IPswen: A Long Term Evolution Approach for the IPv4 Internet Architecture

Bing Swen Sun
School of Computer Science, Peking University, Beijing 100871
Key Laboratory of Computational Linguistics (Peking University), Ministry of Education,China
bswen@pku.edu.cn

***Abstract*—After decades of IPv6 development, it is clear that the "legacy" IPv4 protocol stack and the TCP/IPv4 architecture will coexist with IPv6 in the foreseeable future, even if IPv6 deployment or availability rate is reaching 100%. This paper presents a practical approach for the long term coexistence and evolution of the TCP/IPv4 architecture. The core idea is to adopt a variable length addressing and routing scheme (IPswen) to preserve the entire IPv4 stack with bidirectional interworking. Under strict design constraints, IPswen is an incrementally deployable and binary backwards compatible (evolutionary) solution to upgrade existing IPv4 networks, with a sufficiently large address space for long term evolution, which is a multilevel L3 address space that consists of 8 subspaces of address lengths ranging from 4 to 11 bytes. Results of a single-stack Linux implementation demonstrate its usability and binary code compatibility with TCP/IPv4 and existing socket API applications. We argue that IPv4 still possesses a potential for substantial extension after adopting CIDR and NAT44, and IPswen might be in a much more promising direction for the continued evolution of the TCP/IPv4 architecture than the conventional wisdom that we need to transit everything of the Internet from IPv4 to IPv6.**

I. INTRODUCTION

It may seem to be a remarkable fact to the networking research community and industry that presently the global Internet is still dominated by IPv4, even at this late stage of the lingering process of IPv4 address space exhaustion. Under such a circumstance the current Internet architecture actually includes two parallel protocol stack versions, the so-called dual stacks of TCP/IPv4 and TCP/IPv6, that correspond to the two L3 addressing schemes, namely, IPv4 (RFC 791) and IPv6 (RFC 8200) respectively. TCP/IPv6, the newer version, has been actively developed and deployed, and after nearly three decades of persistent promotion, made significant progress in adoption and widely deployed [19-24, 41-42]. The other version, TCP/IPv4, is generally regarded to be completed and essentially ossified in that future developed standards are no more required to be IPv4 compatible, as recommended by the Internet Architecture Board [18].

In retrospect, it took the networking community (too) many years to understand and accept a simple but significant fact that IPv6 is not the "next-generation IPv4", namely, it is not the successor to upgrade and evolve the IPv4 Internet to the next version; Instead, it is a new, incompatible internetworking protocol stack that is parallel to IPv4, even though the IPv6 design principles and goals of addressing, routing and application interfacing have become more and more consistent with that of IPv4 after constant revision. This fact means that IPv6 cannot smoothly upgrade and supplant IPv4 within IPv4 environments, but rather it needs to develop a new protocol stack that mirrors the IPv4 architecture, where each layer (from L2 to L5) has almost identical functionalities but is not interchangeable with the corresponding version in the other stack, e.g., each layer is scattered with incompatible IP addresses. The fact also determines that the two protocol stacks are inherently not compatible and substitutable, no matter what v4/v6 transition or interworking mechanisms are developed. Therefore, from the beginning the goal of the IPv6 design and deployment efforts is to roll out a new Internet protocol stack, instead of incrementally upgrade existing IPv4 technological ecology. This seems to be an idealistic goal, and hence is a complicated, difficult and time-consuming process that imposes huge costs [39-40, 47].

From a point of view of technology competition, a full success of IPv6 means the shutdown and exit of all the existing IPv4 networks, applications and business systems. However, decades of experience prove that legacy IPv4 technologies and asserts will not automatically exit and disappear. On the contrary, they will exist indefinitely or perhaps forever. It is worth noting that the attitude to wait and see the "natural elimination and extinction of IPv4" to drive everything of the Internet from IPv4 to move to IPv6 is problematic, if not mistaken and irresponsible. IPv4 is and will be an active technological ecosystem, which deserves respect and support, and hence to give up or suppress IPv4 technology research and innovation would be a backward and irresponsible attitude towards the networking community.

In this paper, we attempt to develop a long term evolution scheme for the TCP/IPv4 Internet based on a variable length L3 addressing scheme. Technically, our work is motivated by three basic observations, namely, **1**, it is necessary and important to maintain IPv4 compatibility; **2**, IPv4 will coexist with IPv6 and need an independent evolution approach; and **3**, IPv4 does have significant capacity for compatible extensions.

Based on such considerations, our intention to develop a new L3 addressing scheme for TCP/IPv4 evolution includes three folds, namely, **1**, to preserve the IPv4 addressing and routing architecture; **2**, to preserve the IPv4 "narrow waist" positioning; and **3**, to preserve the IPv4 socket architecture and its applications. Under these preconditions, the resulting scheme provides a feasible evolution approach for the IPv4 architecture, as well as for maintaining the legacy IPv4 networks and systems that would not have the possibility or investment to be upgraded to IPv6, allowing for a choice for those who cannot or do not need to transit everything to IPv6, or in case that existing IPv4 facilities cannot be replaced by or do not have IPv6 versions.

In summary of the contribution of this work, we specifically introduce three key points for a new addressing and routing scheme: **1**, the notion of a variable length L3 address space for extending the small IPv4 address space in a

ISBN 978-3-903176-48-5

multilevel gradual extension fashion; **2**, the idea of a single-stack implementation and transition to maintain the IPv4 "narrow waist" architecture; and **3**, the concept of a strong bidirectional compatibility constraint implied by retaining the IPv4 socket API. They are the three support points for this scheme to be a deployable and realistic, serious solution of long term evolvability.

The novelty of this work is based on the highest degree of compatibility and coexistence with existing IPv4 devices and applications so far as we know of, and hence its maximized potential for realistic deployment and large-scale application (instead of yet another "clean room paper design"). In hindsight, however, the general idea and the train of reasoning in this work may be regarded to follow the traditional discussions on the design philosophy of the Internet protocols [1], the end to end arguments [2,3], as well as design ideas for tussles, unpredictability, and changes at the edge and top layers [4]. From the perspective of research paradigm, this work may also be regarded as yet another footnote to the "clean-slate versus evolutionary research" conversation [5].

For the convenience of discussion, in the following we will use the code name 'IPswen' to denote this scheme. The outline of the rest of the paper is as follows. Section II discusses related work and brief comparison. Section III discusses the design goals, constraints, and guiding principles. The details of addressing and routing of this scheme are described in section IV and V respectively. Section VI discusses transition issues. Section VII describes implementation elements and evaluation results. Section VIII is a conclusion.

II. Related Work

It is well known that IPv4 has evolved through a series of addressing extensions, and the options mechanism is the major extensibility built in the protocol. There is a rich set of related studies in the literature. Only a few that are closely related are listed as follows.

A. IPv4 Option Extensions

Early IPng Proposals: AEIOU [25] seems to be the first proposal to suggest a scheme for providing bigger addresses without breaking the current IPv4 routing and addressing architecture, using a simple address extension mechanism with two new IPv4 options "Source Address Extension" (SAE) and "Destination Address Extension" (DAE). To be in favor of the IPng effort, the proposal was recommended by the 29th IETF meeting to be revered, withdrawn, and unimplemented [26], and hence no further study of implementation, compatibility degree or protocol stack support is available.

