Please post. Tnx.

INTERNET-DRAFT
<draft-ietf-ipngwg-addrconf-privacy-03.txt>

Thomas Narten
IBM
Richard Draves
Microsoft Research
September 19, 2000

Privacy Extensions for Stateless Address Autoconfiguration in IPv6

<draft-ietf-ipngwg-addrconf-privacy-03.txt>

Status of this Memo

Abstract

Nodes use IPv6 stateless address autoconfiguration to generate addresses without the necessity of a DHCP server. Addresses are formed by combining network prefixes with an interface identifier. On interfaces that contain embedded IEEE Identifiers, the interface identifier is typically derived from it. On other interface types, the interface identifier is generated through other means, for example, via random number generation. This document describes an extension to IPv6 stateless address autoconfiguration for interfaces whose interface identifier is derived from an IEEE identifier. Use of the extension causes nodes to generate global-scope addresses from

interface identifiers that change over time, even in cases where the interface contains an embedded IEEE identifier. Changing the interface identifier (and the global-scope addresses generated from it) over time makes it more difficult for eavesdroppers and other information collectors to identify when different addresses used in

different transactions actually correspond to the same node.

Contents

1. Introduction

Stateless address autoconfiguration [ADDRCONF] defines how an IPv6

node generates addresses without the need for a DHCP server. Some types of network interfaces come with an embedded IEEE Identifier (i.e., a link-layer MAC address), and in those cases stateless address autoconfiguration uses the IEEE identifier to generate a 64-bit interface identifier [ADDRARCH]. By design, the interface identifier is globally unique when generated in this fashion. The interface identifier is in turn appended to a prefix to form a 128-bit IPv6 address.

All nodes combine interface identifiers (whether derived from an IEEE identifier or generated through some other technique) with the reserved link-local prefix to generate link-local addresses for their

attached interfaces. Additional addresses, including site-local and global-scope addresses, are then created by combining prefixes advertised in Router Advertisements via Neighbor Discovery [DISCOVERY] with the interface identifier.

Not all nodes and interfaces contain IEEE identifiers. In such cases, an interface identifier is generated through some other means (e.g., at random), and the resultant interface identifier is not globally unique and may also change over time. The focus of this document is on addresses derived from IEEE identifiers, as the concern being addressed exists only in those cases where the interface identifier is globally unique and non-changing. The rest of this document assumes that IEEE identifiers are being used, but the techniques described may also apply to interfaces with other types of globally unique and persistent identifiers.

This document discusses concerns associated with the embedding of non-changing interface identifiers within IPv6 addresses and describes extensions to stateless address autoconfiguration that can help mitigate those concerns in environments where such concerns are significant. Section 2 provides background information on the issue. Section 3 describes a procedure for generating alternate interface identifiers and global-scope addresses. Section 4 discusses implications of changing interface identifiers.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [KEYWORDS].

2. Background

This section discusses the problem in more detail, provides context for evaluating the significance of the concerns in specific environments and makes comparisons with existing practices.

2.1. Extended Use of the Same Identifier

The use of a non-changing interface identifier to form addresses is a specific instance of the more general case where a constant identifier is reused over an extended period of time and in multiple independent activities. Anytime the same identifier is used in multiple contexts, it becomes possible for that identifier to be used to correlate seemingly unrelated activity. For example, a network sniffer placed strategically on a link across which all traffic to/from a particular host crosses could keep track of which destinations a node communicated with and at what times. Such

information can in some cases be used to infer things, such as what hours an employee was active, when someone is at home, etc.

One of the requirements for correlating seemingly unrelated activities is the use (and reuse) of an identifier that is recognizable over time within different contexts. IP addresses provide one obvious example, but there are more. Many nodes also have DNS names associated with their addresses, in which case the DNS name serves as a similar identifier. Although the DNS name associated with an address is more work to obtain (it may require a DNS query) the information is often readily available. In such cases, changing the address on a machine over time would do little to address the concern raised in this document, as the DNS name would become the correlating identifier.

The use of a constant identifier within an address is of special concern because addresses are a fundamental requirement of communication and cannot easily be hidden from eavesdroppers and other parties. Even when higher layers encrypt their payloads, addresses in packet headers appear in the clear. Consequently, if a mobile host (e.g., laptop) accessed the network from several different locations, an eavesdropper might be able to track the

movement of that mobile host from place to place, even if the upper layer payloads were encrypted [SERIALNUM].

