Space-efficient signaling scheme for IP prefix and realm information in Virtual Networks

ANTTI MÄKELÄ
Aalto University, Department of Communications and Networking, Espoo, Finland, antti.makela@tkk.fi

JOUNI KORHONEN
Nokia Siemens Networks, Helsinki, Finland jouni.korhonen@nsn.com

Keywords: compression, Mobile IP, virtual operator, prefix, realm

As residential Internet access has become increasingly commoditized, the incentives to lower costs in enterprise and similar networks with service level agreements have grown as well. Simply switching to a VPN provided over Internet brings cost savings but at the same time loses any guarantees of service level. We have devised a Mobile IP-based approach to virtualize networks without neglecting Quality of Service. As part of this specific approach, the signaling traffic levels go up as complexity of the network grows. To mitigate this issue, we have created a simple yet effective scheme to compress information on IPv4 network prefixes and realms. In this paper, we present analysis of our scheme's effectiveness and feasibility using both generated and real-world test material. We also consider the extensibility to IPv6 network prefixes.

1. Introduction

Various broadband Internet access technologies have become commoditized in recent years. However, while the commoditized access for public Internet works more than adequately for residential and individual purposes, corporations and other organizations require more customized service. Instead of basic Internet access, organizations often have needs for such services as private site-to-site connectivity and authentication of users – all provided with quality assurances. These, in turn, are not commodities, but may be prohibitively expensive especially for smaller entities.

A virtual service operator model attempts to provide benefits of both commoditized access and customized services. The model is based on the concept of leveraging the commoditized access and implementing an additional, virtual layer providing the required services without costly infrastructure investments. A very basic example is providing site-to-site connectivity with a Virtual Private Network [1] instead of more traditional dedicated line, Frame Relay, ATM or MPLS connections. The traditional and VPN-based site-to-site connectivity are illustrated in *Fig. 1.*

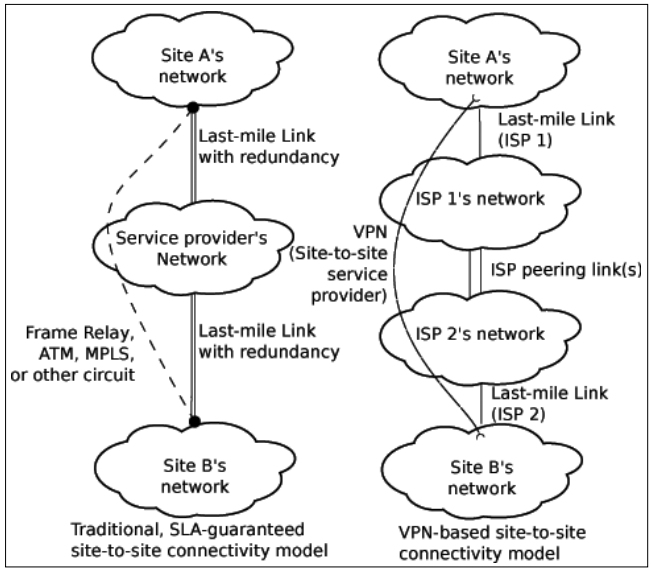
A problematic issue with VPNs as a technology has been lack of flexibility – capabilities to do dynamic reconfiguration of the virtual network when underlying topology changes: e.g., links go up and down, IP allocation changes, and similar issues. Furthermore, redundancy in case of outages is often limited.

The other alternative to VPNs – a traditional multi-homing setup – is also not possible with commodity connections, since in practice you cannot set up routing protocols through such connection. Even if you could, the convergence time of multi-homing can be in the order of minutes.

Allowing sites to have multiple inexpensive, dynamically reconfiguring, redundant connections, possibility to have the sites mobile, and providing all this with at least limited Quality of Service requires new approaches.

As one possible approach to address the issues mentioned earlier, we have proposed a solution based on Mobile IPv4 Network Mobility protocol [2]. The Mobile IPv4 Network Mobility takes care of near-instant switchovers in case of outages. Mobile IP can address the dynamic reconfiguration issue as well; Mobile IP is, by de-

Figure 1. Traditional operator and VPN models



sign, always actively striving for a functional point of attachment. The primary advantage of Mobile IP as a technology compared to e.g. routing protocols is extremely lightweight yet comprehensive signaling, allowing for the fast reaction times to changing conditions. All signaling transactions consist of single request and response messages, yet the message format facilitates complex information structures due to extensibility. Thus, reactions to topology changes can be as fast as single round-trip-time (RTT). Furthermore, Network Mobility allows routing information to be disseminated from a single, centralized point and does not require any sort of routing protocols to be set up for internal use.

Mobile IP Network Mobility has several immediately obvious challenges, especially when the number of sites and site-to-site connections increase. With the basic IPv4 Network Mobility, the Home Agent acts as a topological anchor for all data plane traffic to and from sites, creating a bottleneck at the center of the virtual topology. Therefore, our proposed extensions [3] to the basic Mobile IPv4 Network Mobility protocol add Route Optimization functionality. The intention is to offload most of traffic away from the Home Agent and allow direct and optimal paths between sites, thus conserving Home Agent's bandwidth and avoid the Home Agent becoming the bottleneck.

Route optimization in case of Mobile IPv4 introduces another operational challenge. Assuming that the number of sites and their connectivity is not fixed and varies over time, then a mechanism should exist for each site to learn about each other without extensive preconfiguration. To address this issue, our proposed extensions allow a Home Agent to distribute information about existing Mobile Routers to their peer routers. With this Home Agent assisted Route Optimization ("HAaRO") approach, Mobile Routers, either when registering to the Home Agent or when updating their mobility bindings, will learn of each other, thus being able to route traffic directly between them instead of via the central Home Agent.

Depending on the number of sites and the networks located behind the Mobile Routers the overhead of Mobile IPv4 signaling grows significantly. The signaling overhead may not be a show-stopper in general for the HAaRO approach but needs to be addressed as networks grow.

In order to reduce the signaling overhead when our proposed approach is used, the extensions include simple encoding algorithms for the route optimization information in the Mobile IPv4 signaling messages. There are two algorithms: one for compressing the subnet routing-related IP network prefix information, and one for compressing the administrative scoping-related realm information. In this paper we evaluate the usefulness of the compression algorithms and study their performance in various scenarios. The algorithms have been designed having low computation and memory footprint requirements in mind. Additionally, our algorithms do not expect past history information of previous messages and each message is self-contained compression-wise. This design sacrifices efficiency over simplicity to some degree. The compression of route optimization information is an optimization for the HAaRO approach, not the core functionality of the whole solution. Furthermore, the algorithms may have applications in other areas where IP topology information needs to be transmitted between nodes that have both limited processing power and bandwidth, allowing for a cost-effective means for communication.