EIP (RFC 1385) proposed an EIP option for a larger address structure of {network ID number, host ID number}. A fixed length addressing was adopted in the draft scheme, while socket address and sockets API extensions were not explained. IPv7 (RFC 1475) uses a 10-byte option ADDEXT to carry the extended address bytes. It is a fixed length (64-bit) addressing scheme. Both were deprecated by RFC 6814. In addition, the early design drafts of *TCP Version 3* [10] and *IP Version 2* [11] described an idea of using variable length network addresses (of the "class A" fashion), which was (of course) not IPv4 (or IPv4 socket) compatible.

Locator/Identifier Separation Schemes: Some earlier work provided experimental proposals for exploring various approaches to evolving the Internet Architecture, including ILNP (Identifier-Locator Network, RFC 6227, 6740), LISP, NIRA, and hIPv4 or "hierarchical IPv4" (RFC 6036), which also suggested to encode extended addresses in some options.

B. NAT44 Extensions

NAT44 Traversal Schemes: IPv4+4 [27] and Enhanced IP [28] uses a similar 64-bit address structure {*RouterPublicIP, NodePrivateIP*}. Easy IPv4 [29] is similar to the Enhanced IP NAT traversal design, but uses a 64-bit EzIP address structure defined as {*RouterPublicIP, ClassEAddress*}. IPNL [30] inserts a new header after the IPv4 header for routing among the upgraded NAT routers, using another kind of fixed length (10 bytes) address structure {*PublicIP, NAT-Realm-ID, PrivateIP*}.

C. IoT and Related Addressing

Variable-length addresses are often proposed to support efficient packet processing in tailored or low-power devices such as IoT communications. For example, New IP [43] is Huawei's proposal for a new IP-like architecture to connect heterogenous networks and systems (e.g., via 5G data links), where variable-length addresses are used to reduce costs for interconnecting nearby (local area) devices. New IP has nothing to do for supporting or extending the existing IP (v4 or v6) architecture, nor is it compatible with the two protocol stacks. Flexible Address System (FAS) [44] provides a theoretical framework for analyzing trie structure optimization when it is extended to use variable addresses to replace both IPv4 and IPv6. Study of the necessary issues of protocol stack change and compatibility were provided. FlexIP Addressing [45] proposed a draft for using an unbounded address space for extending an early and obsoleted IPv4 extension scheme PIP (RFC 1621).

In summary, there is an essential difference between all these related proposals and IPswen, which is related to the IPswen's three design constraints on introducing variable length L3 addresses and routing. As far as we know, IPswen is presently the only scheme that maintains full compatibility (in binary code) with the IPv4 socket interface and its numerous existing applications, preserving the highest degree of coexistence and interworking so far. Without such a degree of compatibility constraints, IPv4 hardware and sockets applications could hardly be retained without modification or replacement for continued use.

III. Design Goals

As the purpose or "design philosophy" of our approach is to only modify (or "patch") the 32-bit addressing part of IPv4, namely, "IPv4 with only more compatible addresses", there are three design goals as the constraints based on the above three basic observations of IPv4.

Fundamental Goal: The top-level goal is to preserve the existing IPv4 addressing and routing scheme, including its block-based address allocation and prefix routing architecture, which is essential to maintain the existing IPv4 Internet.

Second Goal: A second goal is to make a minimal impact to the stability of the whole TCP/IPv4 architecture, and preserve the firmly established IPv4 "narrow waist" [6,7]. This will restrict our design from using methods that introduce any new (L2.5 or L3.5) sublayers or new L3 packet herders.

Third Goal: The third goal for this scheme is to preserve the support to existing IPv4 socket API and its underlying "L4.5" architecture. This is to allow for existing applications using IPv4 sockets to continue running in binary code without

modification while introducing a larger address space into the underlying IPv4 protocol stack. The socket API is not only the most important programming interface of networking applications, but also the de facto standard of implementing the TCP/IP protocol stack in operating system kernels.

To minimize the anticipated costs of upgrade and transition, we further aim to achieve the compatibility and interworking degrees to the maximum extent, namely, to allow new applications using both IPv4 and IPswen addresses to run on both the old and the new systems that use the two different version addresses. This would be a quite stronger compatibility goal than that of the general requirements on coexistence and interaction of the early IPng White Paper (RFC 1671). We refer to this goal as *bidirectional compatibility and interworking*, which is designed using a specific technique that "overlaps" the underlying socket structures of different address types as described later.

In summary, a scheme to extend IPv4 while preserving its existing addressing and routing must satisfy 3 requirements: **1**, it cannot use fixed length addresses, meaning that its using variable length addresses is inevitable; **2**, it cannot alter the IPv4 packet format, nor introduce any new interpretation to the L3 and L2 (hardware) packet structures; and **3**, it cannot modify the underlying IPv4 socket structures, excluding the possibility of any dual-stack implementation mode. Any violation of these requirements means the loss of full backward compatibility and hence the goal of an evolutionary approach and a realistic solution.

IV. Variable Length Network Addressing

The prefix based block allocation and routing of IPv4 should be kept in the same manner, no matter how large the new address space can be. Variable length encoding as adopted by this scheme is an idea that may root back to (at least) two preeminent code extension examples in the computer industry: one is the UTF-8 charset encoding (RFC 3629), which preserves the entire 7-bit ASCII charset as the shortest code plane; The other one is the variable length x86-64 instruction set, which retains full binary code backward compatibility to the legacy 32-bit x86 architecture via register extension prefix bytes (*REX* = {0100 *xxxx*}). Variable length encoding has the obvious inefficiency in storage and processing, but in fact there is always necessary (and soon negligible) sacrifice to preserve the old and shorter codes.

A. Multiple Level Subspaces Design

As a first constraint, we need to design the new addressing to be able to interwork with the 32-bit IPv4 addresses. Then the new addresses should have a format of multilevel variable length (at least 2 levels: 32-bit and longer addresses). Among various possible variable data structures, the simplest address format adopted by IPswen is a byte aligned address structure for grading extensions, where new bits of the extended address field are appended to the least significant bit of the IPv4 "base address", one byte (an octet) a time, called an "extension level". Such a concept is illustrated in figure 1.

As we discuss later, there must be an upper limit for such grading extensions. Actually, the IPv4 base address can be appended by at most 7 bytes, resulting in the longest possible subspace of 11-byte (88-bit) addresses, which is explained in the following discussion on packet header options encoding and socket address extensions. The resulting new address space to offer is a collection of these 8 subspaces of 8 address lengths. It might be not that impressive compared to the huge IPv6 address space, but as IPv4 has demonstrated, a 88-bit address space seems to be able to last for quite a while, especially when it is combined with new NAT mechanisms.

