dnsop J. Woodworth
Internet-Draft D. Ballew
Updates: 2308, 2535 (if approved) S. Bindinganaveli Raghavan
Intended status: Standards Track CenturyLink, Inc.
Expires: December 31, 2015 June 30, 2015

BULK DNS Resource Records
draft-woodworth-bulk-rr-00

Abstract

The BULK DNS resource record type defines a method of pattern based creation of DNS resource records to be used in place of NXDOMAIN errors which would normally be returned. These records are currently restricted to registered DNS resource record types A, AAAA, PTR and CNAME. The key benefit of the BULK resource record type is the simplification of maintaining "generic" record assignments which would otherwise be too many to manage or require scripts or proprietary methods as bind's $GENERATE.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 3
1.1. Background and Related Documents 4
1.2. Reserved Words 4
2. The BULK Resource Record 4
2.1. BULK OPTIONAL Hidden Wildcards 4
2.2. BULK RDATA Wire Format 5
2.2.1. The Match Type Field 5
2.2.2. The Label Pattern Field 5
2.2.2.1. Single hyphen 5
2.2.2.2. Numeric ranges 6
2.2.2.3. String values 6
2.2.3. The Replacement Pattern Field 6
2.3. The BULK RR Presentation Format 6
2.4. BULK RR Examples 7
3. BULK Replacement 7
3.1. Matching BULK "owner" field 8
3.2. Matching the BULK "Match Type" field 8
3.3. Matching the BULK "Label Pattern" field 8
3.3.1. Automatic Label Pattern matching 8
3.3.2. Manual Label Pattern matching 9
3.3.2.1. Manual Label Pattern matching examples . . . 9
3.4. Record Generation using the BULK "Replacement Pattern" field 11
3.4.1. Replacement Pattern Backreferences 11
3.4.1.1. Backreference Notation 11
3.4.1.1.1. Simple numeric backreference replacement 11
3.4.1.1.2. Star backreference replacement 12
3.4.1.1.3. Numeric range backreference replacement 12
3.4.1.1.4. Numeric set backreference replacement 12
3.4.1.1.5. Backreference delimiter 12
3.4.1.1.6. Backreference delimiter interval 13
3.4.1.1.7. Backreference padding length 13
3.4.1.1.8. Backreference Position 13
3.4.1.1.9. Backreference Position Negation 14
3.4.2. Replacement Pattern examples 14
4. The NPN Resource Record 16
4.1. NPN RDATA Wire Format 16
4.1.1. The Match Type field 16
4.1.2. The Flags field 16
4.1.3. The Owner Ignore field 17
4.1.4. The Left Ignore field 17
4.1.5. The Right Ignore field 17
4.2. The NPN RR Presentation Format 17
4.3. Normalization Processing of NPN RRs 18
4.3.1. Pseudocode for NPN Normalization Processing . . . 19
4.3.2. NPN Normalization Processing examples 19

5. Positive Side-Effects 23
5.1. Record Superimposition 23
5.2. Pattern Based DNSSEC support 24
6. Known Limitations 24
6.1. Increased CPU utilization for NXDOMAIN RRs 24
6.2. Pre-Adoption Nameserver Implications 24
7. Security Considerations 25
7.1. DNSSEC Signature Strategies 25
7.1.1. On-the-fly (Live) Signatures 25
7.1.2. Normalized (NPN Based) Signatures 25
7.1.3. Non-DNSSEC Zone Support Only 26
7.2. DNSSEC Verifier Details 26
7.3. DDOS Attack Vectors and Mitigation 26
8. IANA Considerations 26
9. Acknowledgements 26
10. References 26
10.1. Normative References 26
10.2. Informative References 27

1. Introduction

The BULK DNS Resource Record (BULK) defines a maskable pattern based method for real-time on-the-fly resource record generation. Specifically, it allows one to manage large blocks of DNS records based entirely on record owner data in the RR query and patterns (or templates) designed by knowledgeable zone administrators. Existing DNS resource records covered by this document are Address (A), IPv6 Address (AAAA), Pointer (PTR) and Canonical Name (CNAME). Although other RR types are not explicitly forbidden from use with BULK logic they fall outside of scope and will not be discussed in this document. This document defines the purpose of this new resource record (BULK), its RDATA format, its presentation format (ASCII representation) as well as generated responses to matched DNS queries.

