Network Working Group
Request for Comments: 4367
Category: Informational

J. Rosenberg, Ed.
IAB

February 2006

What's in a Name: False Assumptions about DNS Names

Status of This Memo

Copyright Notice

Abstract

The Domain Name System (DNS) provides an essential service on the Internet, mapping structured names to a variety of data, usually IP addresses. These names appear in email addresses, Uniform Resource Identifiers (URIs), and other application-layer identifiers that are often rendered to human users. Because of this, there has been a strong demand to acquire names that have significance to people, through equivalence to registered trademarks, company names, types of services, and so on. There is a danger in this trend; the humans and automata that consume and use such names will associate specific semantics with some names and thereby make assumptions about the services that are, or should be, provided by the hosts associated with the names. Those assumptions can often be false, resulting in a variety of failure conditions. This document discusses this problem in more detail and makes recommendations on how it can be avoided.

Table of Contents

1. Introduction

The Domain Name System (DNS) [1] provides an essential service on the Internet, mapping structured names to a variety of different types of data. Most often it is used to obtain the IP address of a host associated with that name [2] [1] [3]. However, it can be used to obtain other information, and proposals have been made for nearly everything, including geographic information [4].

Domain names are most often used in identifiers used by application protocols. The most well known include email addresses and URIs, such as the HTTP URL [5], Real Time Streaming Protocol (RTSP) URL [6], and SIP URI [7]. These identifiers are ubiquitous, appearing on business cards, web pages, street signs, and so on. Because of this, there has been a strong demand to acquire domain names that have significance to people through equivalence to registered trademarks, company names, types of services, and so on. Such identifiers serve many business purposes, including extension of brand, advertising, and so on.

People often make assumptions about the type of service that is or should be provided by a host associated with that name, based on their expectations and understanding of what the name implies. This, in turn, triggers attempts by organizations to register domain names based on that presumed user expectation. Examples of this are the

various proposals for a Top-Level Domain (TLD) that could be associated with adult content [8], the requests for creation of TLDs associated with mobile devices and services, and even phishing attacks.

When these assumptions are codified into the behavior of an automaton, such as an application client or server, as a result of implementor choice, management directive, or domain owner policy, the overall system can fail in various ways. This document describes a number of typical ways in which these assumptions can be codified, how they can be wrong, the consequences of those mistakes, and the recommended ways in which they can be avoided.

Section 4 describes some of the possible assumptions that clients, servers, and people can make about a domain name. In this context, an "assumption" is defined as any behavior that is expected when accessing a service at a domain name, even though the behavior is not explicitly codified in protocol specifications. Frequently, these assumptions involve ignoring parts of a specification based on an assumption that the client or server is deployed in an environment that is more rigid than the specification allows. Section 5 overviews some of the consequences of these false assumptions. Generally speaking, these consequences can include a variety of different interoperability failures, user experience failures, and system failures. Section 6 discusses why these assumptions can be false from the very beginning or become false at some point in the future. Most commonly, they become false because the environment changes in unexpected ways over time, and what was a valid assumption before, no longer is. Other times, the assumptions prove wrong because they were based on the belief that a specific community of clients and servers was participating, and an element outside of that community began participating.

Section 7 then provides some recommendations. These recommendations encapsulate some of the engineering mantras that have been at the root of Internet protocol design for decades. These include:

> Follow the specifications.
>
> Use the capability negotiation techniques provided in the protocols.
>
> Be liberal in what you accept, and conservative in what you send. [18]

Overall, automata should not change their behavior within a protocol based on the domain name, or some component of the domain name, of the host they are communicating with.

2. Target Audience

This document has several audiences. Firstly, it is aimed at implementors who ultimately develop the software that make the false assumptions that are the subject of this document. The recommendations described here are meant to reinforce the engineering guidelines that are often understood by implementors, but frequently forgotten as deadlines near and pressures mount.

The document is also aimed at technology managers, who often develop the requirements that lead to these false assumptions. For them, this document serves as a vehicle for emphasizing the importance of not taking shortcuts in the scope of applicability of a project.

Finally, this document is aimed at domain name policy makers and administrators. For them, it points out the perils in establishing domain policies that get codified into the operation of applications running within that domain.