This paper is based on earlier work [4] published in ConTEL 2009 conference. New additions are focusing on our efforts to extend the functionality to IPv6 network addresses. Thus we are also studying the algorithms' effectiveness on IPv6 network prefixes, although the original design was based on IPv4 network addressing. Besides that, we have made some clarifications to the text in general and our earlier results.

The rest of the paper is structured as follows. Section 2 has a more in-depth details on the Mobile IP-based Virtual Operator model. Section 3 provides an introduction to the two compression algorithms. Section 4 details our experimentation set-up and testing procedure, with corresponding results of our study. Finally, Section 5 concludes this paper with Section 6 containing some possibilities for future work.

2. Virtual operator model utilizing network mobility and route optimization

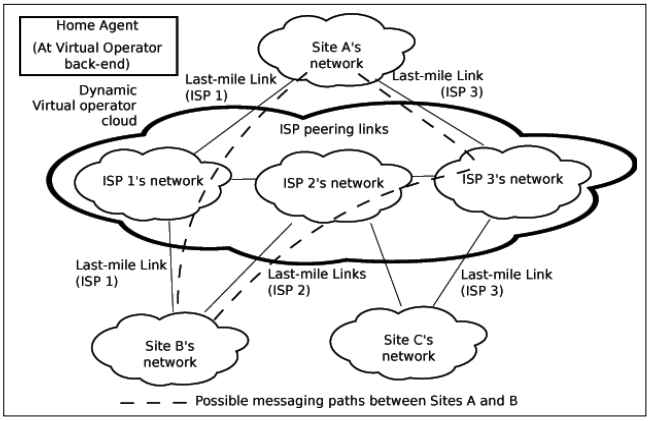
This paper frequently uses terms such as an *IPv4 prefix* [6], *IPv6 prefix* [7], and a *realm* [8].

The realm within this paper defines an administrative domain. Realms are named in similar fashion to Internet domain names, such as *"foo.example.com"*.

The IPv4 prefix corresponds to an IPv4 network address, in form of *a.b.c.d/yy,* where *yy* is the prefix length, in bits. In the same vein, the IPv6 prefix is an IPv6 network address, in form *aaaa:bbbb:cccc:dddd::/yy*. For more details on these concepts and how they relate to our algorithms see Section 3.

Fig. 2 shows an example virtual operator deployment, where the customer entity, such as an enterprise, has

Figure 2. Mobile IP Nemo/Route optimization model



three regional sites (sites A, B and C). Each site is connected to two regular ISPs with regular, commodity Internet connections. For example, Site A-to-B connection has two possible paths – one directly through ISP 1 and other utilizing ISP's 2 and 3. The gateway router at each site connecting to the Internet also acts as a Mobile Router. The regional site networks, in effect, become Mobile Networks. Unlike in the usual case where the mobility processing (handovers) occur when the network physically moves, in this case the mobility is triggered by changing conditions of the ISP's networks. As an example, assume that Site A's primary link is provided by ISP 1. When Site A's link to ISP 1 goes down, the inter-site connectivity remains up – the Mobile IP process simply changes the network's point of attachment to ISP 2 and connectivity is preserved. The scenario is more fully outlined in [5].

As noted earlier, utilizing Mobile IPv4 Network Mobility bestows heavy throughput requirements on the Home Agent and the virtual operator's own back-end network. The HAaRO approach [3] proposes an extensive use of route optimization, where Mobile Routers at customer sites would exchange traffic directly as much as possible without routing traffic via the Home Agent. The HAaRO approach also attempts to solve route optimization related management issues, such as discovering peer Mobile Routers, by having the Home Agent distribute prefix and realm information to Mobile Routers piggybacked in the Mobile IPv4 registration signaling. In this way the virtual operator can provide almost self-organizing composition of dynamically changing mobile networks without extensive configuration management. One area worth mentioning is that full convergence is not required; Triggering of Route Optimization can start at any Mobile Router. Non-optimized traffic is simply forwarded via the Home Agent as a fallback measure. However, assuming that all Mobile Routers have uniform Mobile IPv4 registration refresh time $refresh_{MIP4}$, it is possible to totally distribute the changes in the virtual network approximately within $refresh_{MIP4}$ time.

In IP communication protocols, a very common model is to separate control plane functionality from data plane. Furthermore, the signaling messages that are transmitted on the control plane typically have several constraints. Typical constraint is preference for low bandwidth. Another common constraint is a signaling format requiring all information to be transmitted within single Packet Data Unit (PDU). This is especially the case in request-response type protocols where each request warrants only a single response.

A large enterprise might have tens or hundreds of regional sites with a varying number of IP subnets due to incremental nature of deploying networks. A Home Agent distributing large amount of customer site prefix and realm information easily increases the PDU size beyond what is considered reasonable. While the transport protocol, UDP, allows PDU sizes up to 64 kilobytes, the underlying IP layer is responsible for fragmentation which is typically not desired or at least should be kept to a minimum.

It becomes evident that the growth of the signaling message size must be addressed. This paper evaluates the usefulness of the compression algorithms we have developed for reducing the signaling message size and studies their performance in various scenarios.

3. Space efficient encoding of prefix and realm information

A) Introduction

As stated in Section 2, IP network prefixes correspond to IP network address, in form of *a.b.c.d/yy* (IPv4) or *aaaa:bbbb:cccc:dddd::/yy* (IPv6).

An IP address is divided into *network* portion and *host* portion. In case of IPv4, the highest *yy* bits are considered the network portion identifying a subnet, while lower bits are for distinguishing individual hosts within the subnet. Networks are generally [9] allocated according to the number of expected hosts in the subnet: e.g. a network with prefix length of *yy*=28 consists of 16 host addresses. The first address of the network is the network itself, and last is reserved for broadcast, thus allowing for 14 true hosts. The network may be further subnetted, e.g. to two *yy*=29 networks, each accommodating up to six hosts. The network prefix length can be 0-30, network size /31 having no room for true hosts and /32 being a single IP address.

In case of IPv6, the lowest 64 bits of 128-bit IPv6 address are considered to be the host portion and highest 64 bits are the network portion – these allocations do not change. Thus, for purposes of network prefixes, only the first 64 bits have significance. The network prefix length can thus be 0-64, and primary consideration on prefix length is further subnetting. The prefix length has no effect on number of hosts that can be accommodated in the network.

The compression algorithms for both prefix and realm compression had the following as guiding design goals:

- The algorithms are for generic optimization purposes only, not core functionalities of the HAaRO approach.
- The algorithms should have low computational requirements.
- The algorithms should have a minimal memory footprint as e.g., a Home Agent may be serving thousands of sites, thus memory consumption is an issue.
- The algorithm should not require maintaining state between messages i.e., each compressed message should be self-contained.

Given the above design goals we can already see that the compression efficiency is not going to be among best available algorithms.

There are existing approaches to address IPv4 network prefix compression, for example in the context of MANETs [10]. However, the design parameters of our proposals are different from MANET.