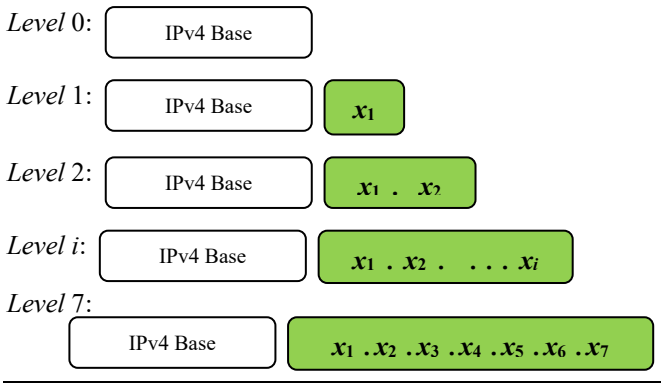


Fig. 1. Conceptual design of the IPswen variable length address space

Since IPswen addresses are in variable length, the extension level (the number of extended address bytes) must be explicitly specified, with such a general text representation:

$$\texttt{a.b.c.d}\ (\pm i)\,x_1.x_2.\ \ldots\ .x_i\ ,\quad \text{for } i = 0, 1..7.$$

where addresses with negative levels are used for designing some new stateless NAT mechanisms (discussed later). Here are a few examples of the address representation:

```
const char *a = "1.2.3.4(0)", *b = "1.2.3.4(1)5",
     *addr0_level4 = "0.0.0.0(4)0:4-1",
     *mask_L5 = "255.255.255.255(-5)255:5-1",
     *prefix80_L7 = "8.8.8.8(7)8:7-1/80";
```

where the index range expressions *ddd:i-j* mean that the bytes from *i* to *j* are set to value *ddd*. Since the maximal size is predetermined in design, a fixed length storage representation of addresses of all levels may have a memory layout (in big-endian byte order) that looks like as follows, which is defined as a core API data structure *ipswen_addr*{}.

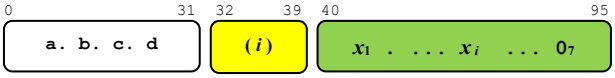


Fig. 2. Variable length address storage

For fixed length addressing, the (binary) storage format of the following three representations are the same: addresses in the packet header (including source and destination), addresses used by sockets (API) applications, and the addresses carried in the (routing) protocol messages. That is no more the case for these variable length addresses. Also, as we will see later, the level 0 addresses are no more exactly the IPv4 addresses, for the later does not have an "extension level" (*i*) field, and hence use different storage and packet encoding representation. The use of the variable length addresses at routers and hosts is almost the same as the fixed IPv4 or IPv6 addresses, except for two points: **1**, the ID of a network, represented by an address block (namely, a prefix), now is a variable length prefix of level 0 to 7; And **2**, the longest prefix match algorithms are extended to deal with variable length prefixes. Practically, the major concern with a variable length network addressing would be the implications on the router performance, as it is essential to maintain line rate speed to process IP packets on the core networks of the Internet. We discuss this hardware acceleration issue later in section VI.

B. Single Stack Based Extension

The IPv4 narrow waist compatibility constraint requires that IPswen must be a single stack implementation. Such a requirement is met with the IPv4 options field to encode the address extension, supporting a single stack of packet I/O.

Thus, to encode the new addresses in the IPv4 packet header, a few new options are defined for embedding a pair of variable length source and destination addresses. One of the two unused option type codes (10 and 11 in binary) may now be used. Here we choose the option type 11, and hence the new type byte takes the bit pattern of $\{111f_T\ f_Sxxx\}$, where the bit f_T indicates a source ($f_T = 0$) or a destination ($f_T = 1$) address extension; the lowest three bits *xxx* denote the binary value (0~7) of the address space level, and the f_S bit is the sign flag of the address level ($f_S = 0$ for a positive level and $f_S = 1$ for a negative level). For example, a new option of type byte 0xE0 indicates the extension bytes of the source address field, and option type 0xF8 indicates the extension of the destination address with a negative level. In the present version, the option length is required to set to the level value plus 2.

C. Socket Interface Extension

It is, however, not sufficient to support full IPv4 backward compatibility if only to preserve the IPv4 packet header format and introduce address extensions in the options field. Since the multilevel addressing of IPswen is required to maintain binary backward compatibility to the IPv4 sockets and its API, we need further to preserve the IPv4 sockets architecture, which is based on the IPv4 socket address structure as follows.

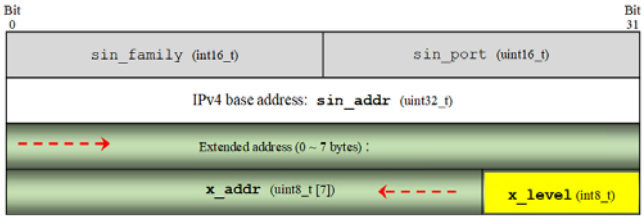


Fig. 3. Extended *AF_INET* socket address structure

IPswen must be "in-place" implemented altogether with IPv4 in any TCP/IP protocol stack. Since it must preserve the source code compilation of existing IPv4 applications, the extended socket address structure must be a union structure, which merges the new (fixed length) address *ipswen_addr*{} with the IPv4 address *in_addr*{}. Therefore, the largest address extension level of IPswen is up to level 7, where an extension flag byte denotes the extension byte array length and the rest 7 bytes are reserved for the extended address field.

The socket address structure extended as such provides two parallel sets of old and new IP addresses without interference to each other, so as to make the least intrusion to existing code, where code using the original address structure *in_addr*{} and its API is not affected, though the same *sockaddr_in*{} is interpreted differently by the extended socket API. In this way, the same IPv4 source code can compile and run on both new and old systems with equivalent object code. Such a parallel compilation technique proves to play a critical role in "in-place" single stack implementation.

D. Extended Protocol Stack

Necessary extensions to support the IPswen addressing within the TCP/IPv4 stack tend to be rather simple and straightforward. **Firstly**, extensions to ARP and RARP are straightforward, where adding the support of the above defined structure *ipswen_addr*{} into their protocol messages is simple for the general nature of their L2 and L3 address fields. Extension to ICMP and IGMP is also as simple, since IPswen (so far) does not extend the multicast addressing scheme of IPv4. **Secondly**, TCP and UDP need no modification when a simple rule is introduced: Just ignore any address extension information carried in the options field of any IPv4 packet. Abiding by the basic principle of layering design, it is good for an L4 protocol not to have any knowledge of the "optional L3 information" that is provided in the options field of the L3 header. Meanwhile, transport layer flows of {*src, dst*}{*L3_addr, L4_port*} will be properly handled by the unified socket operations. **Lastly**, L5 messaging protocols that are not directly related to IPv4 addresses are not affected and hence need little upgrading on IPswen supported OS kernels and C runtime libraries, which is a large category including HTTP, SMTP, POP3, LDAP, RPC, NFS, etc., together with their related software packages. Extensions to address related L5 protocols include DNS, DHCP, and routing message protocols, which are briefly discussed in the next section.

V. Variable Length Address Routing

The 8 subspaces of the IPswen address space are independent in terms of address allocation, packet forwarding and routing. For each of the 8 address subspaces (level 0 to level 7), every address level has an independent network ID numbering space that uses its own prefix and mask structures. So there shall be 8 levels of address prefixes. The same bit sequences of different address levels represent independent network IDs in different address subspaces. Hence every prefix or mask (as a special case of address) needs to explicitly specify its level. IPswen nodes on the same subnetwork (with direct L2 and L1 links) can establish direct L3 connections (packet delivery) only when they are assigned with address prefixes (NetIDs) of the same length and the same extension level. This section briefly discusses necessary extensions to the IPv4 address and routing protocols. (Detailed analysis and improvements will be addressed in follow-up work and reported elsewhere.)

A. Routing table extensions

Firstly, the CIDR notation of IPv4 prefixes is extended straightforwardly to the 8 address levels as follows:

{NetID Prefix of Level *i*, HostID = 0} / PrefixLength *n*

where level $i = 0..7$ and PrefixLength $n = 0..88$. Here is a 33-bit prefix in the 40-bit level one subspace: 1.2.3.4(1)128/33.