Two Key benefits of this record type are; a) the ability to transfer BULK RR intentions from primary to secondary nameservers with minimal bandwidth and memory requirements; and b) the ability to manage large volumes of pattern based records such as an IPv6 /64 CIDR or larger in a single entry.
Support options for DNSSEC related complications resulting from BULK generated records are also provided in this document. One such option is in the form of the Numeric Pattern Normalization (NPN) record type described in further detail in this document.

1.1. Background and Related Documents

This document assumes the reader is familiar with the basic DNS concepts described in [RFC1034], [RFC1035], and the subsequent documents that update them, particularly [RFC2181] and [RFC2308].

The reader is also assumed to be familiar with DNSSEC basics as described in [RFC4033], [RFC4034] and [RFC4035] as well as the DNS cryptographic signature generation process described in [RFC2535], [RFC2536], [RFC2931] and [RFC3110].

1.2. Reserved Words

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 [RFC2119].

2. The BULK Resource Record

The BULK resource record consists of details which enable a DNS nameserver to generate RRs of other types based upon query received and patterns provided. Unless otherwise stated the letters used in hexadecimal numbers (a-f) MUST be case insensitive and are assumed to be lowercase. All examples in this document using hexadecimal are provided in lowercase.

The Type value for the BULK RR type is XX.

The BULK RR is class independent.

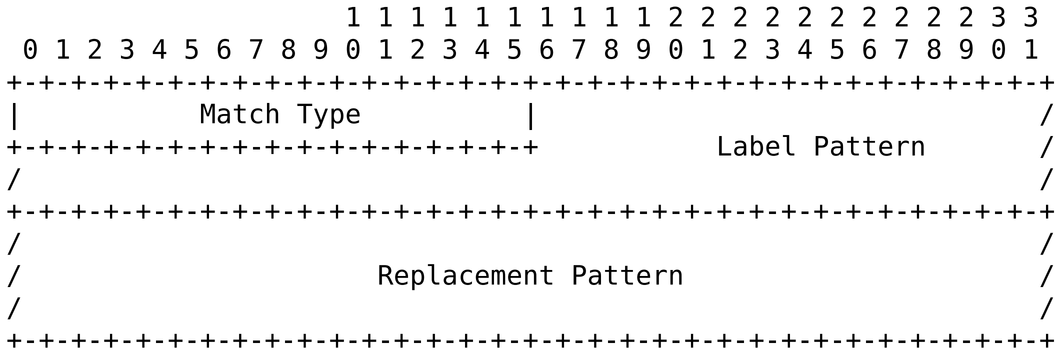
The BULK RR has no special TTL requirements but some security guidelines are offered in a later section.

2.1. BULK OPTIONAL Hidden Wildcards

The BULK RR extends current wildcard substitution logic as defined in [RFC1034] by allowing a single hyphen "-" in the leftmost label to represent the intent of leveraging a modified wildcard matching mechanism. If this condition exists wildcard logic SHALL be used for generated replacement records but not for the BULK resource records themselves. This will become clearer in the "BULK Replacement" section of this document. If an asterisk "*" (the standard wildcard character) is used default wildcard behavior MUST be used.

2.2. BULK RDATA Wire Format

The RDATA for a BULK RR consists of a 2 octet Match Type Field, a Label Pattern Field and the Replacement Pattern Field.



2.2.1. The Match Type Field

The Match Type field identifies the type of the RRset identified by this BULK record.

2.2.2. The Label Pattern Field

The Label Pattern Field consists of a text string which may be evaluated by the sections below. The character encoding for this field is US-ASCII and may not contain whitespace unless enclosed within double-quote characters. The value of a single hyphen "-" has special implications and will be discussed in greater detail below.

```
<pattern> ::=   "-" | {"} <part> <part>* {"}
<part>    ::=   <range> | <string>
<range>   ::=   "[" <numbers> { "-" <numbers> } "]"
<numbers> ::=   <number> <number>*
<number>  ::=   0 ... 9 | 0 ... f
```

Label Patterns MUST NOT contain square braces "[" or "]" which are outside a numeric range as described in the following sections.

2.2.2.1. Single hyphen

If the label pattern field consists of a single hyphen it is not necessary to evaluate for numeric ranges or strings. Implementors SHOULD simply set a flag indicating all ranges matching the query's label are true and backreferences (described in further detail in the "BULK Replacement" section) will be automatically set.