IPv6 has significantly more development. Recently 6LOWPAN [11] has standardized a frame format for the

transmission of IPv6 packets over low-power personal area IEEE 802.15.4 networks [13]. Since the IEEE 802.15.4 maximum physical layer packet size is only 127 octets with 25 octets frame overhead that leaves only 102 octets for the media access control layer. Therefore 6LOWPAN has gone to extreme on defining a space efficient encoding of IPv6 and transport layer headers [12]. From the 6LOWPAN we could actually take advantage of its novel approaches for compressing IPv6 addresses. First, add a specific handling for well-known IPv6 prefixes effectively replacing the static known part up to the first 64 bits of the IPv6 prefix with one bit or maximum two-three more bits. Second, use of *contexts*. In our concept, a *context* is actually close to *Master Prefix,* which is further discussed in Section 3.B. The difference is that in our algorithm the context size has been one, whereas in 6LOWPAN there can be up to 16 contexts.

Besides 6LOWPAN, the IPv6 specification [7] includes zero compression for text formatted IPv6 addresses. However, the zero-compression serves only notational purposes of simplifying human-readable addresses and has no direct significance for our purposes.

B) Prefix compression Algorithm

1. IPv4 prefix compression

The prefix compression algorithm is fully presented in [3], Section 4-1. The basis for the algorithm's design is the assumption that transmitted prefixes will be relatively close to one another – in best case, sequential, (e.g. 1.1.1.0/24, 1.1.2.0/24,...). The assumption stems from common design where an organization is assigned a single IP block which is then subnetted and distributed amongst organization's network sites and the use case with Mobile Routers.

Other design parameters were simplicity in the common case of a single prefix per Mobile Router and possibility to extend the functionality later. One such extension is the IPv6 prefix compression illustrated in next section.

The presented prefix compression algorithm does not define compressor/decompressor implementation; only the data format. There may be more efficient compression implementations (See Section 6) compared to our approaches. The implementation-specific differences stem from the order the prefixes are processed by the algorithm.

The compression algorithm is based on concept of *Master* and *Delta* prefixes. At least one Master Prefix is sent before any Delta Prefixes. After the initial Master Prefix is sent, the following prefixes can be either Master or Delta.

Master prefixes are encoded asis, except tailing octets comprising wholly of zeroes are dropped. Thus e.g. 1.2.3.0/24 is encoded with 3 octets and 1.2.0.0/16 with 2 octets; However, 1.2.0.0/20 is also encoded with 3 octets since prefix length of /20 extends to the third octet, even though the contents of the third octet is zero.

Delta prefixes are always encoded as a single octet; the 8 least-significant bits of the prefix are included. Delta prefixes can be used if the difference between the delta prefix and the master prefix fits to the 8 encoded octets. If the Master Prefix is longer than the current prefix, the tailing bits are not compared due to them implicitly being zeroes. Thus even significantly different prefix may still be compressed if the length is shorter, e.g. if 1.2.3.4/28 is the master prefix, 1.6.0.0/16 can be encoded with a single byte.

Content	O	D	M	Plen	Data
Bits	1	1	1	5	0,8,16,24 or 32

Figure 3. Protocol structure concerning prefixes

When decoding a Delta-encoded prefix, the prefix is formulated as follows:

– Fill the right-hand side of the prefix with zeroes until Prefix Length (Plen) is reached.
– Fill the 8 bits, starting from Plen, from the received Delta prefix.
– Fill the remaining bits from Master Prefix.

The repeatable protocol structure defined in the [3] related to prefixes is presented in *Fig. 3.* Of the three flags, the 'O' concerns network topology and is of no significance to the compression. Flags 'D' and 'M' determine the contents of the Data field.

The data field can contain either a Mobile Router address (if M=1; in this case value of 'D' will be ignored) or a *Master* or *Delta* prefix. The algorithm always maintains the current master prefix; If a new Master prefix is received, the new one will replace the existing one.

Thus, the first time the structure is received, it always has M=1 and gives the Mobile Router the following prefixes are bound to; The next one always has D=0 and M=0, and provides the first Master Prefix. After that all combinations are permissible. Note that Delta prefixes will *always* apply to the current Master Prefix, even if the Mobile Router has changed. Thus, the value of M bit also has no significance to the compression.

Prefix Length is included in the separate 5-bit Plen field, allowing for values of 0-31.

2. Refinements concerning IPv6

The original header octet only allows for prefix length sizes of 0-31, which is too small for IPv6 prefixes. Thus the header octet has been redesigned and is slated to be included in the next version of [3]. The revised version, shown in *Fig. 4,* allows for more signals for Mobile Router purposes and also allows transmitting for IPv6 prefix lengths.

Content	O	D	Plen or indication	Data
Bits	1	1	6	0,8,16,24 or 32

Figure 4. Protocol structure concerning prefixes

We take advantage of the fact that besides IPv4 prefixes of length 0-31, we can handle the cases without prefix information, or with special prefix lengths, by overloading the Plen field. Whether the Plen field corresponds to prefix length or one of the special cases is now identified by a new I-bit (Indication), and old 'M'

and 'O' bits have been removed. The special cases, where I is set to 1, affect interpretation of Plen field as follows:

- When the address is a prefix of length /64, the Plen is set to zero. This effectively extends Plen field to 7 bits to allow for the /64.
- When the prefix length is IPv6-compatible representation of IPv4 prefixes [7], in effect a prefixes of size /96-/127, the Plen is set to a value from 32 to 63, again simply extending the Plen field to 7 bits. When the prefix is a Master Prefix, only 0-4 octets are encoded since the highest 96 bits are well-known.
- When the address is a Mobile Router, that is, corresponding to the old header's case M=1, the Plen is set to 1. As an extended Plen field, this would correspond to a prefix size /65, which is an illegal prefix length for both IPv4 and IPv6.
- When the address is a prefix of length /128, corresponding to a single host, the Plen field is set to 2. As with the case of Mobile Router, this would correspond to a prefix length /66, which is also considered illegal.
- Since the old 'O' field is related to Mobile Router instead of individual prefixes, it can also be indicated with a specific bit within Plen field if I=1. For our efficiency study purposes, we did not assign a specific value.
- Further signaling can be added as necessary.

Since the prefix size variation can be larger with IPv6, it might be beneficial to use more than one single delta byte. However, currently we can only indicate the presence of delta with a single bit, not delta sizes. Additional bits to indicate presence of multiple delta octets would also require an extra octet, thus the effectiveness is questionable.

C) Realm compression Algorithm

In the context of Mobile IP, the term *Realm* is used to signify an administrative domain. "Realm name" can be considered a Domain Name System's (DNS) domain name; Similar structure is evident. The compression algorithm for realm names is inspired by the original domain name compression presented in RFC 1035 [14]. However, our algorithm has been enhanced further to a full-fledged dictionary-based system with a simple, computationally lightweight encoder and decoder. The algorithm is designed to perform well in the common cases of relatively few (maximum 128) separate realm labels. A maximum length of a realm or a domain name is less than 256 octets. Assuming a naive implementation of the dictionary that makes separate copies of the stored strings and some indexing overhead, the maximum dictionary memory usage is around 40 kilobytes (i.e., 128* (255+*overhead*) bytes).