2.2. Not a New Issue

Although the topic of this document may at first appear to be an issue new to IPv6, similar issues exist in today's Internet already. That is, addresses used in today's Internet are often non-changing in practice for extended periods of time. In many sites, addresses are assigned statically; such addresses typically change infrequently. However, many sites are moving away from static allocation to dynamic allocation via DHCP [DHCP]. In theory, the address a client gets via DHCP can change over time, but in practice servers return the same address to the same client (unless addresses are in such short supply that they are reused immediately by a different node when they become free). Thus, although many sites use DHCP, clients end up using the same address for months at a time.

Nodes that need a (non-changing) DNS name generally have static addresses assigned to them to simplify the configuration of DNS servers. Although Dynamic DNS [DDNS] can be used to update the DNS dynamically, it is not widely deployed today. In addition, changing an address but keeping the same DNS name does not really address the underlying concern, since the DNS name becomes a non-changing identifier. Servers generally require a DNS name (so clients can

connect to them), and clients often do as well (e.g., some servers refuse to speak to a client whose address cannot be mapped into a DNS name that also maps back into the same address).

Many network services require that the client authenticate itself to the server before gaining access to a resource. The authentication step binds the activity (e.g., TCP connection) to a specific entity (e.g., an end user). In such cases, a server already has the ability to track usage by an individual, independent of the address they happen to use. Indeed, such tracking is an important part of accounting.

Web browsers and servers typically exchange "cookies" with each other [COOKIES]. Cookies allow web servers to correlate a current activity with a previous activity. One common usage is to send back targeted

advertising to a user by using the cookie supplied by the browser to identify what earlier queries had been made (e.g., for what type of information). Based on the earlier queries, advertisements can be targeted to match the (assumed) interests of the end-user.

The use of non-changing interface identifiers in IPv6 has implications in two quite different contexts: stationary devices (i.e., those that generally do not move physically such as desktop PCs), and mobile devices (i.e., those that move frequently, including laptops, cell phones, etc.).

In today's internet, many home users do not have permanent connections and indeed are assigned temporary addresses each time they connect to their ISP. Consequently, the addresses they use change frequently over time and are shared among a number of different users. If addresses are generated from an interface identifier, however, a home user's address could contain an interface identifier that remains the same from one dialup session to the next. The way PPP is used today, however, PPP servers typically unilaterally inform the client what address they are to use (i.e., the client doesn't generate one on its own). This practice, if continued in IPv6, would avoid the concerns that are the focus of this document.

A more interesting case concerns always-on connections (e.g., cable modems, ISDN, DSL, etc.) that result in a home site using the same address for extended periods of time. This is a scenario that is just starting to become common in IPv4 and promises to become more of a concern as always-on internet connectivity becomes widely available. The technique described later in the document attempts to address this concern by changing the interface identifier portion of an address. However, it should be noted that in the case of always-on connections, the network prefix portion of an address is in effect a

constant identifier. All nodes at (say) a home, would have the same network prefix. This has implications for privacy, though not at the same granularity (i.e., all nodes within a home would be lumped together for the purposes of collecting information). This issue is also non-trivial to address, because the routing prefix part of an address contains topology information and cannot contain arbitrary values.

Another case concerns mobile devices (e.g., laptops, PDAs, etc.) that move topologically within the Internet. Whenever they move (in the absence of technology such as mobile IP [MOBILEIP]), they form new addresses for their current topological point of attachment. This is typified today by the "road warrior" who has Internet connectivity both at home and at the office. While the node's address changes as it moves, however, the interface identifier contained within the address remains the same (when derived from an IEEE Identifier). In such cases, the interface identifier could (in theory) be used to track the movement and usage of a particular machine [SERIALNUM]. For example, a server that logs usage information together with a source addresses, is also recording the interface identifier since it is embedded within an address. Consequently, any data-mining technique that correlates activity based on addresses could easily be extended to do the same using the interface identifier. This is of particular concern with the expected proliferation of next-generation network-connected devices (e.g., PDAs, cell phones, etc.) in which large numbers of devices are in practice associated with individual users (i.e., not shared). Thus, the interface identifier embedded within an address could be used to track activities of an individual, even as they move topologically within the internet.