2.2.2.2. Numeric ranges

Numeric ranges include decimal or hexadecimal ranges depending on which record type was used in the query. This logic will be described in further detail in the "Replacement Logic" section.

The numeric range pattern will be a range of allowed numbers lower and upper values separated by a single hyphen "-". If upper and lower values are identical a single numeric value (without hyphen) will suffice. To easily distinguish numeric range patterns from string values they MUST be enclosed within square braces "[" and "]".

2.2.2.3. String values

All values found before or after Numeric ranges (excluding single-hyphen rule) are considered to be string values. These values will be taken literally when evaluating for pattern matches in the "BULK Replacement" section below.

2.2.3. The Replacement Pattern Field

The Replacement Pattern field describes how the answer RRset SHOULD be generated for the matching query. It can either be a single hyphen "-" or a string containing backreferences (described in further detail in the "BULK Replacement" section). This field MUST be evaluated for proper syntax for resource records of its Match Type defined above. A "read" evaluation MAY be performed when a zone is first committed to memory either while converting from Text to Wire format (from stored zone files) or when a RR transfer is received (raw Wire format). Stage two "write" evaluations MUST be performed prior to returning generated replacement answers. Since logic to perform a stage two evaluation is already a requirement for DNS nameservers it may be easier for implementors to perform just stage two evaluations. Stage-two-only evaluation may be also preferred for performance purposes and is acceptable behavior. Any stage two evaluation errors MUST be processed as if the record did not exist and if all BULK generated records for a query answer-set evaluate to errors the original condition of an NXDOMAIN error state MUST be restored.

2.3. The BULK RR Presentation Format

The Match Type field is represented as an RR type mnemonic. When the mnemonic is not known, the TYPE representation as described in [RFC3597], Section 5, MUST be used.

The Label Pattern and Replacement Pattern fields MUST be presented as the TXT RR type described in [RFC1035], Section 3.3.14.

2.4. BULK RR Examples

EXAMPLE 1
The following BULK RR stores a block of A RRs for example.com.

```
*.example.com. 86400 IN BULK A (
                                 pool-A-[0-255]-[0-255].example.com.
                                 10.55.${1}.${2}
                               )
```

The first four fields specify the owner name, TTL, Class, and RR type (BULK). Value "A" indicates that this BULK RR defines the A record type (Address). Value "pool-A-[0-255]-[0-255].example.com." indicates the Label Pattern. Value "10.55.${1}.${2}" indicates the Replacement Pattern. The owner in this example is a wildcard and matches any query ending with the string right of the asterisk.

EXAMPLE 2
The following BULK RR stores the reverse block of PTR records for the first example.

```
*.55.10.in-addr.arpa. 86400 IN BULK PTR (
                                 [0-255].[0-255].55.10.in-addr.arpa.
                                 pool-A-${1}-${2}.example.com.
                               )
```

The first four fields specify the owner name, TTL, Class, and RR type (BULK). Value "PTR" indicates that this BULK RR defines the PTR record type (Pointer). Value "[0-255].[0-255].55.10.in-addr.arpa." indicates the Label Pattern. Value "pool-A-${1}-${2}.example.com." indicates the Replacement Pattern. The owner in this example is a wildcard and matches any query ending with the string right of the asterisk.

Additional examples can be found in the "BULK Replacement" section.

3. BULK Replacement

The BULK Record is designed to enable DNS zone maintainers to manage large blocks of DNS RRs which all conform to a common pattern. The Label Pattern field provides both a tertiary filter (after owner and type) and a definition of all numeric pattern ranges.

When a query is first received by a DNS nameserver it begins its job of locating an answer-set. In its simplest form this begins by locating the query owner (or wildcard suffix), class and type then returning any matching RR RDATA (or errors).

In the event no matches for the query are found the nameserver of authority will return an error type defined as NXDOMAIN. In the case

of a "BULK" enabled authoritative nameserver an additional step MUST be performed. The nameserver MUST query its local RR database for any "BULK" RRs with a matching owner, class and compatible Match Type. If any such RRs are found the query's owner MUST then be matched against the Label Pattern and all matching BULK records MUST be placed into a temporary processing answer-set. This temporary processing answer-set MUST then follow the Replacement Pattern for each matched record and provided no errors are found SHALL then write this new answer-set to the query's complete answer set. Matching replacements will be of the type specified in the Match Type field of the corresponding BULK RR. Additional detail is provided in the following sections.