The algorithm works on a label level. The dictionary is updated dynamically with one or more labels from the input "realm name" to the algorithm. Every time a single label or a suffix of a "realm name" is not found in the dictionary, one or more labels are inserted to the dictionary and encoded into the output octet stream using the encoding shown in *Fig. 5.* Of every input "realm name" the suffixes that were not found in the dictionary are also inserted into the dictionary.

7	6	5	4	3	2	1	0	1-127 Octets
0	**Label Length**							**Label**

Figure 5. Bit and octet encoding of a new label

When a label or a suffix of a "realm name" is found in the dictionary, the index to the dictionary is encoded to the output octet stream as shown in *Fig. 6.* The dictionary holds maximum 128 strings. When the 129*th* string would be inserted into the dictionary, the dictionary gets reset and the new string will become the 1*st* string in the dictionary.

Figure 6. Bit and octet encoding of a dictionary index

7	6	5	4	3	2	1	0
1	**Index**						

Consider an example of the input realm *"foo.bar.example.com"*. The compression algorithm searches the dictionary with the following strings:

- *"foo.bar.example.com",*
- *"foo.bar.example",*
- *"foo.bar", and*
- *"foo"*.

The longest found string is encoded. If no matching string was found, the label *"foo"* is encoded and inserted into the dictionary. The process is repeated until the whole input realm has been encoded (i.e. *"bar.example.com"* would be the input realm for the second iteration).

The algorithm also keeps track of the longest suffix per input realm that only consist of encoded labels. Every time an index to the dictionary can be encoded the tracked longest suffix gets reset. Once the input realm has been processed, the longest tracked suffix is inserted into the dictionary.

For example, if *"bar.example.com"* is the longest tracked suffix of *"foo.bar.example.com"*, then the algorithm will insert *"bar.example.com"* and *"example.com"* into the dictionary. The realm compression algorithm implements a greedy approach of encoding matches. The search algorithm and the dictionary update function does not try to optimize updates and immediately encodes the first found match. Lazy evaluation algorithms are known to gain better compression than greedy ones [20].

The octet value 0x00 is used to terminate the encoded stream of input realm. The encoding of both label or index cannot output the value 0x00 and the Mobile IPv4 extensions defined for HAaRO make use of it in header encoding.

The realm compression algorithm has also other interesting properties. The compressor/decompressor state is actually the dictionary. This allows easy and efficient handling of the packet data. There is no need to keep past or future input data available for the algorithm, only the very short piece of data that needs to be com-

pressed or decompressed at time. Furthermore, the entity doing the compression may at any time switch off the compression algorithm without the decompressor noticing it or causing any encoding size penalty compared to completely uncompressed data. This is due the design of the whole encoding of the HAaRO messages. The uncompressed form of the realm information is as compact as the original realm octet string.

1. Realm-compression Algorithm as a basis for dictionary-based IPv6 prefix compression

Due to the limitation of single-byte-deltas, we wanted to study the feasibility of dictionary-based approach for compressing IPv6 prefixes. We adjusted our realm-compression algorithm to make it more suitable for prefix compression while still maintaining same low memory footprint requirements. The changes are based on the idea that single octet of an IPv6 address can now be considered a "label".

The encoding is changed so that in Fig. 5, the "label length" now corresponds to "number of labels following". Since label length is always one octet, there is no need to identify each previously unknown label separately. Furthermore, single octets are not stored in the dictionary as was the case with realms, rather, all sub-prefixes (in contrast to suffixes in realms) are stored instead. Also, instead of using a terminating octet `0x00`, the total length of prefix is encoded as-is before each prefix.

As an example, consider two IPv6 prefixes, 2001:0000:00a9::/55 and 2001:0000:00c0::/42. When encoding the first prefix, the dictionary is empty.

The encoding for the first prefix (besides prefix length) is the following: `0x07 0x20 0x01 0x00 0x00 0x00 0xA9 0x00`. This consists of number of labels following (7 octets, since all labels are unknown), and the prefix asis. After the prefix has been encoded, the following sequences are inserted into dictionary:

2001:0000:00a9:00
2001:0000:00a9
2001:0000:00
2001:0000
2001:00
2001

The shortest sub-prefix, 20, is not inserted, as there is no gain in encoding an octet with another octet.

The encoding of the second prefix is now `0x82 0x01 0xc0`, where `0x82` is reference to the third dictionary entry of "2001:0000:00". `0x01` corresponds to single octet that follows, which is encoded asis (`0xc0`). After encoding, the dictionary gets one additional entry: 2001:0000:00c0, as the previous sub-prefixes are already in dictionary and single octets are not stored.

4. Testing setup and results

A) Prefix compression

Two implementations of the prefix compression algorithm, one for IPv4 and one for IPv6, were written in C, utilizing standard Socket API. These consisted of requester and responder components, with the requester acting in the role of a Mobile Router and responder acting in a role of Home Agent. The responder initially reads in and processes a list of network prefixes associated with Mobile Router addresses, and then starts waiting for a request. The requester contacts the responder with a request message, and the responder returns a list of prefixes, compressed in accordance to the scheme. This followed standard Mobile IP processing, although true Mobile IP headers were not used.

The implementations offers the possibility to send data either uncompressed or compressed. The uncompressed option was used to obtain the baseline comparison for each test case, where neither delta-compression or removing tailing zeroes are used. The compressed option causes the responder to proceed as follows:

1. Read in all the prefixes and Mobile Router addresses associated with each prefix.
2. Group all prefixes managed by single Mobile Router together.
3. For each Mobile Router, sort all prefixes in order; The IP addresses are simply considered unsigned 32-bit integers, or in case of IPv6, unsigned 64-bit integers. Prefix lengths are not taken into account at this point.
4. Merge the prefixes from Mobile Routers back together in the input order.
5. Process prefixes in accordance to the compression algorithm:
   For each prefix, check whether the prefix can be encoded as a delta of previous prefix;
   If yes, encode as delta prefix (1 octet),
   if no, encode as new master prefix (0-8 octets, depending on the prefix length and IP protocol version). If Mobile Router changes, encode the new Mobile Router's address.
6. Save the result into a buffer, waiting to be sent to the requester.

In encoding, we used the formats presented in Section 3.B, the original format for IPv4 prefixes and revised version for IPv6 prefixes.

To come up with realistic test cases, the following considerations were taken into account:

– Intended usage and design:
  Prefixes are likely to be numerically close to each other, including the completely sequential case.
– The algorithm is designed to provide efficient compression while prefix length variations are small. When prefix lengths have greater variation, the delta may no longer fit into a single octet and a new master prefix has to be set, consuming space. This is especially a concern with IPv6.
– Number of prefixes may vary depending on organizations size.
– Number of Mobile Routers may vary depending on organizations size.