2.3. Possible Approaches

One way to avoid some of the problems discussed above is to use DHCP for obtaining addresses. With DHCP, the DHCP server could arrange to hand out addresses that change over time.

Another approach, compatible with the stateless address autoconfiguration architecture, would be to change the interface id portion of an address over time and generate new addresses from the interface identifier for some address scopes. Changing the interface identifier can make it more difficult to look at the IP addresses in independent transactions and identify which ones actually correspond to the same node, both in the case where the routing prefix portion of an address changes and when it does not.

Many machines function as both clients and servers. In such cases, the machine would need a DNS name for its use as a server. Whether

the address stays fixed or changes has little privacy implication

since the DNS name remains constant and serves as a constant identifier. When acting as a client (e.g., initiating communication), however, such a machine may want to vary the addresses it uses. In such environments, one may need multiple addresses: a "public" (i.e., non-secret) server address, registered in the DNS, that is used to accept incoming connection requests from other machines, and (possibly) an "anonymous" address used to shield the identity of the client when it initiates communication. These two cases are roughly analogous to telephone numbers and caller ID, where a user may list their telephone number in the public phone book, but disable the display of its number via caller ID when initiating calls.

To make it difficult to make educated guesses as to whether two different interface identifiers belong to the same node, the algorithm for generating alternate identifiers must include input that has an unpredictable component from the perspective of the outside entities that are collecting information. Picking identifiers from a pseudo-random sequence suffices, so long as the specific sequence cannot be determined by an outsider examining just the identifiers that appear in addresses or are otherwise readily available (e.g., a node's link-layer address). This document proposes the generation of a pseudo-random sequence of interface identifiers via an MD5 hash. Periodically, the next interface identifier in the sequence is generated, a new set of anonymous addresses is created, and the previous anonymous addresses are deprecated to discourage their further use. The precise pseudo-random sequence depends on both a random component and the globally unique interface identifier (when available), to increase the likelihood that different nodes generate different sequences.

3. Protocol Description

The goal of this section is to define procedures that:

1) Do not result in any changes to the basic behavior of addresses generated via stateless address autoconfiguration [ADDRCONF].

2) Define new procedures that create additional global-scope addresses based on a random interface identifier for use with global scope addresses. Such addresses would be used to initiate outgoing sessions. These "random" or anonymous addresses would be used for a short period of time (hours to days) and would then be deprecated. Deprecated address can continue to be used for already established connections, but are not used to initiate new connections. New anonymous addresses are generated periodically to replace anonymous addresses that expire, with the exact time between address generation a matter of local policy.

3) Produce a sequence of anonymous global-scope addresses from a sequence of interface identifiers that appear to be random in the sense that it is difficult for an outside observer to predict a future address (or identifier) based on a current one and it is difficult to determine previous addresses (or identifiers) knowing only the present one.

4) Generate a set of addresses from the same (randomized) interface identifier, one address for each prefix for which a global address has been generated via stateless address autoconfiguration. Using the same interface identifier to generate a set of anonymous addresses reduces the number of IP multicast groups a host must join. Nodes join the solicited-node multicast address for each unicast address they support, and solicited-node addresses are dependent only on the low-order bits of the corresponding address. This decision was made to address the concern that a node that joins a large number of multicast groups may be required to put its interface into promiscuous mode, resulting in possible reduced performance.

3.1. Assumptions

The following algorithm assumes that each interface maintains an associated randomized interface identifier. When anonymous addresses are generated, the current value of the associated randomized interface identifier is used. The actual value of the identifier changes over time as described below, but the same identifier can be used to generate more than one anonymous address.

The algorithm also assumes that for a given anonymous address, one can determine the corresponding public address. When an anonymous address is deprecated, a new anonymous address is generated. The specific valid and preferred lifetimes for the new address are dependent on the corresponding lifetime values in the public address.

Finally, this document assumes that when a node initiates outgoing communication, anonymous addresses can be given preference over other public addresses. This can mean that all outgoing connections use anonymous addresses by default, or that applications individually indicate whether they prefer to use anonymous or public addresses. Giving preference to anonymous address is consistent with on-going work that addresses the topic of source address-selection in the more general case [ADDR SELECT].