3.1. Matching BULK "owner" field

The owner field of all BULK records MUST be that of either a wildcard or hidden wildcard as defined in previous sections. While a hidden wildcard will not be searched for BULK records it will be added to the database for use with the corresponding type field of each BULK RR. This allows location of BULK records to be less conspicuous to the public while still leveraging logic already included in the nameserver thus minimizing the complexity of implementation.

A query SHALL pass the first filter stage (owner match) if an NXDOMAIN is set as the query's current answer set AND the query's owner ends with the BULK record's owner field past the leading hyphen "-" or asterisk "*".

3.2. Matching the BULK "Match Type" field

The RR type of the received query must be compatible with that of the Match Type of owners matched in the section above. That is to say a query for an "A" record will only match BULK records with matching owner and Match Types of "A" (or "CNAME"). All other BULK records matching the query's owner are incompatible and MUST be ignored as part of the selected answer set.

3.3. Matching the BULK "Label Pattern" field

Assuming the RR owner and Match Type fields match the next step is to find compatible Label Patterns. The logic for this falls into two categories; automatic and manual which are described in greater detail in the following sections.

3.3.1. Automatic Label Pattern matching

Automatic Label Pattern matching is determined by use of a single hyphen "-" as the value for Label Pattern field. This assumes everything matches and all hexadecimal or decimal fields will be captured for use as backreferences in the Replacement

Pattern (described below). Automatic Label Pattern matching is often preferred for large blocks such as the reverse IPv6 address space for the simplicity of record management.

3.3.2. Manual Label Pattern matching

Manual Label Pattern matching, while more complex is designed to be both simple to implement and simple to use. Below is an example implementation for label matching using a combination of parsing by regular expression and matching of numeric ranges.

Label Patterns evaluate to current zone ORIGIN as defined in [RFC1034], Section 3. In short this means all Manual Label Patterns must be terminated with a period "." or are assumed relative to the RR's origin.

Numeric Ranges are either decimal or hexadecimal as determined by conditions of query.

If query type is "A" ranges are set to decimal.

If query type is "AAAA" ranges are set to hexadecimal.

If query type is PTR or CNAME the RR owner is used to determine decimal or hexadecimal.

If RR owner ends in ".ip6.arpa." ranges are set to hexadecimal.

If RR owner does _not_ end in ".ip6.arpa." ranges are set to decimal.

Label Patterns MUST NOT contain square braces "[" or "]" which are not part of a numeric range.

3.3.2.1. Manual Label Pattern matching examples

EXAMPLE 1
For this example the query is defined as a PTR record for "10.2.3.4" with an origin of "2.10.in-addr.arpa." and the evaluating BULK RR as:

```
-.2.10.in-addr.arpa. 86400 IN BULK PTR (
                                       [0-255].[0-10]
                                       pool-A-${1}-${2}.example.com.
                                       )
```

STEP 1
Ensure "Label Pattern" is Fully Qualified

```
[0-255].[0-10] == [0-255].[0-10].2.10.in-addr.arpa.
```

```
   STEP 2
      Determine whether range is decimal or hexadecimal

   Query type == "PTR" AND RR owner != "*.ip6.arpa." so range is
   decimal.

   STEP 3
      Build regular expression based on fully qualified label pattern.

   [0-255].[0-10].2.10.in-addr.arpa. ==
                  /^([0-9]{1,3})\.([0-9]{1,2})\.2\.10\.in-addr\.arpa\.$/

   The above regular expression simply matches numeric ranges based on
   decimal or hexadecimal and length.  Numeric range validation occurs
   in the next step.

   STEP 4
       Compare captured numbers and validate ranges

   4.3.2.10.in-addr.arpa.
               =~ /^([0-9]{1,3})\.([0-9]{1,2})\.2\.10\.in-addr\.arpa\.$/

   "4" is captured and within range 0-255 (decimal)
   "3" is captured and within range 0-10 (decimal)

   EXAMPLE 2
   For this example the query is defined as an "AAAA" record for
   "pool-A-ff-aa.example.com." with an origin of "example.com." and
   the evaluating BULK RR as:

   -.example.com. 86400 IN BULK AAAA (
                                      pool-A-[0-ffff]-[0-ffff]
                                      fc00::${1}:${2}
                                     )

   STEP 1
      Ensure Label Pattern is Fully Qualified

   pool-A-[0-ffff]-[0-ffff] == pool-A-[0-ffff]-[0-ffff].example.com.

   STEP 2
      Determine whether range is decimal or hexadecimal

   Query type == "AAAA" so range is hexadecimal.

   STEP 3
      Build regular expression based on fully qualified label pattern.

   pool-A-[0-ffff]-[0-ffff].example.com. ==
        /^pool-A-([0-9a-fA-F]{1,4})-([0-9a-fA-F]{1,4})\.example\.com\.$/
```