With this in mind, a prefix generator was implemented that can generate list of IPv4 network prefixes in *a.b.c.d/yy* format or IPv6 network prefixes in *aaaa:bbbb:*

cccc:dddd::/yy format, and associate each prefix with a Mobile Router address. This prefix list could then be fed to the responder component of the implementation, allowing for a quick testing of various cases. Generator can provide selectable number of prefixes of various lengths around specified base prefix length (*/yy*) with selectable degree of randomness induced both to prefix length and prefix sequentialness. The random number generator used was standard ISO C rand() function which provides 32-bit random integers with uniform distribution. For IPv6 prefixes, two 32-bit integers were concatenated to form a single 64-bit prefix. The random number generator was initialized using seed derived from system time before each new batch of prefixes.

In the cases where randomness plays a part – e.g. when testing the performance where the compressed prefixes are not completely sequential or prefix lengths vary – each test case was conducted 10 times and results averaged. Minimum and maximum of each case was also recorded to check for outliers.

Besides these generated test cases, a more realistic source for real-world network prefixes was used for IPv4 – the global Internet BGP routing table. In case of IPv4, picking an A-class network of appropriate size to correspond for each generated test set allowed a comparison of the generator to real-world subnetting schemes.

The test matrix chosen includes 10, 500 and 5000 prefixes, in sequential, near-sequential or totally random order, shared equally between either 1 or 10 Mobile Routers. Furthermore, each case had the prefix length either as static (/24), or varying to a smaller (<8 bits) or greater (<16 bits) degree. The "totally random" case attempted to utilize entire (IPv4 or IPv6) address space, to create appearance of totally unrelated network prefixes. "Near-sequential" simply means occasionally skipping the next network in sequence, and is our closest expectation of real-world use-case.

In addition, three test cases for IPv4 were taken from global BGP routing Table [15]. The routing table was based on data on 28th of August, 2008. The chosen /8 networks and their sizes are listed in *Table 1*.

It should be noted that when conducting IPv6 tests, lessons learned from IPv4 testing could already be taken into account, and some IPv6-specific tests concerning the increased address space were added. These differences are further detailed in Section 4.G.

B) IPv4 network prefix compression results

Before measuring compression efficiency, a baseline had to be established. The baseline for IPv4 prefix compression – simply not compressing data at all – can be seen in *Table 2*.

The non-compressed size for each case shows that as number of prefixes grow, the number of Mobile Routers has less and less proportional effect to the overall size – each new Mobile Router adds a static five octets (1 header octet, 4 octets for address) to the data. Thus, for further observations, the case with 10 MRs is not significantly different from the case with a single MR as an additional MRs simply increases data size by constant 5 octets each.

The effect of sequentialness to the compression is shown in *Table 3*, based on the case where prefix lengths do not vary. The compression factors in the table are based on the average compression in each of the randomized cases. As can be seen, the performance is best when prefixes are sequential and worst when random. The real-world networking data, shown in the BGP column, appears to reflect the sequential/near-sequential cases more than the completely random case, which is encouraging.

As mentioned, all test cases with random elements were ran 10 times. Under no circumstances did the individual test runs significantly differ from the average.

Table 1. Real-world BGP routing table entries

Network	Number of Prefixes	Purpose
56.0.0.0/8	10	Comparison for 10 prefixes
129.0.0.0/8	497	Comparison for 500 prefixes
72.0.0.0/8	4534	Comparison for 5000 prefixes
64.0.0.0/8	5907	Comparison for 5000 prefixes

Table 2. IPv4 Network Prefix compression test cases, uncompressed data sizes

Prefixes/MRs	Uncompressed size
10/1	55
10/10	100
500/1	2505
500/10	2550
5000/1	25005
5000/10	25050

The results in *Table 3* are based on the case where the prefixes are all of same length; In this case, /24. The effect of varying the prefix lengths are shown in *Table 4*. The values in percentages show compressed data length compared to uncompressed data length with same number of prefixes. The effect of adding varying prefix lengths to a small degree (cases with /8, prefix lengths from /20 to /27) causes what was expected compared to static

Table 3. Effect of IPv4 prefix sequence to compression. Percentages in parenthesis are the remaining data compared to uncompressed data.

Count	Sequential	Near-seq	Random	BGP
10	27 (49%)	27 (49%)	45 (88%)	(49%)
500	1009 (40%)	1017 (41%)	1973 (79%)	(45%)
5000	10045 (40%)	10124 (40%)	19697 (79%)	(41/42%)

Table 4. Effect of varying prefix lengths on compression. Case indicates Number of Prefixes / Maximum variation.

Case	Uncompr	Sequential	Near-seq	Random
10/8	55	27 (49%)	29 (53%)	48 (87%)
500/8	2505	1027 (41%)	1225 (49%)	1973 (79%)
5000/8	25005	10241 (41%)	12261 (49%)	18934 (76%)
10/16	55	27 (50%)	32 (59%)	44 (80%)
500/16	2505	1030 (41%)	1379 (55%)	1793 (72%)
5000/16	25005	10237 (41%)	13643 (55%)	16226 (65%)

case – the compression factor decreases. Remaining data length is around 50% of the original compared to 40% in the non-varying case.

Increasing the variability to a larger degree (cases /16, prefix lengths from /16 to /31) causes more interesting results. In the sequential case, results are identical with the smaller variation, which was expected. With near-sequential cases, the resulting data size is slightly larger but not significantly, which is expected. However, in the totally random case, the compressed data size is actually smaller than with smaller variance.

The reason for the better compression ratio in totally random case, especially with the full 5000 prefix set, might stem from the limits of IPv4 address space. As our definition of "random" is filling the entire 32-bit address space, and the data set includes short prefixes, the individual prefixes in the data set may not be so unrelated after all – a very short prefix may include significant portion of the entire address space.

However, since the effect also appears with cases of 500 and 10 prefixes, the more likely reason is the algorithm's ability to drop tailing zeroes and make the delta to higher-end bits of the master prefix. When prefix length variation is small, almost all prefixes become master prefixes. When prefix length variations are higher, the tailing zeroes effect kicks in; e.g. a /28 prefix followed by /16 one. The latter prefix quite likely has identical or near-identical most significant bits, and the tailing zeroes can be left out; Thus it becomes a delta-prefix. e.g. when Master Prefix being 1.12.23.16/28 the prefix 1.13.0.0/16 can be expressed as a delta, while 1.13.34.32/28 would be encoded as a new Master Prefix.

C) Realm compression

Our C++ implementation of the realm compressor/decompressor was tested against the test material containing 500 *Fully Qualified Domain Names* (FQDN) generated from the zone *"example.com"* (total size of 11188 octets). In reality a MIP4 message would contain in maximum few dozens of realms. The material was generated using a list of English dictionary words of 3 letters and higher, a subzone label (*aaaa...xxxx*) repeating 25 times and suffix *"example.com"*.