3. Modeling Usage of the DNS

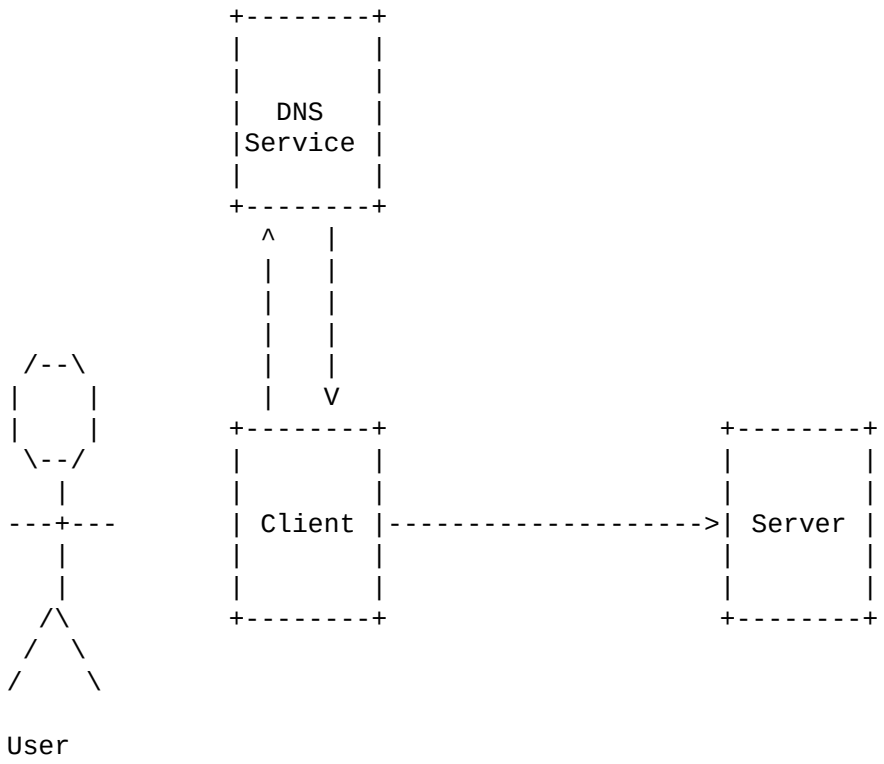

Figure 1

Figure 1 shows a simple conceptual model of how the DNS is used by applications. A user of the application obtains an identifier for particular content or service it wishes to obtain. This identifier is often a URL or URI that contains a domain name. The user enters this identifier into its client application (for example, by typing in the URL in a web browser window). The client is the automaton (a software and/or hardware system) that contacts a server for that application in order to provide service to the user. To do that, it contacts a DNS server to resolve the domain name in the identifier to an IP address. It then contacts the server at that IP address. This simple model applies to application protocols such as HTTP [5], SIP [7], RTSP [6], and SMTP [9].

>From this model, it is clear that three entities in the system can potentially make false assumptions about the service provided by the server. The human user may form expectations relating to the content of the service based on a parsing of the host name from which the content originated. The server might assume that the client connecting to it supports protocols that it does not, can process content that it cannot, or has capabilities that it does not. Similarly, the client might assume that the server supports protocols, content, or capabilities that it does not. Furthermore, applications can potentially contain a multiplicity of humans, clients, and servers, all of which can independently make these false assumptions.

4. Possible Assumptions

For each of the three elements, there are many types of false assumptions that can be made.

4.1. By the User

The set of possible assumptions here is nearly boundless. Users might assume that an HTTP URL that looks like a company name maps to a server run by that company. They might assume that an email from a email address in the .gov TLD is actually from a government employee. They might assume that the content obtained from a web server within a TLD labeled as containing adult materials (for example, .sex) actually contains adult content [8]. These assumptions are unavoidable, may all be false, and are not the focus of this document.

4.2. By the Client

Even though the client is an automaton, it can make some of the same assumptions that a human user might make. For example, many clients assume that any host with a hostname that begins with "www" is a web server, even though this assumption may be false.

In addition, the client concerns itself with the protocols needed to communicate with the server. As a result, it might make assumptions about the operation of the protocols for communicating with the server. These assumptions manifest themselves in an implementation when a standardized protocol negotiation technique defined by the protocol is ignored, and instead, some kind of rule is coded into the software that comes to its own conclusion about what the negotiation would have determined. The result is often a loss of interoperability, degradation in reliability, and worsening of user experience.