3.2. Generation Of Randomized Interface Identifiers.

We describe two approaches for the maintenance of the randomized interface identifier. The first assumes the presence of stable storage that can be used to record state history for use as input into the next iteration of the algorithm across system restarts. A second approach addresses the case where stable storage is unavailable and a randomized interface identifier may need to be generated at random.

3.2.1. When Stable Storage Is Present

The following algorithm assumes the presence of a 64-bit "history value" that is used as input in generating a randomized interface identifier. The very first time the system boots (i.e., out-of-the-box), a random value should be generated using techniques that help ensure the initial value is hard to guess [RANDOM]. Whenever a new interface identifier is generated, a value generated by the computation is saved in the history value for the next iteration of the algorithm.

A randomized interface identifier is created as follows:

1) Take the history value from the previous iteration of this algorithm (or a random value if there is no previous value) and append to it the interface identifier generated as described in [ADDRARCH].
2) Compute the MD5 message digest [MD5] over the quantity created in the previous step.
3) Take the left-most 64-bits of the MD5 digest and set bit 6 (the left-most bit is numbered 0) to zero. This creates an interface identifier with the universal/local bit indicating local significance only. Save the generated identifier as the associated randomized interface identifier.
4) Take the rightmost 64-bits of the MD5 digest computed in step 2) and save them in stable storage as the history value to be used in the next iteration of the algorithm.

MD5 was chosen for convenience, and because its particular properties were adequate to produce the desired level of randomization. IPv6 nodes are already required to implement MD5 as part of IPsec [IPSEC], thus the code will already be present on IPv6 machines.

In theory, generating successive randomized interface identifiers using a history scheme as above has no advantages over generating them at random. In practice, however, generating truly random numbers can be tricky. Use of a history value is intended to avoid the

particular scenario where two nodes generate the same randomized interface identifier, both detect the situation via DAD, but then proceed to generate identical randomized interface identifiers via the same (flawed) random number generation algorithm. The above algorithm avoids this problem by having the interface identifier (which will often be globally unique) used in the calculation that generates subsequent randomized interface identifiers. Thus, if two nodes happen to generate the same randomized interface identifier, they should generate different ones on the followup attempt.

3.2.2. In The Absence of Stable Storage

In the absence of stable storage, no history value will be available across system restarts to generate a pseudo-random sequence of interface identifiers. Consequently, the initial history value used above will need to be generated at random. A number of techniques might be appropriate. Consult [RANDOM] for suggestions on good sources for obtaining random numbers. Note that even though machines may not have stable storage for storing a history value, they will in many cases have configuration information that differs from one machine to another (e.g., user identity, security keys, serial numbers, etc.). One approach to generating a random initial history value in such cases is to use the configuration information to generate some data bits (which may remain constant for the life of the machine, but will vary from one machine to another), append some random data and compute the MD5 digest as before.

3.3. Generating Anonymous Addresses

[ADDRCONF] describes the steps for generating a link-local address

when an interface becomes enabled as well as the steps for generating addresses for other scopes. This document extends [ADDRCONF] as follows. When processing a Router Advertisement with a Prefix Information option carrying a global-scope prefix for the purposes of address autoconfiguration (i.e., the A bit is set), perform the following steps:

1) Process the Prefix Information Option as defined in [ADDRCONF], either creating a public address or adjusting the lifetimes of existing addresses, both public and anonymous. When adjusting the lifetimes of an existing anonymous address, only lower the lifetimes. Implementations MUST NOT increase the lifetimes of an existing anonymous address when processing a Prefix Information Option.
2) When a new public address is created as described in [ADDRCONF] (because the prefix advertised does not match the prefix of any

address already assigned to the interface, and the Valid Lifetime in the option is not zero), also create a new anonymous address.
3) When creating an anonymous address, the lifetime values are derived from the corresponding public address as follows:

   - Its Valid Lifetime is the lower of the Valid Lifetime of the public address or ANON_VALID_LIFETIME.
   - Its Preferred Lifetime is the lower of the Preferred Lifetime of the public address or ANON_PREFERRED_LIFETIME.