The above regular expression simply matches numeric ranges based on decimal or hexadecimal and length. Numeric range validation occurs in the next step.

STEP 4
    Compare captured numbers and validate ranges

```
pool-A-ff-aa.example.com.
  =~ /^pool-A-([0-9a-fA-F]{1,4})-([0-9a-fA-F]{1,4})\.example\.com\.$/
```

"ff" is captured and within range 0-ffff (hexadecimal)
"aa" is captured and within range 0-ffff (hexadecimal)

3.4. Record Generation using the BULK "Replacement Pattern" field

Once it has been determined a query meets all criteria for a BULK record generation the below rules are followed to process captured numeric data and Replacement Pattern into RRs to apply to the answer-set.

3.4.1. Replacement Pattern Backreferences

Before a record may be generated data must be captured in the Label Pattern comparison step above. Each provided numeric range is assigned to a temporary buffer to be used in this step. To make the jobs' of zone administrators easier the order of these buffers will change based on the Match Type and owner so they will default to feel more natural or intuitive. Captured patterns and backreferences are in the same vein as regular expressions and are intended to feel "familiar". This is described in further detail (with examples) in the sections below.

3.4.1.1. Backreference Notation

BULK RRs use a dollar-sign "$" and curly braces "{" and "}" to enclose backreferences within the Replacement Pattern. The following rules are used to determine the final replacement string.

3.4.1.1.1. Simple numeric backreference replacement

The simplest form of backreference notation is its numeric form. In this form only the backreference number falls between the curly braces "{" and "}". An example is "${1}" which would be replaced by the value in the first capture position. Position is described in detail in a later section.

Numeric backreference replacement indices start with one "1" to maintain consistency with regular expression backreferences.

3.4.1.1.2. Star backreference replacement

The next form of backreference notation is its star (or asterisk "*") form. In this form only an asterisk falls between the curly braces "{" and "}". This form "${*}" would be replaced by all captured values in order of ascending position delimited by its default delimiter (described below). Position is described in detail in a later section.

3.4.1.1.3. Numeric range backreference replacement

The next form of backreference notation is the numeric range form. In this form a range of numbers falls between the curly braces "{" and "}". An example of this is "${1-4}" which would be replaced by all captured values within this range (1-4) in order of positions provided delimited its default delimiter (described below). To reverse the order of positions in this example one could simply reverse the upper and lower values to look like "${4-1}". Position is described in detail in a later section.

3.4.1.1.4. Numeric set backreference replacement

The next form of backreference notation is the numeric set form. In this form a set of numbers falls between the curly braces "{" and "}" separated by commas. An example of this is "${1,4}" which would be replaced by the first and fourth captured values in the order of position provided delimited its default delimiter (described below). Position is described in detail in a later section.

This notation may be combined with the numeric range form allowing specific positions or position ranges to be used. Examples would be "${3,2,1,4-8}" and "${8-12,1-4}".

3.4.1.1.5. Backreference delimiter

The above sections reference a default delimiter. In an effort to provide an intuitive zone management experience the default delimiter will be based on the BULK RR's Match Type. For Match Types "A" and "AAAA" the default delimiter will be a period "." and for Match Types "PTR" and "CNAME" the default delimiter will be a hyphen "-". In either case the delimiter may be overridden by including it in the backreference braces after the set selectors and a backreference field separator character, the pipe "|". An example would be "${*|-}" which would force a hyphen "-" delimiter. An empty or null delimiter is allowed by not specifying a delimiter character, for

example "${*|}", which would simply concatenate all captured values in order of capture position. Position is described in detail in a later section.