Authentication Algorithm: Though a protocol might support a multiplicity of authentication techniques, a client might assume that a server always supports one that is only optional according to the protocol. For example, a SIP client contacting a SIP server in a domain that is apparently used to identify mobile devices (for example, www.example.cellular) might assume that the server supports the optional Authentication and Key Agreement (AKA) digest technique [10], just because of the domain name that was used to access the server. As another example, a web client might assume that a server with the name https.example.com supports HTTP over Transport Layer Security (TLS) [16].

Data Formats: Though a protocol might allow a multiplicity of data formats to be sent from the server to the client, the client might assume a specific one, rather than using the content labeling and negotiation capabilities of the underlying protocol. For example, an RTSP client might assume that all audio content delivered to it from media.example.cellular uses a low-bandwidth codec. As another example, a mail client might assume that the contents of messages it retrieves from a mail server at mail.example.cellular are always text, instead of checking the MIME headers [11] in the message in order to determine the actual content type.

Protocol Extensions: A client may attempt an operation on the server that requires the server to support an optional protocol extension. However, rather than implementing the necessary fallback logic, the client may falsely assume that the extension is supported. As an example, a SIP client that requires reliable provisional responses to its request (RFC 3262 [17]) might assume that this extension is supported on servers in the domain

sip.example.telecom. Furthermore, the client would not implement the fallback behavior defined in RFC 3262, since it would assume that all servers it will communicate with are in this domain and that all therefore support this extension. However, if the assumptions prove wrong, the client is unable to make any phone calls.

Languages: A client may support facilities for processing text content differently depending on the language of the text. Rather than determining the language from markers in the message from the server, the client might assume a language based on the domain name. This assumption can easily be wrong. For example, a client might assume that any text in a web page retrieved from a server within the .de country code TLD (ccTLD) is in German, and attempt a translation to Finnish. This would fail dramatically if the text was actually in French. Unfortunately, this client behavior is sometimes exhibited because the server has not properly labeled the language of the content in the first place, often because the server assumed such a labeling was not needed. This is an example of how these false assumptions can create vicious cycles.

4.3. By the Server

The server, like the client, is an automaton. Let us consider one servicing a particular domain -- www.company.cellular, for example. It might assume that all clients connecting to this domain support particular capabilities, rather than using the underlying protocol to make this determination. Some examples include:

Authentication Algorithm: The server can assume that a client supports a particular, optional, authentication technique, and it therefore does not support the mandatory one.

Language: The server can serve content in a particular language, based on an assumption that clients accessing the domain speak a particular language, or based on an assumption that clients coming from a particular IP address speak a certain language.

Data Formats: The server can assume that the client supports a particular set of MIME types and is only capable of sending ones within that set. When it generates content in a protocol response, it ignores any content negotiation headers that were present in the request. For example, a web server might ignore the Accept HTTP header field and send a specific image format.

Protocol Extensions: The server might assume that the client supports a particular optional protocol extension, and so it does not support the fallback behavior necessary in the case where the client does not.

Client Characteristics: The server might assume certain things about the physical characteristics of its clients, such as memory footprint, processing power, screen sizes, screen colors, pointing devices, and so on. Based on these assumptions, it might choose specific behaviors when processing a request. For example, a web server might always assume that clients connect through cell phones, and therefore return content that lacks images and is tuned for such devices.

5. Consequences of False Assumptions

There are numerous negative outcomes that can arise from the various false assumptions that users, servers, and clients can make. These include:

Interoperability Failure: In these cases, the client or server assumed some kind of protocol operation, and this assumption was wrong. The result is that the two are unable to communicate, and the user receives some kind of an error. This represents a total interoperability failure, manifesting itself as a lack of service to users of the system. Unfortunately, this kind of failure persists. Repeated attempts over time by the client to access the service will fail. Only a change in the server or client software can fix this problem.