   An anonymous address is created only if this calculated Preferred Lifetime is greater than REGEN_ADVANCE time units. In particular, an implementation MUST NOT create an anonymous address with a zero Preferred Lifetime.
4) New anonymous addresses are created by appending the interface's current randomized interface identifier to the prefix that was used to generate the corresponding public address. If by chance the new anonymous address is the same as an address already assigned to the interface, generate a new randomized interface identifier and repeat this step.
5) Perform duplicate address detection (DAD) on the generated anonymous address. If DAD indicates the address is already in use, generate a new randomized interface identifier as described in Section 3.2 above, and repeat the previous steps as appropriate up to 5 times. If after 5 consecutive attempts no non-unique address

was generated, log a system error and give up attempting to generate anonymous addresses for that interface.

Note: because multiple anonymous addresses are generated from the same associated randomized interface identifier, there is little benefit in running DAD on every anonymous address. This document recommends that DAD be run on the first address generated from a given randomized identifier, but that DAD be skipped on all subsequent addresses generated from the same randomized interface identifier.

3.4. Expiration of Anonymous Addresses

When an anonymous address becomes deprecated, a new one should be generated. This is done by repeating the actions described in Section 3.3, starting at step 3). Note that, except for the transient period when an anonymous address is being regenerated, in normal operation at most one anonymous address corresponding to a public address should be in a non-deprecated state at any given time. Note that if an anonymous address becomes deprecated as result of processing a Prefix Information Option with a zero Preferred Lifetime, then a new anonymous address MUST NOT be generated. The Prefix Information

Option will also deprecate the corresponding public address.

To insure that a preferred anonymous address is always available, a new anonymous address should be regenerated slightly before its predecessor is deprecated. This is to allow sufficient time to avoid race conditions in the case where generating a new anonymous address is not instantaneous, such as when duplicate address detection must be run. It is recommended that an implementation start the address regeneration process REGEN_ADVANCE time units before an anonymous address would actually be deprecated.

As an optional optimization, an implementation may wish to remove a deprecated anonymous address that is not in use by applications or upper-layers. For TCP connections, such information is available in control blocks. For UDP-based applications, it may be the case that only the applications have knowledge about what addresses are actually in use. Consequently, one may need to use heuristics in deciding when an address is no longer in use (e.g., the default

ANON_VALID_LIFETIME suggested above).

3.5. Regeneration of Randomized Interface Identifiers

The frequency at which anonymous addresses should change depends on how a device is being used (e.g., how frequently it initiates new communication) and the concerns of the end user. The most egregious privacy concerns appear to involve addresses used for long periods of time (weeks to months to years). The more frequently an address changes, the less feasible collecting or coordinating information keyed on interface identifiers becomes. Moreover, the cost of collecting information and attempting to correlate it based on interface identifiers will only be justified if enough addresses contain non-changing identifiers to make it worthwhile. Thus, having large numbers of clients change their address on a daily or weekly basis is likely to be sufficient to alleviate most privacy concerns.

There are also client costs associated with having a large number of addresses associated with a node (e.g., in doing address lookups, the need to join many multicast groups, etc.). Thus, changing addresses frequently (e.g., every few minutes) may have performance implications.

This document recommends that implementations generate new anonymous addresses on a periodic basis. This can be achieved automatically by generating a new randomized interface identifier at least once every (ANON_PREFERRED_LIFETIME - REGEN_ADVANCE) time units. As described above, generating a new anonymous address REGEN_ADVANCE time units before an anonymous address becomes deprecated produces addresses

with a preferred lifetime no larger than ANON_PREFERRED_LIFETIME. When the preferred lifetime expires, a new anonymous address is generated using the new randomized interface identifier.

Because the precise frequency at which it is appropriate to generate new addresses varies from one environment to another, implementations should provide end users with the ability to change the frequency at which addresses are regenerated. The default value is given in ANON_PREFERRED_LIFETIME and is one day. In addition, the exact time at which to invalidate an anonymous address depends on how applications are used by end users. Thus the default value given of

one week (ANON_PREFERRED_LIFETIME) may not be appropriate in all environments. Implementations should provide end users with the ability to override both of these default values.

4. Implications of Changing Interface Identifiers

The IPv6 addressing architecture goes to great lengths to ensure that interface identifiers are globally unique. During the IPng discussions of the GSE proposal [GSE], it was felt that keeping interface identifiers globally unique in practice might prove useful to future transport protocols. Usage of the algorithms in this document would eliminate that future flexibility.