Finding a comparable real-world case to determine the accuracy of our generated examples is hard in this case, as the usage pattern is different from regular DNS. However, regular DNS deployment can still give some indication on the algorithms effectiveness on domain/realm names. Thus, we retrieved the all FQDNs from zone *"cs.tut.fi"*, consisting of 1285 entries and size of 26403 octets. A real HAaRO deployment is not intended to be used with so many different entries.

The ordering of realms in the input data was expected to have an impact on the compression efficiency. If realms that share common suffixes are close to each other, then the dictionary gets updated less frequently. Low number of updates to the dictionary implies fewer dictionary resets, which in turn was expected to have positive effect on compression. Each dictionary reset causes the compressor to lose its current context. It will take a while for the compressor to re-populate the dictionary and provide good compression again.

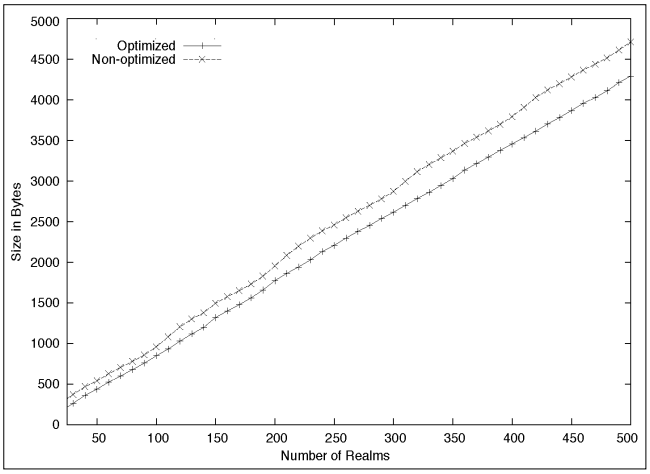
D) Realm compression results

As mentioned, our realm compression test material is a generated list of FQDNs (i.e., realms in this paper) from *"example.com"* domain and containing up to 500 realms consisting total of 11188 characters. In optimal order, with least number of dictionary resets, the algorithm compresses it to 4294 octets, compressing the data to 38% of the original size. The optimal order in this case being one where all realms within the same subzone (e.g. *"aaaa.example.com"*) are all presented in sequence.

With a harder, non-optimal order, the compressed data contains 4716 octets, leaving 42% of original length. As the dictionary has 128 entries, even non-optimal ordering does not cause significant efficiency losses until after first reset. The "non-optimal" order is one where the adjacent subzones are always different (e.g. *"bbbb.example. com"*, *"cccc.example.com"*, ...) until after 25 cycles.

The "bumps" caused by dictionary resets can be seen in *Fig. 7*.

Figure 7. Realm compression results



Approximately after every 100*th* input of non-optimally ordered realms, the dictionary gets reset. For optimally ordered realms the resets are less frequent and less visible due the dictionary learning faster the current context. In overall we are satisfied of the result given the simplicity of our algorithm.

Comparing the results from generated data and our chosen real-world example, the zone "cs.tut.fi" compressed from 26403 octets to 13783, leaving 52% of original data. Considering that the example is from a standard DNS deployment where host names vary significantly, slightly worse compression factor is to be expected.

E) Combined IPv4 prefix and realm compression

Most interesting results are expected when both prefix and realm compression are used together, the reason being that both prefix and realm input data compresses differently depending on the order the individual prefixes and realms are stored in the input data.

In the HAaRO specification, there is a dependency between prefixes and realms. Each prefix may be associated with zero or one realms. A realm cannot be encoded on its own without a prefix. However, prefixes can be encoded without realms.

The optimal ordering of prefixes may not favor realms, and vice versa. Therefore, in the combined case we examine what is the impact of favoring either prefixes or realms optimal ordering during the compression. We also present results of unrealistic case where both prefixes and realms are ordered optimally for a comparison purposes (this is not supported by the current signaling message format although network design may take this into account).

F) Combined IPv4 prefix and realm compression results

Fig. 8 shows the results of combined prefix and realm compression. We can make two obvious conclusions. First, the compression has significant effect on the message size and given the simplicity of our algorithms, their use should be encouraged. Second, the ordering of the HAaRO information has an impact on the compression efficiency. We can see that ordering by a prefix will give the best results due the nature of our algorithms – the prefix compression algorithm is a simple delta coding, which effectively only has a history of the one immediately preceding prefix. Therefore it is more sensitive to sudden changes in the input data than the realm compression algorithm which has a history of up to 128 preceding labels and realms. Thus, it's better to optimize the ordering by the algorithm that is most sensitive to the input data and it that way gain the best overall result.

It would be possible to design the Mobile IPv4 message extensions in such way that optimal ordering of both prefixes and realms were possible. However, this would imply increase in extension header overhead, which would effectively void the slightly better gains in compression efficiency. Furthermore, the message extension handling would complicate as two distinct sets of data should be kept in memory until the decompression has completed in order to allow merging of prefix and realm information.

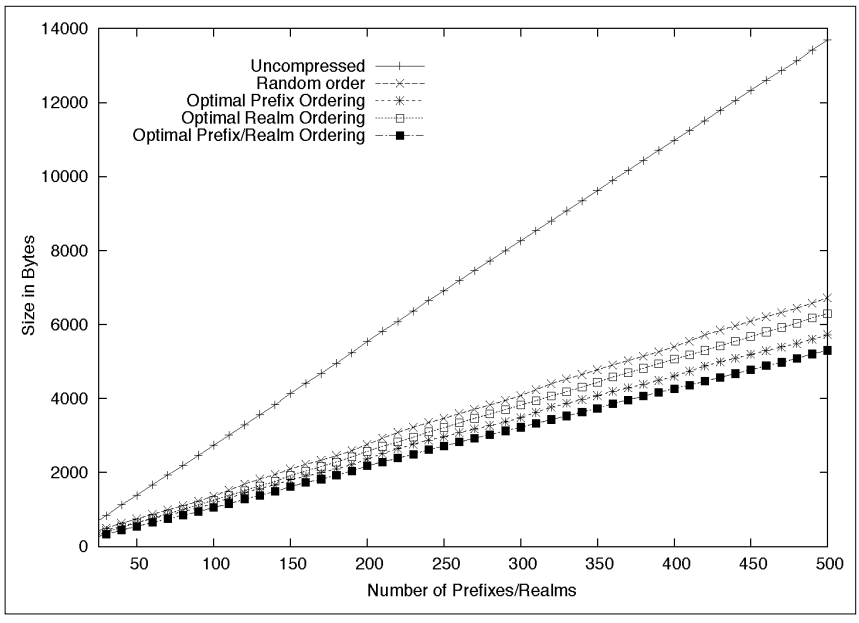
G) IPv6 prefix compression considerations

As stated in Section 3.B-2, IPv6 network prefixes have certain differences to IPv4 network prefixes, which affects test case planning. Furthermore, certain lessons from IPv4 tests could be taken into account.