To improve the efficiency of the longest match of the 8 level prefixes of IPswen addresses, a PCL-trie (namely, path and level compressed trie) [35-38] based routing table lookup design [46] is used, where each node of the PLC-trie includes three fields: {*.length*, *.skip*, *.branch*}, which is used to extend the trie path and level compression. The complexity in space and time is essentially the same as the conventional PLC-tries, for the level compression enhanced lookup performance remains, and the trie height is insensitive to the key lengths, but only dependent on the number and distribution density of the indexed keywords.

B. Routing protocol extensions

When applying the abstract CIDR prefix notation to the actual routing protocols, we need a triplet <*level, prefix, length*> to encode and store a variable length prefix, instead of the usual tuple <*prefix*, *length*>. Here is the working code definition of such a prefix:

```
/* VL prefix <level, prefix, length> e.g., 1.1.1.1(7)7:1/80 */
  struct ipswen prefix  {
         uint8 t  level mark;
         int8 t   length;
         uint8_t  prefix[0];
  };
```

Most of the routing algorithms, routing information tables and protocol messages that deal with IPv4 address prefixes are general enough to adopt the new triplet prefix format. For example, the extensions to the three popular IPv4 unicast routing protocols are very straightforward, including RIPv2 (RFC 2453), OSPFv2 (RFC 2328) and BGP4 (RFC 4271). Their extensions are briefly explained as follows.

RIPv2 Extensions: The Metric field of RIPv2 RTEs is decomposed to 4 bytes, where only the fourth is used for the original Metric values (1~16), and the higher three bytes are now used to denote the IPswen extension levels of the three IPv4 base addresses and network masks of the present RTE, namely {*route network address, subnet mask, next-hop address*}. Then all the base addresses with none 0 levels are appended to the end of the RTE array, with a leading separator flag field 0xFFFFFFFF.

OSPFv2 Extensions: The sender routers first append a separator flag field 0xFFFFFFFF to the message body (of fixed or variable length), followed by the extended prefix list of partial address *ipswen_vext*{} records. The receiver routers then extract the records and restore their complete storage records in the fixed length structure *ipswen_addr*{}.

BGP4+ Extensions: It reinterprets the conventional tuple representation <*prefixLengthByte*, *addressPrefixBytes*> of the compressed network mask and address as follows: If the prefix length byte is less than 0xF0 (240), then it is a fixed length prefix (an IPv4 or IPv6 prefix according to the message context); If it is larger than 0xF0 (240), since there are no protocol families that currently use addresses longer than 240 bits, this byte is interpreted to an IPswen prefix as defined by above. For extending the BGP4+ messages, a combination of <*AFI* = 1, *SAFI* = 127> is used to denote the variable address prefixes, which can be used in the address and prefix tuple fields of the multiprotocol extended attributes (*MP_REACH_NLRI* and *MP_UNREACH_NLRI*).

C. Addressing related application protocols

The two important L5 protocols that are directly related to IPswen extension include DNS and DHCP(v4). Extensions to other high-level applications (VPN, SSH, etc.) are similar.

Extended DNS: We only need to relax the length and text representation syntax of type A and type PTR resource records (RR), instead of adding any new RR types. For example,

```
v4.x.y.z   A  IN 124.193.127.100          # IPv4 RR
www.x.y.z  A  IN  124.193.127.101(0)      # New
mail.x.y.z A  IN  124.193.127.102(3)3.2.1 # New
```

The unified 6-field binary exchange format of the corresponding RR (in both local process memory and DNS message body) are extended as the following RR examples:

```
{ Name(var.len)\0|Type(2)|Class(2)|TTL(4)|RDLENGTH(2)|RDATA    }
 [2]v4[1]x[1]y[1]z[0]   A  IN       600    4   124.193.127.100
 [3]www[1]x[1]y[1]z[0]  A  IN    3600    5  124.193.127.101(0)
 [4]mail[1]x[1]y[1]z[0] A  IN 3600  8  124.193.127.102(3)3.2.1
```

Extended DHCP: DHCP may be regarded as a role model of protocol design with inherent backward compatibility. For IPswen addressing, it only needs to add a new DHCP option with TagCode = 127 (or any other unused tag numbers), which is used to carry all the information for IPswen extension. If such an option is present, then all the address fields with length > 4 (including all the subnet mask addresses) in the same message are inferenced according to the partial IPswen address structure *ipswen_vext*{}.

```
{ Tag: 127; Length: n bytes; Value (variable):
   [xL, xAB]CA [xL, xAB]YA [xL, xAB]SA [xL, xAB]RA }
```

Such an extension scheme does not need to modify any bit of the DHCP packet format.

VI. Transition

Minimizing the transition and deployment costs is the core requirement for this addressing scheme, which is the essence of designing an evolutionary approach to existing technology.

A. Are IPv4 Options an Option?

Conventional wisdom suggests that IPv4 options are limited in usability and not an option for practical capability [33], and IPv4 packets with options tend to be blocked (dropped) or processed on the "slow path", typically by edge routers, for security and hardware performance concerns according to some recommended best practice (RFC 7126).

There are, however, two notable trends that have had significant implications on the current Internet. One is the "flattening" trend, where large cloud services providers are minimizing the average distances and "depths" of the end-users. The other trend is that network layer connectivity has been increasingly directed by application service systems, changing the end-to-end model to be an end-server-end paradigm, including the client/server application architecture and cloud-based services, where the network L3 addresses are no more deemed as a universal ID for applications, but rather application accounts play a key role in connection processing. With such new trends, the situation of IPv4 options usability may gradually change. For example, new best practices on more and more of the core networks already allow for the general acceptance of IP options [34], with configuration flexibilities for specific options to the fast path. Of course, for the old core routers to transition, hardware acceleration technology specifically designed for the variable length address processing is necessary

B. Hardware Acceleration of IPv4 Options

Differently to the end host systems or software routers, packets containing options are typically queued on the "slow path" to the CPU for processing, instead of the "fast path" to line cards to be processed at line rate. It is generally difficult to parse various IPv4 options at line rate. However, hardware acceleration for a specifically selected option is already possible. And IPswen addressing options can be processed at the hardware line rate when appropriate design constraints are further defined. As a first case of study, a simple constraint for implementing line rate parsing of IPswen options is that such options must be the first one or (at most) two options when present, such that it is always at a fixed position in the options field of the packet header. Under such constraint, a fixed 36-byte size lookup of the packet header range will be sufficient for a hardware processing design, where the possible range of packet header sizes, or the *.IHL* filed values, is from 6 to 9. One lookup for the extended address information would then be sufficient.

C. NATswen Based Transition

NAT has become an inherent part of existing IPv4 deployment as well as translation based IPv6 transition. Today any mainstream application must adapt itself to typical NAT44 or NAT64 environments as the Internet becomes

increasingly "flatten" with the pervasive C/S architecture and cloud-based services with middleware nodes (NATs, firewalls, gateways, proxy or internal routers). We here let the term *NATswen* denote the general form of the NAT44 mechanism and NATs (NAT routers) that are extended with IPswen addressing capability. NATswen has 3 derivative configurations that are configurated for 3 specific scenarios.