3.4.1.1.6. Backreference delimiter interval

The default behavior of a backreference set is to combine each captured value specified with a delimiter between each. To allow captured backreferences to be delimited at another interval a third backreference field is provided. An example would be "${*|-|4}" which would concatenate all captured values but delimiting only every fourth value with hyphens "-". This can be a handy feature in the IPv6 reverse namespace where every nibble is captured as a separate value and generated hostnames include sets of 4 nibbles. An empty or null value MUST be interpreted as "1" or every captured value.

3.4.1.1.7. Backreference padding length

When generating BULK based records a common requirement is to convert from one numeric format to another, padding is among the most common of these. The fourth and final backreference field determines what width to pad to. An example would be "${*|||4}" which would set the width of all captured values to 4 by inserting leading zeros to fill the void. The default is empty or null which MUST be interpreted as NO modification. A width of zero "0" has a special interpretation referred to as "unpad" meaning all leading zeros MUST be removed. If a value is provided captured values longer than this width MUST be truncated to fit the specified width. In the case where a delimiter interval is provided captured values between the intervals will be concatenated and the padding or unpadding applied as a unit and not individually. An example of this would be "${*||4|4}" which would combine each range of 4 captured values and pad them to a width of 4 characters by inserting leading zeros where necessary.

3.4.1.1.8. Backreference Position

Great effort has gone into providing zone maintainers an intuitive syntax. As part of this effort, the captured values will reverse direction depending on several factors.

As a general rule of thumb, if it makes sense the numeric ranges are in reverse order from query to answer then they will be reversed. Otherwise they will be in the same order.

Take for example a simple reverse DNS lookup, from "10.2.3.4" to "pool-A-3-4.example.com.". Since DNS zones are arranged according to management authority the records appear reversed numerically. In this example "10.2.3.4" becomes "4.3.2.10.in-addr.arpa.". One would intuitively expect this reversal to be reversed so positional indices of captured values would increment toward the

right of the Replacement Pattern. This expectation is especially important when using range based replacements.

Formally, the rules for position reversal are as follows:

Match Type RRs for "PTR" are reversed for zone owners ending in either ".in-addr.arpa." or "ip6.arpa.". All other Match Type RRs for "PTR" are forward.

Match Type RRs for "A" (Address), "AAAA" (IPv6 Address) and "CNAME" (Canonical Name) are forward.

3.4.1.1.9. Backreference Position Negation

To allow simple reversal of any backreference notation a single exclamation point character "!" MAY be used as the first character of a backreference set. Examples would be "${!*}" and "${!1-4,7}". In both of the examples the backreference positions SHALL be the exact mirror equivalent as those without the leading exclamation point "!". This can be very important if the BULK generated replacements have values in positions opposite to what is required or expected.

3.4.2. Replacement Pattern examples

This section provides examples of several BULK RR Replacement Patterns. Each example is intended to further understanding for implementors and DNS administrators alike.

EXAMPLE 1
For this example the query is defined as a PTR record for "10.2.3.4" with an origin of "2.10.in-addr.arpa." and the evaluating BULK RR as:

```
- 86400 IN BULK PTR - pool-${*}.example.com.
```

This example contains several of the features described above.

First, the record owner is simply a single hyphen "-" denoting it is a "hidden wildcard" (wildcard for generated records but not for BULK).

Second, the Label Pattern is also a single hyphen "-" denoting all queries matching the owner's wildcard pattern for the "PTR" Match Type are accepted and will be captured for use in the Replacement Pattern.

Third, the Replacement Pattern contains a single "star" backreference denoting all captured numeric (decimal) backreferences will be combined with its default delimiter of hyphen "-" (for PTR) and

placed into the backreference's position in the answer-set. Should this generate an invalid hostname the response will be NXDOMAIN unless other BULK records match and are successfully generated without error.

The owner for "10.2.3.4" is "4.3.2.10.in-addr.arpa." and creates matching backreferences for "4", "3", "2" and "10" then reverses their indices so "${1}" resolves to "10", "${2}" to "2", "${3}" to "3" and "${4}" to "4" respectively. When applied to the Replacement Pattern the answer becomes "pool-10-2-3-4.example.com.".

EXAMPLE 2
For this example the query is defined as a PTR record for "10.2.3.4" with an origin of "2.10.in-addr.arpa." and the evaluating BULK RR as:

```
- 86400 IN BULK PTR - pool-${*|||3}.example.com.
```

This example expands on EXAMPLE 1 with the differences outlined below.

The only change to the BULK RR is the Replacement Pattern includes additional fields, specifically null values for delimiter and interval and a padding width of 3.