Since one of IPv6's goals is simplifying routing tables and preventing fragmentation of IPv6 address space, the BGP table for IPv6 is not such a good source for real-world network prefix allocation, as routing strategies prefer to have only Provider Aggregated (PA) addresses to appear in global routing tables [16]. In IPv4, Provider Independent(PI) addresses assigned to organizations are widely used and thus appear in global BGP tables, but they are not considered a scalable solution for IPv6 multihoming [17]. The BGP table for IPv6 thus shows only aggregated address spaces allocated for service providers – the subnetting of the address space is not visible.

IPv6 deployment is still in early phases, thus no "best practices" based on operational experience have emerged. There are a number of IPv6 related recommendations and guidelines for various cellular networks produced for example by IETF. These recommendations mainly concern what is the link model of certain cellular technology, how it shows to the IPv6 stack, what is the recommended addressing for the link model and

Figure 8.
Results of combined prefix and realm compression



so on [18,19]. One particularly interesting point regarding addressing is that in certain cases, IETF has recommended the use of a separate /64 network prefix for each user (or rather mobile host), even if the network contains only a single host. This addressing model has actually been adopted by all major cellular macro networks such as WiMAX and 3GPP.

Also, in more traditional environments, the allocated prefix sizes for a site can be anything from /32 to /64, although the most common allocations appear to be /48, as recommended by the specifications. However, due to the static-sized "host portion", for optimal allocation the prefix size per a site should not be concerned with number of hosts, but number of individual sub-networks. Thus a site with three networks should be allocated a /62. However, an ISP may choose a more relaxed allocation posture and e.g. allocate a /56 or even /48 even though only fraction of the possible subnets will be used.

For IPv6 prefixes, we replicated same test cases as we did with IPv4: 10, 500 and 5000 prefixes, with varying prefix lengths around /48 (comparable to IPv4's /24). Additional cases stem from the mentioned IPv6's address allocation characteristics: Due to the abundance of address space, the prefix lengths might not be fine-tuned to match the needs and even allocated in a lazy fashion. Thus, we added the case where the prefix lengths vary even more, up to 32 bits (causing prefix lengths from /32 to /64 to be generated).

As stated in Section 3.C-1, we also checked the possibility to use dictionary-based algorithm, based on our realm compression work, to compress prefixes. This was also tested with each case.

H) IPv6 network prefix compression results

As stated earlier, the IPv6 work was only commenced as IPv4 work had been completed. Therefore, some specific test cases could be omitted from the start, namely omitting any considerations for the number of Mobile Routers – as with IPv4 prefixes, these would only increase the payload by constant amount each. As with IPv4, a baseline with no compression needed to be established. Affecting considerations for valid baseline is the "zero compression" for addresses mentioned in IPv6 specifications [7].

Although compressing the largest block of zeroes in an address is beneficial, the approach is purely for human-readable text notation. The network equipment still treats IPv6 addresses as 128-bit numbers, even if they primarily consist of zeroes. From signaling perspective, however, we are already utilizing zero compression – tail-dropping zeroes that exceed prefix length.

While a full IPv6 address is 128 bits (16 octets), the latter 8 octets are never considered to be part of a prefix. Thus, instead of considering "uncompressed" data to consist of full 128-bit IPv6 addresses, we are in this case using a value of 9 octets per prefix, counting 1 octet for encoding prefix length and 8 octets for the prefix (highest 64 bits of an IPv6 address). The uncompressed sizes for various data set sizes are shown in *Table 5*.

Prefixes	Uncompressed size
10	90
500	4500
5000	45000

Table 5. IPv6 network prefix compression test cases, uncompressed data sizes

The effect of sequentialness to the compression is shown in *Table 6,* in the case where prefix lengths do not vary. The test case that the table illustrates is functionally identical to the IPv4 Table 3. However, it should be noted that as the base prefix length of /48 was chosen, it is clear that in random case the compression is achieved solely because the tail-dropping of zeroes, in contrast to the IPv4 case where the scarcity of address space allowed for at least some compression. However, percentage-wise, the sequential or near-sequential cases are still impressive. As with IPv4, individual test cases did not significantly differ from the average even in extreme cases.

When varying prefix lengths, things get more interesting. In all cases, the results seem to get better with 500 prefixes than with 10 prefixes, implying that the algorithms effectiveness increases with larger data sets as effect of encoded Master Prefix decreases. In the cases where there sequence has gaps, the efficiency worsens the greater the variance in prefix sizes is, which is

Table 6.
Effect of IPv6 prefix sequence to compression. Percentages in parenthesis are the remaining data compared to uncompressed data.

Count	Sequential	Near-seq	Random
10	25 (28%)	25 (28%)	70 (78%)
500	1010 (22%)	1030 (23%)	3500 (78%)
5000	10100 (22%)	10290 (23%)	35000 (78%)

Table 7. Effect of varying prefix lengths on compression

Count	Uncompr	Sequential	Near-seq	Random
10/8	90	25 (27%)	25 (28%)	74 (82%)
500/8	4500	1046 (23%)	1461 (32%)	3682 (82%)
5000/8	45000	10459 (23%)	14655 (33%)	36859 (82%)
10/16	90	26 (28%)	39 (43%)	74 (82%)
500/16	4500	1051 (23%)	1736 (39%)	3689 (82%)
5000/16	45000	10465 (23%)	17576 (39%)	36889 (82%)
10/32	90	37 (41%)	48 (53%)	82 (91%)
500/32	4500	1064 (24%)	2109 (47%)	3487 (77%)
5000/32	45000	10545 (23%)	21078 (46%)	33474 (74%)

somewhat expected. As before, in the totally random cases the efficiency depends solely on the tail-dropping of zeroes. In random cases, space consumed is greater than with identical prefix lengths case with smaller variances, but with the greatest variance tail-dropping nearly half of the prefix (for /32's) kicks in.

1.Comparison to dictionary based algorithm

As stated in Section 3.C-1, we also tested the feasibility of dictionary-based compression on batches of IPv6 prefixes. Overall, the results are no better than with the delta-compression algorithm, and in most cases worse. For comparison purposes, cases with 5000 prefixes, where any local anomalies have leveled off, are presented in *Table 8*. The uncompressed data in this case is always 45000 bytes.

Case	Delta-compr	Dictionary-compr
Single length, sequence	10100 (22%)	20023 (50%)
Single length, near seq	10296 (23%)	20062 (51%)
Variable length/8, seq	10459 (23%)	20964 (50%)
Variable length/16, partial seq	17576 (39%)	22543 (78%)
Variable length/32, partial seq	21078 (47%)	24393 (54%)

Table 8.
Comparison of algorithms in cases with 5000 prefixes

As can be seen, even though the delta-compression only has a single delta-byte, the dictionary-based approach has significantly poorer performance, even in the most optimal case of sequential networks. The differences are probably stemming from the fact that in dictionary compression there is no concept of "Master Prefix" as such – there are number of "Master Prefixes" in the dictionary, but no implicit one – so master has to always be explicitly encoded, even if it is simply a single label.