System Failure: In these cases, the client or server misinterpreted a protocol operation, and this misinterpretation was serious enough to uncover a bug in the implementation. The bug causes a system crash or some kind of outage, either transient or permanent (until user reset). If this failure occurs in a server, not only will the connecting client lose service, but other clients attempting to connect will not get service. As an example, if a web server assumes that content passed to it from a client (created, for example, by a digital camera) is of a particular content type, and it always passes image content to a codec for decompression prior to storage, the codec might crash when it unexpectedly receives an image compressed in a different format. Of course, it might crash even if the Content-Type was correct, but the compressed bitstream was invalid. False assumptions merely introduce additional failure cases.

Poor User Experience: In these cases, the client and server communicate, but the user receives a diminished user experience. For example, if a client on a PC connects to a web site that provides content for mobile devices, the content may be underwhelming when viewed on the PC. Or, a client accessing a streaming media service may receive content of very low bitrate, even though the client supported better codecs. Indeed, if a user wishes to access content from both a cellular device and a PC using a shared address book (that is, an address book shared across multiple devices), the user would need two entries in that address book, and would need to use the right one from the right device. This is a poor user experience.

Degraded Security: In these cases, a weaker security mechanism is used than the one that ought to have been used. As an example, a server in a domain might assume that it is only contacted by clients with a limited set of authentication algorithms, even though the clients have been recently upgraded to support a stronger set.

6. Reasons Why the Assumptions Can Be False

Assumptions made by clients and servers about the operation of protocols when contacting a particular domain are brittle, and can be wrong for many reasons. On the server side, many of the assumptions are based on the notion that a domain name will only be given to, or used by, a restricted set of clients. If the holder of the domain name assumes something about those clients, and can assume that only those clients use the domain name, then it can configure or program the server to operate specifically for those clients. Both parts of this assumption can be wrong, as discussed in more detail below.

On the client side, the notion is similar, being based on the assumption that a server within a particular domain will provide a specific type of service. Sub-delegation and evolution, both discussed below, can make these assumptions wrong.

6.1. Evolution

The Internet and the devices that access it are constantly evolving, often at a rapid pace. Unfortunately, there is a tendency to build for the here and now, and then worry about the future at a later time. Many of the assumptions above are predicated on characteristics of today's clients and servers. Support for specific protocols, authentication techniques, or content are based on today's standards and today's devices. Even though they may, for the most part, be true, they won't always be. An excellent example is mobile devices. A server servicing a domain accessed by mobile devices

might try to make assumptions about the protocols, protocol extensions, security mechanisms, screen sizes, or processor power of such devices. However, all of these characteristics can and will change over time.

When they do change, the change is usually evolutionary. The result is that the assumptions remain valid in some cases, but not in others. It is difficult to fix such systems, since it requires the server to detect what type of client is connecting, and what its capabilities are. Unless the system is built and deployed with these capability negotiation techniques built in to begin with, such detection can be extremely difficult. In fact, fixing it will often require the addition of such capability negotiation features that, if they had been in place and used to begin with, would have avoided the problem altogether.

6.2. Leakage

Servers also make assumptions because of the belief that they will only be accessed by specific clients, and in particular, those that are configured or provisioned to use the domain name. In essence, there is an assumption of community -- that a specific community knows and uses the domain name, while others outside of the community do not.

The problem is that this notion of community is a false one. The Internet is global. The DNS is global. There is no technical barrier that separates those inside of the community from those outside. The ease with which information propagates across the Internet makes it extremely likely that such domain names will eventually find their way into clients outside of the presumed community. The ubiquitous presence of domain names in various URI formats, coupled with the ease of conveyance of URIs, makes such leakage merely a matter of time. Furthermore, since the DNS is global, and since it can only have one root [12], it becomes possible for clients outside of the community to search and find and use such "special" domain names.

Indeed, this leakage is a strength of the Internet architecture, not a weakness. It enables global access to services from any client with a connection to the Internet. That, in turn, allows for rapid growth in the number of customers for any particular service.

6.3. Sub-Delegation

Clients and users make assumptions about domains because of the notion that there is some kind of centralized control that can enforce those assumptions. However, the DNS is not centralized; it

is distributed. If a domain doesn't delegate its sub-domains and has its records within a single zone, it is possible to maintain a centralized policy about operation of its domain. However, once a domain gets sufficiently large that the domain administrators begin to delegate sub-domains to other authorities, it becomes increasingly difficult to maintain any kind of central control on the nature of the service provided in each sub-domain.