NATs/s: a direct extension of IPv4 NAT and NAPT, which maps IPswen addresses with positive extension levels (+*i*) between two address subsets. When the source and destination addresses are IPv4-only, NATs/s works the same way as NAT44.

NATs/4: mapping positive-level internal (private) IPswen addresses to negative-level public IPswen addresses based on a special internal network address configuration, where the base address field of all the IPswen nodes on an internal network be configured to be the same IPv4 address, which is further set to a public IPv4 address of the NATs/4 router. The negative level addresses of IPswen are in the "semi-public subspaces", which are used to route the packets only by their base address field. For a negative level address, e.g., 124.193.127.99(-3)3.2.1, the base address field 124.193.127.99 will be used for global routing, while the extended address part "3.2.1" denotes a private extension address only effective within the internal NAT network, like a "telephone extension number".

NAT4/s: a mechanism for interworking legacy IPv4-only nodes in edge networks with external IPswen networks, for example, it allows IPv4-only web browsers to access contents hosted on server nodes using arbitrary IPswen addresses.

NAT4/s works in a similar way of IPv6 stateful NAT64 and is simpler in packet translation and state mapping. NAT4/s needs an extended DNS server, DNS4/s for mapping an IPswen server address to a dedicated (temporary) IPv4 address of the client-end. It is simpler due to that all IP nodes involved are single-stacked (for IPswen is always an in-situ rewrite/replacement of the corresponding IPv4 stack). A key difference from NAT64 is that NAT4/s requires that all the DNS requests of the internal hosts be operated by the DNS4/s node, instead of processing DNS requests independently by each client-end node. It means the internal hosts should configure their default DNS server to be the DNS4/s node.

To connect an IPv4-only host, say node A with 10.10.10.10, to an IPswen server, say node S with 1.2.3.4(5)6.7.8.9.10, NAT4/s first delegates the DNS record of S for A, and then allocates a temporary IPv4-only address *addr_A* (e.g., a private IPv4 address) and send back the DNS response with *addr_A* being the DNS RR of the node S. From this point, all traffic from (and to) the node A to (and from) the node S will be routed with the delegate *addr_A* for the node S during the session of the connection.

NAT4/s also allows an IPswen client-end (say node B with 10.10.10.10(1)10) to connect to an IPv4-only server (S2: 1.2.3.4) without relying on the DNS4/s node, similar to the case of NAT64. NAT4/s allows all the existing IPv4 networks to directly interwork with outside IPswen networks without modification, which means that existing IPv4 networks and address allocations can be preserved permanently. It also means that IPswen can be deployed only with new devices and systems to support an evolutionary upgrade of the Internet.

All the three derivative forms of NATswen are very lightweight L3-only address translation, applicable to any L4 protocols and L5 message applications that is directly based on IPv4, with much better performance than NAT44 for TCP, UDP and ICMP messaging. Namely, NATswen routers as a category of (L3 only) stateless translation devices could be expected to be much faster than NAPT44 routers. Due to such simplicity of this mechanism, high performance, low cost and carrier grade NATswen routers can be expectable at lower prices. In summary, the NATswen mechanism is likely the most promising approach that we can devise for a gradual, smooth, "pointwise" and cost effective IPswen transition.

D. 4s4XLAT Transition

Similar to the 464XLAT transition mechanism of IPv6, IPswen can be deployed using a 4s4XLAT mechanism for robust and cost efficient transition of legacy devices and IPv4-ony applications without DNS4/s support. To this end, 4s4XLAT requires that the core networks are upgraded to IPswen first, and then let the legacy applications directly interwork with a client-end module that implements the above NATswen capability for IPv4 address translation. It allows that only the ISP networks are necessary to upgrade to support interworking between IPv4-only and IPswen nodes, without modifying the CPE devices on the user end. 4s4XLAT is much simpler and faster than 464XLAT due to the two factors: IPswen/IPv4 packets translation is trivial, and address translation between IPv4 and IPswen is a simple one, where each IPv4 address is mapped to a level 0 IPswen address, and an IPswen address to a dedicated (e.g., private) IPv4 address.

In addition, stateless IPv4/IPswen translation mechanisms similar to the IPv6 counterparts (SIIT, IVI, etc.) are also available. Overall, they can work more efficient and reliable, thanks to the strict constraint of single-stack design. For level 0 addresses, the interworking is direct and bidirectional; for nonzero level addresses, direct connections can be established with the help of NAT based transition mechanisms.

VII. Implementation and Evaluation

We now discuss what are necessary for an implementation of the IPswen scheme, and evaluation of the related elements.

A. Three Implementation Elements

By design, there are three critical elements that are needed to implement the variable length address scheme, namely, addressing implementation, single stack implementation, and socket overlapping implementation, where the extended IPv4 socket address arguments are employed to carry and pass the new addresses, as illustrated by figure 4.

Within the protocol stack, each socket type is implemented as a jump table from L5 operations to L4 operations: for each triple combination {**F**: address family, **T**: socket type, **P**: L4 protocol}, there should be a corresponding (L5) API *socket*(**F**, **T**, **P**) instance object created by the kernel, which is implemented with a jump table for switching the L5 ops to L4 ops. Therefore, for each L4 protocol of each socket type of each protocol/address family *AF_xxx*, a mapping pair of operations sets <*L5_ops*{}, *L4_ops*{}> should be defined first, which is essentially the kernel representation of the *socket*{} structure. One particular technique issue is how to pass the address arguments from and to the other modules. It proves that the design of reusing the IPv4 socket address structure *sockaddr_in*{} and preserving parallel compilation support can be extremely helpful for reducing the huge amount of engineering workloads associated with address arguments refactoring. Binary code level compatibility to the upper layer (user space) sockets API also greatly improves the design logic.

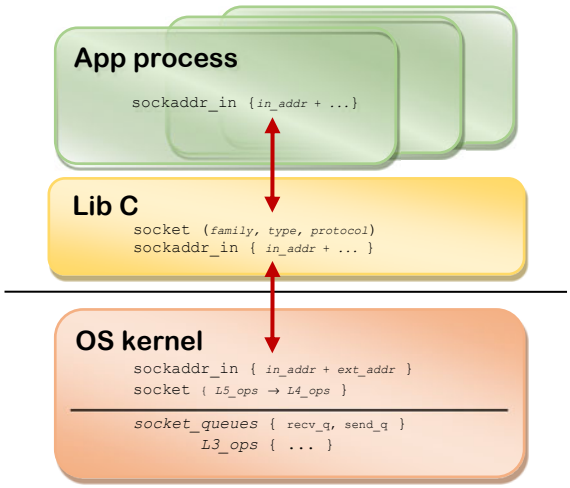


Fig. 4. *AF_INET* socket address argument passing

B. A 4-Layer Reference System Configuration

We use the Linux kernel code repository to implement an IPswen node system for experiment. A reference system configuration includes 4 layers:

L4, Network infrastructure applications layer:
{Quagga/FRR zebra, named, dhcpd} extended;
L3, Network utilities layer:
{net-tools, iproute2, iputils} extended;
L2, CRT library layer:
IPv4 Sockets API {with GNU libc.so} extended;
L1, Kernel protocol stack layer:
{IPv4, ARP/RARP, ICMP, TCPv4/UDPv4} extended.

IPswen support of such a system configuration is software only upgrading, and each of the layers and individual components can be upgraded independently. The "in-place" code revision and parallel compilation methods provided by the socket address extensions are applicable the same way to the kernel stack, runtime library, and applications code.

