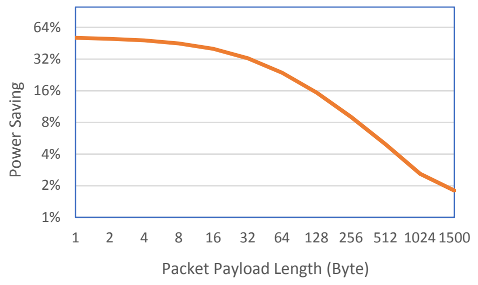

Fig. 15. Power saving over IPv6-based IoT, with the assumption of a 14-byte MAC header and up to 64K IoT entities in a network.

Figure 15 shows the power saving for IPvn over IPv6-based IoT solution. Given the majority packet payload size ranges from a few bytes to a few tens of bytes, the power saving is between 20% to 50%.

VIII. RELATED WORK

Although ATM PNNI [27] adopts hierarchical network and addressing but full addresses are always used. Many IP header compression techniques and protocols have been standardized [28]–[30]. The general principle is to use a full IP header to establish a context so that the subsequent packet headers only contain the incremental changes to the context. These techniques are mostly link based and require more states and processing in router. They are not scalable for lots of small flows. In case such a scheme is applied, our address scheme is still beneficial as an orthogonal optimization.

To support IPv6 over low power wireless personal area network, 6LoWPAN [31], [32] is a standard protocol to support L3 to L2 mapping and allow IPv6 packets with compressed header to be sent over IEEE 802.15.4 based networks. 6LoWPAN is clearly an IoT scenario we are concerned, but the solution is not general enough to be applied to other scenarios, let alone the end-to-end support.

A different variable-length IP address encoding scheme was proposed in [33]. It does not describe the network architecture, router functions, and deployment strategy as in this paper. Modification of L3 header addresses in routers during the packet forwarding is not unusual. Previous examples include NAT, L3 tunnelling, SRv6 [34], and RSIP [35]. However, our scheme is the first that modifies the address length.

IX. Closing Remarks

To combat the Internet ossification, a new layer 3.5 approach was proposed to provide a evolution path for Internet by allowing new network protocols at AS edges [9]. Our principle differs in that we start the evolution from the network edge, because each edge network is naturally a single management domain and it has a clean interface with external networks. Such an approach has been used in advancing the data center network protocols (*e.g.*, DCTCP [36] and RoCE [37]). Similarly, 6LoWPAN also innovates at the IoT edge.

Our solution does not stop at the edge by providing a clear evolving path to expand the scope of the new protocol towards the core and make the entire Internet extensible without an artificial address space boundary. Such an approach allows the pre-standard deployment at edge and the use of IPT gateways to interface with existing IPv4 or IPv6 networks in the core. The places where the new address scheme is mostly appreciated can enjoy the benefits immediately.

The advantages of the new address scheme include, but not limited to, power saving, effective bandwidth improvement, simplified data plane, simplified control plane, and boundless address space extension. The benefits of a hierarchical network architecture, because of the adaptive IP address, is more profound. It introduces scope to the domain name system (DNS) and hierarchy to the autonomous system (AS), which provide better system scalability, security isolation, policy management, and network robustness.

Many open issues are untouched in this paper and left as future work, such as IP mobility and multicast/broadcast. Obviously, the change of Internet architecture, even just partially and incrementally, needs to leave no stone unturned. By releasing the open source project and sharing our preliminary results, we hope our work can trigger more research and development from academia and industry.

References

[1] Statista Research Department, "Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025." https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/, 2019.

[2] RIPE NCC, "The RIPE NCC has run out of IPv4 Addresses." https://www.ripe.net/publications/news/about-ripe-ncc-and-ripe/the-ripe-ncc-has-run-out-of-ipv4-addresses, 2019.

[3] T. Narten, G. Huston, and L. Roberts, "IPv6 Address Assignment to End Sites," RFC 6177, IETF, 2011.

[4] A. Mayer, E. Altomare, S. Salsano, F. L. Presti, and C. Filsfils, "The Network as a Computer with IPv6 Segment Routing: a Novel Distributed Processing Model for the Internet of Things," in *1st International Workshop on Next-Generation Operating Systems for Cyber-Physical Systems (NGOSCPS)*, 2019.

[5] D. Zhuo, K. Zhang, Y. Zhu, H. H. Liu, M. Rockett, A. Krishnamurthy, and T. Anderson, "Slim: OS kernel support for a low-overhead container overlay network," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, 2019.

[6] M. Satyanarayanan, "The Emergence of Edge Computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[7] ISI, "Internet Protocol," RFC 791, IETF, 1981.

[8] Y. Rekhter and T. Li, "An Architecture for IP Address Allocation with CIDR," RFC 1518, IETF, 1993.