Similarly, the usage of domain names with human semantic connotation tends to lead to a registration of multiple domains in which a particular service is to run. As an example, a service provider with the name "example" might register and set up its services in "example.com", "example.net", and generally example.foo for each foo that is a valid TLD. This, like sub-delegation, results in a growth in the number of domains over which it is difficult to maintain centralized control.

Not that it is not possible, since there are many examples of successful administration of policies across sub-domains many levels deep. However, it takes an increasing amount of effort to ensure this result, as it requires human intervention and the creation of process and procedure. Automated validation of adherence to policies is very difficult to do, as there is no way to automatically verify many policies that might be put into place.

A less costly process for providing centralized management of policies is to just hope that any centralized policies are being followed, and then wait for complaints or perform random audits. Those approaches have many problems.

The invalidation of assumptions due to sub-delegation is discussed in further detail in Section 4.1.3 of [8] and in Section 3.3 of [20].

As a result of the fragility of policy continuity across sub-delegations, if a client or user assumes some kind of property associated with a TLD (such as ".wifi"), it becomes increasingly more likely with the number of sub-domains that this property will not exist in a server identified by a particular name. For example, in "store.chain.company.provider.wifi", there may be four levels of delegation from ".wifi", making it quite likely that, unless the holder of ".wifi" is working diligently, the properties that the holder of ".wifi" wishes to enforce are not present. These properties may not be present due to human error or due to a willful decision not to adhere to them.

6.4. Mobility

One of the primary value propositions of a hostname as an identifier is its persistence. A client can change IP addresses, yet still retain a persistent identifier used by other hosts to reach it. Because their value derives from their persistence, hostnames tend to move with a host not just as it changes IP addresses, but as it changes access network providers and technologies. For this reason, assumptions made about a host based on the presumed access network corresponding to that hostname tend to be wrong over time. As an example, a PC might normally be connected to its broadband provider, and through dynamic DNS have a hostname within the domain of that provider. However, one cannot assume that any host within that network has access over a broadband link; the user could connect their PC over a low-bandwidth wireless access network and still retain its domain name.

6.5. Human Error

Of course, human error can be the source of errors in any system, and the same is true here. There are many examples relevant to the problem under discussion.

A client implementation may make the assumption that, just because a DNS SRV record exists for a particular protocol in a particular domain, indicating that the service is available on some port, that the service is, in fact, running there. This assumption could be wrong because the SRV records haven't been updated by the system administrators to reflect the services currently running. As another example, a client might assume that a particular domain policy applies to all sub-domains. However, a system administrator might have omitted to apply the policy to servers running in one of those sub-domains.

7. Recommendations

Based on these problems, the clear conclusion is that clients, servers, and users should not make assumptions on the nature of the service provided to, or by, a domain. More specifically, however, the following can be said:

Follow the specifications: When specifications define mandatory baseline procedures and formats, those should be implemented and supported, even if the expectation is that optional procedures will most often be used. For example, if a specification mandates a particular baseline authentication technique, but allows others to be negotiated and used, implementations need to implement the baseline authentication algorithm even if the other ones are used

most of the time. Put more simply, the behavior of the protocol machinery should never change based on the domain name of the host.

Use capability negotiation: Many protocols are engineered with capability negotiation mechanisms. For example, a content negotiation framework has been defined for protocols using MIME content [13] [14] [15]. SIP allows for clients to negotiate the media types used in the multimedia session, as well as protocol parameters. HTTP allows for clients to negotiate the media types returned in requests for content. When such features are available in a protocol, client and servers should make use of them rather than making assumptions about supported capabilities. A corollary is that protocol designers should include such mechanisms when evolution is expected in the usage of the protocol.

"Be liberal in what you accept, and conservative in what you send" [18]: This axiom of Internet protocol design is applicable here as well. Implementations should be prepared for the full breadth of what a protocol allows another entity to send, rather than be limiting in what it is willing to receive.

To summarize -- there is never a need to make assumptions. Rather than doing so, utilize the specifications and the negotiation capabilities they provide, and the overall system will be robust and interoperable.