The default Linux 5.4.0 kernel configuration of the Ubuntu 20.04 LTS Desktop x86-64 (*/kernel/.configure-generic*) was used for the OS part. Most of the substantial code revision relates to data structures and routines that use a few IPv4 address arguments of the AF_INET family, including the data types *socaddr_in*{}, *if_addr*{}, *in_addr*{} and *__be32*. The only nontrivial part is the extension of the forwarding information base (FIB) and routing tables handling.

C. Evaluation

Using the above 4-layer node system configuration, the evaluation included incremental compatibility tests of socket API software, interworking on existing typical networks, router software performance, along with major issues noticed.

Compatibility: First, socket overlapping compatibility was evaluated. The test cases consisted of all the arbitrary combinations of IPv4 and their extended versions as follows,

$$\binom{IPv4}{IPswen} \times \begin{pmatrix} Socket\ Apps \\ libc.so\ API \\ Linux\ kernel \end{pmatrix}$$

Results indicated that the constraints of mixed compatibility were ideally fulfilled. The compatibility of IPv4 {apps + libc} over the extended IPv4 Linux kernel is surprisingly good, where all the default Ubuntu networking applications worked the same way on the new Linux kernel, which uses the 12-byte address structure (Fig. 2) to handle all of their IPv4 addresses as level 0 addresses.

Interworking: The software router on the node system interworked directly using level 0 addresses on existing IPv4-only networks, with packet I/O between the nodes of all the 4 combinations:

$$\binom{IPv4}{IPswen} nodes \times \binom{IPv4}{IPwen} networks_{\{Sw.\ \&\ Routers\}}$$

The extended DNS, DHCP and routing information message protocols work with level 0 addresses, and interwork with their old versions with IPv4-only configurations.

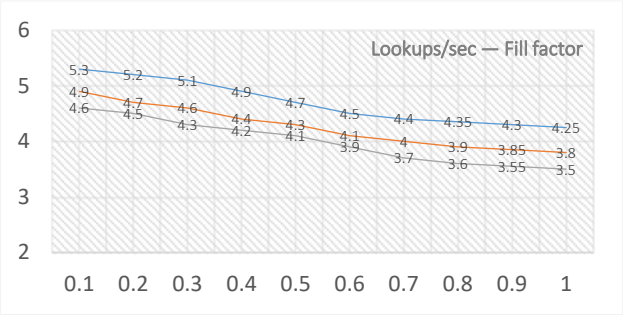
Performance: To the router software that was used for testing, the extra processing overhead of packets with options field seemed insignificant compared to FIB trie lookups. Overall, no perceived performance deterioration beyond system and network fluctuations. The average FIB lookup speeds were: 12.0 million per second for IPv4 prefixes, 8.4 million per second for IPswen prefixes with random level (0~7) distributions. To estimate the resulting performance of the extended PLC-trie lookup, average lookup depths and speed of lookups per second are tested with respect to that of IPv4 addresses processing, which is shown in the following table.

TABLE I. COMPARISON OF 32-BIT AND VARIABLE-LENGTH LOOKUPS

Data set	Route entries	32-bit depth/speed	Variable-length depth/speed
Mae East	38,367	1.66 /12.0M	2.03 / 8.4M
Mae West	15,022	1.29 / 11.7M	2.43 / 6.3M
AADS	20,299	1.42 / 12.6M	1.61 6.8M

The test data sets were from the original static LC-trie experiments [35-38], which were typical router prefix distributions. For the worst-case random distributions of IPswen prefixes, the PLC-trie was very stable in terms of trie depth expansion. Lookup performance dropped relatively more obvious due to trie nodes expansion, but it was quite comparable and sufficiently fast for software-only routers.

To further understand the configuration variances of the extended trie, its fill factor threshold F_t was further investigated. The actual performance is different related to the level compression effects, which heavily relies on the fill factor threshold F_t, namely, larger value F_t leads to less compression of levels; When F_t is set to 100%, only complete binary subtrie can be level compressed. Along with the 10 points of the fill factor threshold F_t, 3 comparisons are tested, using 3 prefixes data sets of diversified distributions generated with a recent IPv4 BGP table, which embeds, randomize and simulates the IPv4 prefix distribution respectively (code names *Embed*/**blue**, *Random*/**yellow**, *BGP-Distr*/**gray**). The results are illustrated as follows, where lookup speed is in 10 million operations, search depths are in 1 bit, memory in KB.



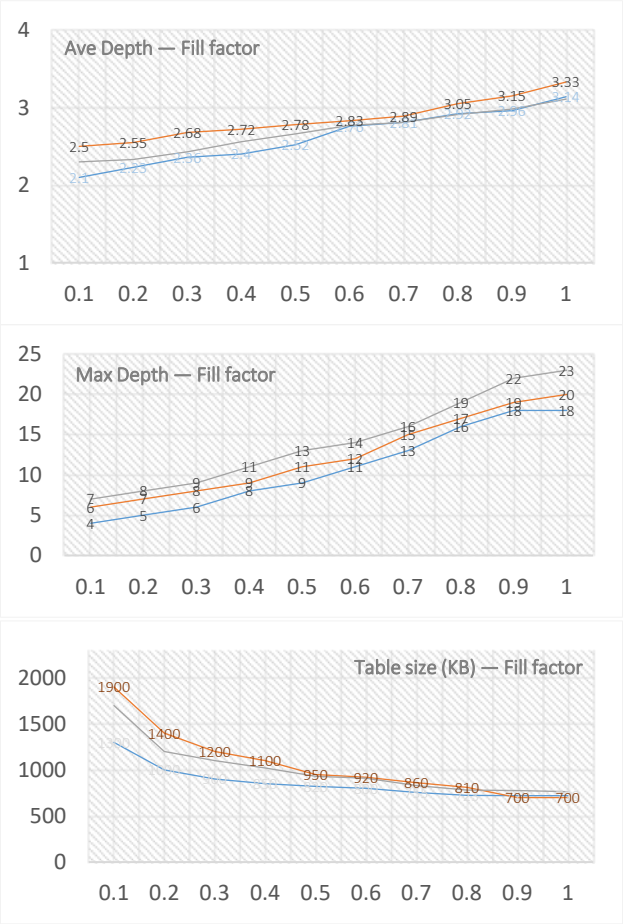


Fig. 5. PLC-trie performance vs. Fill factor threshold F_t

The results indicate that software-only single-thread lookup speed can be over 40 million lookups per second on ordinary PCs, with an average trie search path depth of about 3 nodes. Such a design is suitable for efficient and generic routing table implementation for IPswen and other similar addressing schemes. Lookup speed and memory overhead can be balanced when the fill factor values are adjusted accordingly. The results may also help to develop hardware acceleration techniques to deal with IPswen prefixes, which is a topic for investigation in future work.

D. Limitations to Overcome

There will certainly be considerable limitations if this new addressing is to be deployed at the Internet scale, despite it worked as expected in small networks and local systems with only mainstream applications. In addition to the above upgrade issues of global Internet routers, there are many IPv4 options problems and limitations associated with legacy hardware devices and software modules. A few major cases are described below.

Hardware incompatibility: A few (~1/5) SOHO and Wi-Fi routers along the tested L2 links dropped IPswen packets. Most of these devices would never upgrade until the end of the product life cycle. On the other hand, routers and devices based on embedded Linux works very well thanks to the Linux kernel that has adopted rules of RFC 1122 from the beginning.