[9] J. McCauley, Y. Harchol, A. Panda, B. Raghavan, and S. Shenker, "Enabling a Permanent Revolution in Internet Architecture," in *Proceedings of ACM SIGCOMM*, ACM, 2019.

[10] M. Blanchet, "Special-Use IPv6 Addresses," RFC 5156, IETF, 2008.

[11] T. Stimpfling, N. Bélanger, J. M. P. Langlois, and Y. Savaria, "SHIP: A Scalable High-Performance IPv6 Lookup Algorithm That Exploits Prefix Characteristics," *IEEE/ACM Transactions on Networking*, vol. 27, no. 4, pp. 1529–1542, 2019.

[12] T. Yang, G. Xie, Y. Li, Q. Fu, A. X. Liu, Q. Li, and L. Mathy, "Guarantee IP lookup performance with FIB explosion," in *ACM SIGCOMM*, 2014.

[13] H. Asai and Y. Ohara, "Poptrie: A Compressed Trie with Population Count for Fast and Scalable Software IP Routing Table Lookup," in *ACM SIGCOMM*, 2015.

[14] S. Dharmapurikar, P. Krishnamurthy, and D. Taylor, "Longest Prefix Matching using Bloom Filters," in *ACM SIGCOMM*, 2003.

[15] H. Song, S. Dharmapurikar, J. S. Turner, and J. W. Lockwood, "Fast Hash Table Lookup Using Extended Bloom Filter: an Aid to Network Processing," in *ACM SIGCOMM*, 2005.

[16] P4 Language Consortium, "P4 Language and Related Specifications." https://p4.org/specs/.

[17] P4 Variable Length Header. https://github.com/jafingerhut/p4-guide/tree/master/variable-length-header.

[18] H. T. Dang, H. Wang, T. Jepsen, G. Brebner, C. Kim, J. Rexford, R. Soule, and H. Weatherspoon, "Whippersnapper: A P4 Language Benchmark Suite," in *Proceedings of the Symposium on SDN Research (SOSR)*, ACM, 2017.

[19] Mininet, "An Instant Virtual Network on your Laptop (or other PC)." http://mininet.org/.

[20] P4app Simulation Tool. https://github.com/p4lang/p4app.

[21] Futurewei, "Adaptive IP Address." https://github.com/Fizzbb/ResearchPaper/tree/master/Adaptive-Addresses-for-NG-IP, 2020.

[22] Python Scapy. https://scapy.readthedocs.io/en/latest/build_dissect.html.

[23] M. Hira and L. Wobker, "Improving Network Monitoring and Management with Programmable Data Planes." https://p4.org/p4/inband-network-telemetry/.

[24] B. Martinez, M. Montón, I. Vilajosana, and J. D. Prades, "The power of models: Modeling power consumption for iot devices," *IEEE Sensors Journal*, vol. 15, no. 10, 2015.

[25] S. Zhao, P. V. Rengasamy, H. Zhang, S. Bhuyan, N. C. Nachiappan, A. Sivasubramaniam, M. T. Kandemir, and C. Das, "Understanding energy efficiency in iot app executions," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, 2019.

[26] C. Gray, R. Ayre, K. Hinton, and R. Tucker, "Power consumption of iot access network technologies," in *Workshop on Next Generation Green ICT, IEEE ICC 2015*, 2015.

[27] ATM Forum, "Private NetworkNetwork Interface Specification, Version 1.0," *af-pnni-0055.00*, 1996.

[28] V. Jacobson, "Compressing TCP/IP Headers for Low-Speed Serial Links," RFC 1144, IETF, 1990.

[29] M. Degermark, B. Nordgren, and S. Pink, "IP Header Compression," RFC 2507, IETF, 1999.

[30] S. Casner and V. Jacobson, "Compressing IP/UDP/RTP Headers for Low-Speed Serial Links," RFC 2508, IETF, 1999.

[31] J. Hui and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks," RFC 6282, IETF, 2011.

[32] C. Bormann, "6LoWPAN-GHC: Generic Header Compression for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)," RFC 7400, IETF, 2014.

[33] S. Ren, D. Yu, G. Li, Y. Tian, X. Gong, S. Hu, and R. Moskowitz, "Routing and Addressing with Length Variable IP Address," in *Proceedings of ACM SIGCOMM Workshop on Networking for Emerging Applications and Technologies (NEAT)*, ACM, 2019.

[34] C. Filsfils, D. Dukes, S. Previdi, J. Leddy, S. Matsushima, and D. Voyer, "IPv6 Segment Routing Header (SRH)," RFC 8754, IETF, 2020.

[35] M. Borella and J. Lo, "Realm Specific IP: Framework," RFC 3102, IETF, 2001.

[36] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," in *ACM SIGCOMM*, 2010.

[37] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn, "RDMA over Commodity Ethernet at Scale," in *ACM SIGCOMM*, 2016.