5. Conclusions

Although the presented Prefix and Realm compression algorithms are not designed to provide the most efficient compression factor possible, considering their simplicity and the specific application they perform remarkably well. Our generated test-material also appears to be accurate representation of real-world networking in Prefix Compression case, at least for IPv4.

Prefix and Realm Compression algorithms presented allow for a simple and efficient way to transmit prefix and realm information over standard Mobile IPv4 signaling messages. The effectiveness depends on the order the items are fed to the algorithm; We conclude that optimal prefix ordering should be prioritized over realm ordering. The use cases also support this approach, as number of realms is usually lower than the number of prefixes. On a typical case, we were able to compress the data to approximately 40% of original size.

When extending the prefix compression algorithm to IPv6, it scales surprisingly well provided that the network address allocation strategy conserves addresses. Within a single organization this can well be the case. If address space is used in a more erratic fashion, the efficiency suffers.

While these algorithms provide a quick and simple approach to reduce signaling traffic, a different scheme might be more desirable for larger networks. However, this approach can be implemented with relatively low effort and appears to scale up to several hundred prefixes and realms before even requiring the problematic fragmentation of signaling PDUs.

6. Future work

The compression algorithms presented in this paper are only very basic, simple methods for compressing the realm and prefix information according to the specification. Optimizing the prefix- and realm compression order may provide higher gains. Areas to improve is the significance of Mobile Router compression order; Also, the Mobile Router address itself might be eligible for compression. In case of IPv6, it may also be possible to add variable number of delta octets for added benefit, if prefix length can somehow be more efficiently encoded. Using relative prefix lengths might be a possibility, since it's quite likely that prefix length variation will be limited.

For realm compression algorithm, a more intelligent label and suffix replacement algorithm instead of a complete dictionary reset would most probably better the realm compression significantly.

Acknowledgment

Antti Mäkelä acknowledges TEKES GIGA Program WISEciti project and FI ICT SHOK programs under which this work was conducted.

Authors

ANTTI MÄKELÄ is a post-graduate student and researcher at Aalto University, Department of Communications and Networking. His work history also includes 9 years at largest telecommunications operator in Scandinavia, Teliasonera corporation, where he participated in several research projects covering IPv6 deployment and emerging security challenges before moving to academia. His current areas of interest include virtual operators as business model, IP mobility and routing, and future Internet architectures in general. Mäkelä received his MSc (2002) in Computer Science from Tampere University of Technology, Finland, and is now working on his Ph.D dissertation. He is an active participant in IETF standardization organization. As an additional recognition, he has obtained a Cisco Certified Internetwork Expert (CCIE) certification and is recognized as an inventor in a number of patents in his areas of interest.

JOUNI KORHONEN is a Senior Specialist, Internet standards, at Nokia Siemens Networks. His current areas of interest and responsibilities include Internet technologies at large, wireless cellular network architectures, and the development of a future Internet combined with wireless broadband. He currently participates and leads several projects related to a future (wireless) Internet and also IPv6 deployment challenges. Korhonen received both his Ph.D (2008) and M.Sc. (1998) in Computer Science from the University of Helsinki, Finland. He is an active participant in IETF and 3GPP standardization organizations. He currently serves as the co-chair of the Diameter Maintenance and Extensions (DIME) working group in IETF. He has authored a number of publications, IETF RFCs and 3GPP standards. He also holds a number of patents on his areas of interest.

References

[1] Z. Zhang, X. Chu, B. Li, Y-Q. Zhang,
An overview of Virtual Private Network (VPN):
IP VPN and Optical VPN,
Photonic Network Communications, Vol. 7, Nr. 3,
May 2004.

[2] K. Leung, G. Dommety, V. Narayanan, A. Petrescu,
Network Mobility (NEMO) Extensions for Mobile IPv4,
IETF RFC 5177, April 2008.

[3] A. Mäkelä, J. Korhonen,
Home agent assisted route optimization for
mobile IPv4,
IETF Draft draft-ietf-mip4-nemo-haaro-00.txt,
(Work in progress), 2010.

[4] A. Mäkelä, J. Korhonen,
Space-efficient signaling scheme for
Home Agent Assisted Route Optimization for use
in Virtual Networks,
In Proc. of the 10th International Conference on
Telecommunications (ConTEL 2009), June 2009.

[5] A. Mäkelä,
Concept for providing guaranteed service level over
an array of unguaranteed commodity connections,
In Proc. of the 25th Symposium on Applied Computing
(ACM SAC 2010), March 2010.

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

[7] R. Hinden, S. Deering,
Internet Protocol Version 6 Addressing Architecture,
IETF RFC 4291, February 2006.

[8] B. Aboba, et al,
Review of Roaming Implementations,
IETF RFC 2194, September 1997.

[9] R. Bush, et al,
IPv4 Address Allocation and Assignment Policies
for the RIPE NCC Service Region,
RIPE Document ID 492, February 2010.

[10] T. Clausen, C. Dearlove, J. Dean, C. Adjih,
Generalized MANET Packet/Message Format,
IETF RFC 5144, February 2009.

[11] G. Montenegro, N. Kushalnagar, J. Hui, D. Culler,
Transmission of IPv6 Packets over
IEEE 802.15.4 Networks,
IETF RFC 4944, September 2007.

[12] J. Hui, P. Thubert,
Compression Format for IPv6 Datagrams
in 6LoWPAN Networks,
IETF Draft draft-ietf-6lowpan-hc-07.txt,
(Work in progress), 2010.

[13] IEEE Computer Society,
IEEE Std. 802.15.4-2003, IEEE, October 2003.

[14] P. Mockapetris,
Domain names – Implementation and Specification,
IETF RFC 1035, 1987.

[15] BGP Update Reports,
IPv4 address pools, Advertised.,
http://bgp.potaroo.net/ipv4-stats/prefixes_adv_pool.txt
(Fetched on August 28th, 2008)

[16] G. Van de Velde, C. Popoviciu, T. Chown,
O. Bonness, C. Hahn,
IPv6 Unicast Address Assignment Considerations,
IETF RFC 5375, December 2008.

[17] G. Huston,
Architectural Approaches to Multi-homing for IPv6,
IETF RFC 4177, September 2005.

[18] M. Wasserman,
Recommendations for IPv6 in 3rd Generation
Partnership Project (3GPP) Standards,
IETF RFC 3314, September 2002.

[19] B. Patil, F. Xia, B. Sarikaya, J.H. Choi, S. Madanapalli,
Transmission of IPv6 via the IPv6 Convergence
Sublayer over IEEE 802.16 Networks,
IETF RFC 5121, February 2008.

[20] R.N. Horspool,
The effect of non-greedy parsing in Ziv-Lempel
compression methods,
IEEE Data Compression Conf., pp.302–311, 1995.