The desires of protecting individual privacy vs. the desire to effectively maintain and debug a network can conflict with each other. Having clients use addresses that change over time will make it more difficult to track down and isolate operational problems. For example, when looking at packet traces, it could become more difficult to determine whether one is seeing behavior caused by a single errant machine, or by a number of them.

Some servers refuse to grant access to clients for which no DNS name exists. That is, they perform a DNS PTR query to determine the DNS name, and may then also perform an A query on the returned name to verify that the returned DNS name maps back into the address being used. Consequently, clients not properly registered in the DNS may be unable to access some services. As noted earlier, however, a node's DNS name (if non-changing) serves as a constant identifier. If the extension described in this document becomes widely deployed, servers will likely need to change their behavior to not require every address be in the DNS. Another alternative is to register anonymous addresses in DNS using random names (for example a string version of the address itself).

5. Defined Constants

Constants defined in this document include:

ANON_VALID_LIFETIME -- Default value: 1 week. Users should be able to override the default value.
ANON_PREFERRED_LIFETIME -- Default value: 1 day. Users should be able to override the default value.
REGEN_ADVANCE -- 5 seconds

6. Open Issues and Future Work

An implementation might want to keep track of which addresses are being used by upper layers so as to be able to remove a deprecated anonymous address from internal data structures once no upper layer protocols are using it (but not before). This is in contrast to current approaches where addresses are removed from an interface when they become invalid [ADDRCONF], independent of whether or not upper layer protocols are still using them. For TCP connections, such information is available in control blocks. For UDP-based applications, it may be the case that only the applications have knowledge about what addresses are actually in use. Consequently, it may need to use heuristics in deciding when an address is no longer in use (e.g., as is suggested in Section 3.4).

Use of the extensions defined in this document is likely to make debugging and other operational troubleshooting activities more difficult. Consequently, it may be site policy that anonymous addresses should not be used. Implementations MAY provide a method for a trusted administrator to override the use of anonymous addresses.

7. Security Considerations

The motivation for this document stems from privacy concerns for individuals. This document does not appear to add any security issues beyond those already associated with stateless address autoconfiguration [ADDRCONF].

8. Acknowledgments

The authors would like to acknowledge the contributions of the IPNGWG working group and, in particular, Matt Crawford and Steve Deering for their detailed comments.

9. References

[ADDRARCH] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 2373, July 1998.

[ADDRCONF] Thomson, S. and T. Narten, "IPv6 Address Autoconfiguration", RFC 2462, December 1998.

[ADDR_SELECT] Draves, R. "Default Address Selection for IPv6", draft-ietf-ipngwg-default-addr-select-00.txt.

[COOKIES] Kristol, D., Montulli, L., "HTTP State Management Mechanism", draft-ietf-http-state-man-mec-12.txt.

[DHCP] Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, March 1997.

[DDNS] Vixie et. al., "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, April 1997.

[DISCOVERY] Narten, T., Nordmark, E. and W. Simpson, "Neighbor Discovery for IP Version 6 (IPv6)", RFC 2461, December 1998.

[GSE] Crawford et. al., "Separating Identifiers and Locators in Addresses: An Analysis of the GSE Proposal for IPv6 ", draft-ietf-ipngwg-esd-analysis-04.txt.

[IPSEC] Kent, S., Atkinson, R., "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

[KEYWORDS] Bradner,S. "Key words for use in RFCs to Indicate Requirement Levels" RFC 2119, March 1997.

[MD5] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[MOBILEIP] Perkins, C., "IP Mobility Support", RFC 2002, October 1996.

[RANDOM] "Randomness Recommendations for Security", Eastlake 3rd, D., Crocker S., Schiller, J., RFC 1750, December 1994.

[SERIALNUM] Moore, K., "Privacy Considerations for the Use of Hardware Serial Numbers in End-to-End Network Protocols", draft-iesg-serno-privacy-00.txt.

10. Authors' Addresses

Thomas Narten
IBM Corporation
P.O. Box 12195
Research Triangle Park, NC 27709-2195
USA

Phone: +1 919 254 7798
EMail: narten@raleigh.ibm.com

Richard Draves
Microsoft Research
One Microsoft Way
Redmond, WA 98052

Phone: +1 425 936 2268
Email: richdr@microsoft.com