Hardware optimization issues: IPswen options may have a direct impact on some hardware optimization techniques that are used by some devices. For example, some popular acceleration methods for NIC offloads and TSO hardware engines may need to be configurated or revised to copy all the new IPswen options upon fragmentation, or they must be disabled to run the new Linux kernel.

Software optimization issues: We also have encountered a few problematic software modules, mostly device drivers, and surprisedly found that they used a byte alignment optimization technique to packs two adjacent IPv4 address variables in a 64-bit integer. Such a technique prevents the upgraded kernel and C library from correctly getting and setting the IPv4 addresses as a field in the IPswen address storage. Such modules could not work and had to be unloaded.

It is worth noting that all such devices that cannot be upgraded to IPswen by software-only update can still be connected to IPswen networks via appropriate NAT transition mechanism such as NAT4/s and 4s4XLAT as discussed above.

E. Most Common Questions

In addition to the above hardware and software limitations, we have also received useful feedbacks, concerns, or objections to various parts of this approach while evaluating its variable length addressing scheme. We include here some typical topics for a brief discussion.

Legacy IPv4 devices can't communicate with IPswen hosts: *Web browsers on IPv4-only devices must be able to access webpages on Web servers running on IPswen, or the whole scheme will be meaningless.* —— Well, they can, only if they are configurated to use a NAT4/s router to access IPswen hosts, as described in section VI.

The Internet is too small to add a third IP protocol: *Upgrading directly to IPv6 is far better than first upgrading to IPswen and later on going another upgrade to IPv6.* —— Firstly, IPswen is not a third IP stack, but is an integrated part/extension of IPv4 per se, designed to be an in-place, single-stack and 100% compatible substitute of the IPv4 protocol stack. Secondly, a significant part of the IPv4 Internet will probably never be modified or upgraded. Initially only a tiny fraction may be upgraded to IPswen, and "once in IPswen, always in IPswen", there will no more be any need for another upgrade from IPv4-only or IPswen to IPv6, as discussed early.

The work is dismissive of IPv6: *It might bring a new breath to IPv4 but could postpone the eventual migration to IPv6.* —— As stated earlier, IPv4 and IPv6 are two parallel versions of the Internet L3 and most probably will evolve in parallel. It is not the purpose of designing this scheme to affect or slow down the progress of IPv6 adoption, particularly at this stage when IPv6 deployment is getting momentum. On the other hand, many IPv4 systems and devices may never transit to IPv6, and may upgrade to IPswen or NAT4/s instead.

The work is redundant with NATs: *The current IPv4 space is enough for maintaining legacy systems, and new systems should go IPv6, with NAT64/46 to allow communication between IPv4 and IPv6.* —— There are two parts in this question. The first part is similar to the above one: IPv4/v6 are two parallel architectures, and over the years no known 4/6 transition methods are sufficiently perfect. The second part relates to whether or not a long-term evolution is necessary for TCP/IPv4. We think that it is so, for the IPv4 Internet has not had new public addresses to offer for over a decade and needs an evolutionary solution, otherwise the *openness principle* of Internet connectivity is lost: Anyone who needs a (public) address space can get an address space.

IPv4 options considered harmful (or not an option): *The current Internet is not ready to deal with them, as core routers don't process them at line rate, drop rate at edge routers are too high, too little value to gain to upgrade routers to optimize for variable length addressing.* —— Indeed, the

current Internet is not yet ready for large-scale application of IPswen options, but the same used to be true for IPv6. The host ends and software updated routers are mostly ready though. To upgrade router hardware to put IPswen optioned packets to the fast path is simple, with about the same cost of CIDR routers upgrading. In essence, this again relates to how to keep the IPv4 "legacy": To use it or to drop it. Logically, most IPv4 networks and systems will be alive and well, demanding long term maintenance and appropriate upgrading.

VIII. Conclusion

The TCP/IPv4 architecture possesses a potential for addressing extensibility, which is inherent in its internal logic. We construct a falsification case, the variable length addressing scheme IPswen, to demonstrate that the IPv4 protocol stack and addressing architecture are not "inherently limited and ossified", but can be extended for long term evolution without the necessity to be discarded by introducing a sufficiently large L3 address space, contrary to the conventional wisdom that IPv6 is the only way to support the ever-growing demands of internetworking addresses and routing prefixes. And in defence of IPv4 evolution, our key point is that it is necessary, possible, and responsible for the Internet community to provide a feasible approach for the continued and long-term evolution of the technological ecosystem of TCP/IPv4. We argue that, after CIDR and NAPT, an IPv4 compatible variable length addressing scheme might be another viable approach for maintaining healthy and exponential growth of the IPv4 Internet, which needs its independent evolution path that is parallel to that of IPv6. More importantly, it is possible to transit to IPswen in a smooth, seamless and non-intrusive way, where all that is needed technically for the stagnating IPv4 Internet to gradually evolve into "a brave new world" is to support "a brave new IPv4 option" (the IPswen address extension option), whose cost and time span can be comparable to that of CIDR upgrading of the whole Internet in 1990s, with far less efforts and chaos than the IPv6 roll-out.

It is also worth mentioning that from a theoretical or algorithmic point of view, the solution proposed here is of neither outstanding novelty nor much technical creativity, or seems quite "hacky" with respect to the key part of IPv4 socket binary compatibility. Our understanding is that the reason for such a solution not to be of much creativity or novelty is that it does not need to be. The topic of IPv4 evolution is so "old fashioned and banal" that it is just a problem of attitude whether or not to perceive and appreciate the remarkable extensibility and adaptivity of the IPv4 architecture, and to acknowledge that its long term evolution is worth both defending and acceptance.