8. A Note on RFC 2219 and RFC 2782

Based on the definition of an assumption given here, the behavior hinted at by records in the DNS also represents an assumption. RFC 2219 [19] defines well-known aliases that can be used to construct domain names for reaching various well-known services in a domain. This approach was later followed by the definition of a new resource record, the SRV record [2], which specifies that a particular service is running on a server in a domain. Although both of these mechanisms are useful as a hint that a particular service is running in a domain, both of them represent assumptions that may be false. However, they differ in the set of reasons why those assumptions might be false.

A client that assumes that "ftp.example.com" is an FTP server may be wrong because the presumed naming convention in RFC 2219 was not known by, or not followed by, the owner of domain.com. With RFC 2782, an SRV record for a particular service would be present only by explicit choice of the domain administrator, and thus a client that

assumes that the corresponding host provides this service would be wrong only because of human error in configuration. In this case, the assumption is less likely to be wrong, but it certainly can be.

The only way to determine with certainty that a service is running on a host is to initiate a connection to the port for that service, and check. Implementations need to be careful not to codify any behaviors that cause failures should the information provided in the record actually be false. This borders on common sense for robust implementations, but it is valuable to raise this point explicitly.

9. Security Considerations

One of the assumptions that can be made by clients or servers is the availability and usage (or lack thereof) of certain security protocols and algorithms. For example, a client accessing a service in a particular domain might assume a specific authentication algorithm or hash function in the application protocol. It is possible that, over time, weaknesses are found in such a technique, requiring usage of a different mechanism. Similarly, a system might start with an insecure mechanism, and then decide later on to use a secure one. In either case, assumptions made on security properties can result in interoperability failures, or worse yet, providing service in an insecure way, even though the client asked for, and thought it would get, secure service. These kinds of assumptions are fundamentally unsound even if the records themselves are secured with DNSSEC.

10. Acknowledgements

The IAB would like to thank John Klensin, Keith Moore and Peter Koch for their comments.

11. IAB Members

Internet Architecture Board members at the time of writing of this document are:

Bernard Aboba

Loa Andersson

Brian Carpenter

Leslie Daigle

Patrik Faltstrom

Bob Hinden

Kurtis Lindqvist

David Meyer

Pekka Nikander

Eric Rescorla

Pete Resnick

Jonathan Rosenberg

12. Informative References

[1] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.

[2] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.

[3] Mealling, M., "Dynamic Delegation Discovery System (DDDS) Part Three: The Domain Name System (DNS) Database", RFC 3403, October 2002.

[4] Davis, C., Vixie, P., Goodwin, T., and I. Dickinson, "A Means for Expressing Location Information in the Domain Name System", RFC 1876, January 1996.

[5] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[6] Schulzrinne, H., Rao, A., and R. Lanphier, "Real Time Streaming Protocol (RTSP)", RFC 2326, April 1998.

[7] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[8] Eastlake, D., ".sex Considered Dangerous", RFC 3675, February 2004.

[9] Klensin, J., "Simple Mail Transfer Protocol", RFC 2821, April 2001.

[10] Niemi, A., Arkko, J., and V. Torvinen, "Hypertext Transfer Protocol (HTTP) Digest Authentication Using Authentication and Key Agreement (AKA)", RFC 3310, September 2002.

[11] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[12] Internet Architecture Board, "IAB Technical Comment on the Unique DNS Root", RFC 2826, May 2000.

[13] Klyne, G., "Indicating Media Features for MIME Content", RFC 2912, September 2000.

[14] Klyne, G., "A Syntax for Describing Media Feature Sets", RFC 2533, March 1999.

[15] Klyne, G., "Protocol-independent Content Negotiation Framework", RFC 2703, September 1999.

[16] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[17] Rosenberg, J. and H. Schulzrinne, "Reliability of Provisional Responses in Session Initiation Protocol (SIP)", RFC 3262, June 2002.

[18] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.

[19] Hamilton, M. and R. Wright, "Use of DNS Aliases for Network Services", BCP 17, RFC 2219, October 1997.

[20] Faltstrom, P., "Design Choices When Expanding DNS", Work in Progress, June 2005.

Author's Address

Jonathan Rosenberg, Editor
IAB
600 Lanidex Plaza
Parsippany, NJ 07054
US

Phone: +1 973 952-5000
EMail: jdrosen@cisco.com
URI: http://www.jdrosen.net

Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).