References

[1] D. Clark, 1988. The Design Philosophy of the DARPA Internet Protocols. In Proceedings of ACM SIGCOMM, 1988.
[2] J. Saltzer, D. Reed, D. Clark, 1984. End-to-End Arguments in System Design. ACM TOCS, Vol 2, N. 4, pp. 277-288, November 1984.
[3] M. Blumenthal, D. Clark, 2001. Rethinking the design of the Internet: The end to end arguments vs. the brave new world. ACM Transactions on Internet Technology, 1(1), pp. 70-109, August 2001. https://doi.org/10.1145/383034.383037
[4] D. Clark, J. Wroclawski, K. Sollins, R. Braden, 2002. Tussle in Cyberspace: Defining Tomorrow's Internet. In Proceedings of ACM SIGCOMM, 2002.
[5] J. Rexford, C. Dovrolis, 2010. Future Internet Architecture: Clean-Slate Versus Evolutionary Research. Communications of the ACM, Vol. 53 No. 9, pp. 36-40, September 2010.
[6] S. Akhshabi, C. Dovrolis, 2011. The Evolution of Layered Protocol Stacks Leads to an Hourglass-Shaped Architecture. In Proceedings of ACM SIGCOMM, 2011.
[7] C. Dovrolis, T. Streelman, 2010. Evolvable Network Architectures: What can we learn from Biology? ACM SIGCOMM Computer Communications Review, 40(2), 2010.
[8] A. Tanenbaum, D. Wetherall, 2010. Computer Networks, Fifth Edition. ISBN: 978-7-111-35925-8. Prentice Hall, 2010.
[9] K. Fall, W. Stevens, 2012. TCP/IP Illustrated, Volume 1: The Protocols, Second Edition. ISBN: 978-0-321-33631-6. Addison-Wesley Professional, 2012.
[10] V. Cerf, J. Postel, "TCP 3 Specification", IEN #21, January 1978. https://www.rfc-editor.org/ien/ien21.pdf
[11] J. Postel, "Draft Internetwork Protocol Specification, Version 2", IEN #28, January 1978. https://www.rfc-editor.org/ien/ien28.pdf
[12] P. Tsuchiya, T. Eng, 1993. Extending the IP internet through address reuse. ACM SIGCOMM Computer Communication Review, Vol.23 Issue 1, pp. 16-33, January 1993.
[13] Nejc Skoberne, Olaf Maennel, Iain Phillips, Randy Bush, Jan Zorz, and Mojca Ciglaric, 2014. IPv4 Address Sharing Mechanism Classification and Tradeoff Analysis. IEEE/ACM Transactions on Networking, Vol. 22, No. 2, April 2014.
[14] IPv4 Market Group, 2019. IPv4 Price Trends. http://ipv4marketgroup.com/ipv4-price-trends/
[15] B. Kuerbis, M. Mueller, 2019. Is there hope for IPv6? The Internet Governance Project, January 2019. https://www.internetgovernance.org/2019/01/04/is-there-hope-for-ipv6/
[16] B. Kuerbis, M. Mueller, 2009. The Hidden Standards War: Economic Factors Affecting IPv6 Deployment. The Internet Governance Project, February 2019.
[17] C. Marsan, 2019. Biggest mistake for IPv6: It's not backwards compatible, developers admit. NetworkWorld Reports, March 2009. https://www.networkworld.com/article/2265836/biggest-mistake-for-ipv6--it-s-not-backwards-compatible--developers-admit.html
[18] C. Morgan, 2016. IAB Statement on IPv6. Internet Architecture Board, November 2016. https://www.iab.org/2016/11/07/iab-statement-on-ipv6/
[19] The Internet Society 2021. IPv6 Statistics. https://www.internetsociety.org/deploy360/ipv6/statistics/
[20] The Internet Society 2018. State of IPv6 Deployment, 6th June 2018. https://www.internetsociety.org/globalinternetreport/
[21] APNIC 2021. IPv6 Measurement Maps. https://stats.labs.apnic.net/ipv6
[22] Why No IPv6, 2021. Top 1000 Alexa Sites Statics. http://whynoipv6.com
[23] M. Nikkhah, R. Guerin, 2016. Migrating the Internet to IPv6: An Exploration of the When and Why. IEEE/ACM Transactions on Networking, Vol.24, No.4, August 2016.
[24] J. Czyz, et al., 2014. Measuring IPv6 Adoption. In Proceedings of ACM SIGCOMM, 2014.
[25] B. Carpenter 1994. Address Extension by IP Option Usage (AEIOU). Internet Draft, March 1994. https://tools.ietf.org/html/draft-carpenter-aeiou-00.txt
[26] Peter Ford, 1994. Minutes: Brian Carpenter's Presentation on AEIOU. In Proceedings of 29th IETF meeting, Seattle, April 1994. https://www.ietf.org/proceedings/29/ipng/aeiou.html
[27] Z. Turanyi, A. Valko, 2002. IPv4+4. In Proceedings of the 10th IEEE International Conference on Network Protocols, 2002.
[28] W. Chimiak, S. Patton, 2014. Enhanced IP: IPv4 with 64 Bit Addresses. IEEE Computer, Volume 47, Issue 2, February 2014.
[29] A. Chen, et al.,2019. Adaptive IPv4 Address Space. Internet Draft, June 2019. https://tools.ietf.org/html/draft-chen-ati-adaptive-ipv4-address-space-05
[30] P. Francis, R. Gummadi, 2001. IPNL: A NAT-Extended Internet Architecture. In Proceedings of ACM SIGCOMM, 2001.
[31] U.S. GAO, 2020. INTERNET PROTOCOL VERSION 6: DOD Needs to Improve Transition Planning. June 2020. https://www.gao.gov/products/GAO-20-402
[32] Sylvia Ratnasamy, Scott Shenker, Steven McCanne. Towards an evolvable internet architecture. SIGCOMM 2005. https://doi.org/10.1145/1080091.1080128
[33] R. Fonseca, et al., "IP Options are not an option", Technical Report, CS Division, UC Berkeley, December 9, 2005.
[34] B. Goodchild, et al., "The Record Route Option is an Option!", in Proceedings of IMC '17: Internet Measurement Conference, London, United Kingdom, November 1–3, 2017.
[35] S. Nilsson and G. Karlsson, "Fast Address Look-up for Internet Routers," Proceedings of the IFIP 4th International Conference on Broadband Communications, pp. 11-22, 1998.
[36] S. Nilsson and G. Karlsson, "IP-Address Lookup Using LC-Tries," IEEE Journal on Selected Areas in Communications, vol. 17, no. 6, pp. 1083-92, June 1999.
[37] Stefan Nilsson and Matti Tikkanen, "Implementing a dynamic compressed trie", Proceedings of 2nd Workshop on Algorithm Engineering, August 1998.
[38] Stefan Nilsson and Matti Tikkanen, "An experimental study of compression methods for dynamic tries", Algorithmica, 33(1):19-33, 2002.
[39] Geoff Huston, 2017. Are we ready for an IPv6 only Network? APNIC Presentations. AINTEC 2017, Bangkok. Also in Netnod AB, Oct 21 2017. https://www.youtube.com/watch?v=LvOUEjdhHI4
[40] G. Huston, 2020. The Good, Bad, And Ugly Of IPv6. IPv6 Buzz 055, July 9, 2020. https://packetpushers.net/podcast/ipv6-buzz-055-the-good-bad-and-ugly-of-ipv6-with-geoff-huston/
[41] T. Eklöv. The Christmas Goat and IPv6 (Year 12, 11, 10). https://circleid.com/members/4450
[42] IPv6 Adoption. https://www.google.com/intl/en/ipv6/
[43] IEEE INFOCOM 2020 Workshop on New IP, The Next Step. https://infocom.info/workshops/track/New
[44] S. Ren, *et al.*, 2019. Routing and Addressing with LengthVariable IP Address. In ACM SIGCOMM 2019 Workshop on Networking for Emerging Applications and Technologies (NEAT'19), August 19, 2019, Beijing, https://doi.org/10.1145/33415583342204
[45] S. Ren, 2019. FlexIP Addressing. Internet Draft, https://www.ietf.org/archive/id/draft-moskowitz-flexip-addressing-00.txt
[46] Bin S. Sun, 2019. An Improved PLC-Trie Based Routing Table Design for Variable Length IP Address Lookup. In Proceedings of 14th International Conference on Future Internet Technologies (CFI'19), August 7–9, 2019, Phuket, Thailand. https://dl.acm.org/citation.cfm?id=3341189
[47] I. Livadariu, A. Elmokashfi, A. Dhamdhere, 2020. A data centric computational model of IPv6 adoption. In Proceedings of IFIP Networking 2020, 19th International IFIP TC6 Networking Conference, pp. 361-369. Paris, France, June 22-25, 2020. https://ieeexplore.ieee.org/document/9142758 & https://dl.ifip.org/db/conf/networking/networking2020/1570619437.pdf