The owner for "10.2.3.4" is "4.3.2.10.in-addr.arpa." and creates matching backreferences for "4", "3", "2" and "10" and reverses their indices so "${1}" resolves to "10", "${2}" to "2", "${3}" to "3" and "${4}" to "4" respectively. When applied to the Replacement Pattern the answer becomes "pool-010002003004.example.com.".

EXAMPLE 3
This example contains a classless IPv4 delegation on the /22 CIDR boundary as defined by [RFC2317]. The network for this example is

"10.2.0/22" delegated to a nameserver "ns1.sub.example.com.". RRs for this example are defined as:

```
$ORIGIN 2.10.in-addr.arpa.
0-3 86400 IN      NS    ns1.sub.example.com.
-   86400 IN BULK CNAME [0-255].[0-3] ${*|.}.0-3
```

For this example, the query would come in for "25.2.2.10.in-addr.arpa.". After matching the owner filter (ending in ".2.10.in-addr.arpa.") and the fully qualified label pattern of "[0-255].[0-3].2.10.in-addr.arpa." the answer-set would include a generated RR consisting of captured values "25" and "2" joined by the custom delimiter of period "." then joined by ".0-3" and made fully qualified. The resulting RR would be a "CNAME" with RDATA of

"25.2.0-3.2.10.in-addr.arpa.". This record is now one delegated to "ns1.sub.example.com." as its authority and the answer-set is complete.

4. The NPN Resource Record

The NPN resource record provides pre-processing directives for Numeric Pattern Normalization (NPN) based RR signature generation.

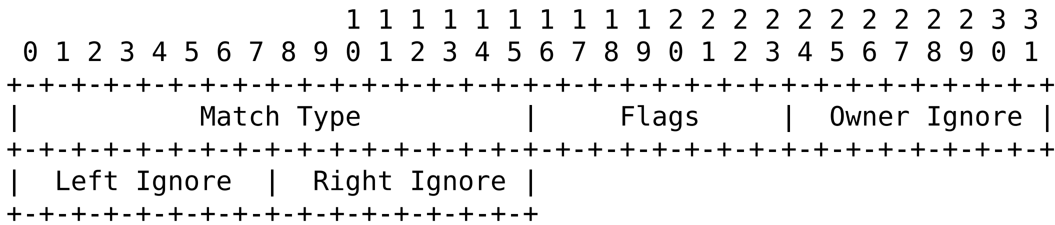
The Type value for the NPN RR type is XX.

The NPN RR is class independent.

The NPN RR has no special TTL requirements.

4.1. NPN RDATA Wire Format

The RDATA for a NPN RR consists of a 2 octet Type Field, a 1 octet Label Ignore Field, a 1 octet Left Ignore Field and 1 octet Right Ignore field.



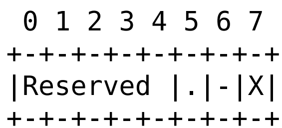
```
                     1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Match Type          |     Flags     |  Owner Ignore |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Left Ignore  |  Right Ignore |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

4.1.1. The Match Type field

The Match Type field identifies the type of the RRset identified by this NPN record.

4.1.2. The Flags field

The Flags field defines additional processing parameters for data normalization. This document defines only the Period-As-Number flag "." (position 5), the Hyphen-As-Number "-" (position 6) and the hexadecimal flag "X" (position 7). All other flags are reserved for future use.



```
 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
|Reserved |.|-|X|
+-+-+-+-+-+-+-+-+
```

Bits 0-4: Reserved for future
   These flags have no default value if set to false (0).

Bit 5: Period As Number (.) Flag
This flag indicates the period (dot) will be processed as a number.
This flag has no default value if set to false (0).

Bit 6: Hyphen As Number (-) Flag
This flag indicates the hyphen (dash) will be processed as a number.
This flag has no default value if set to false (0).

Bit 7: Hexadecimal (X) Flag
This flag indicates the highest value for Normalization Processing is "f". Normalization Processing will be described in a later section. This flag has a default value of "9" if set to false (0).

4.1.3. The Owner Ignore field

The Owner Ignore field defines the length of characters as counted from the left-hand side of the owner which MUST be ignored by the normalization process. Normalization Processing will be described further in a later section.

4.1.4. The Left Ignore field

The Left Ignore field defines the length of characters as counted from the left-hand side of the generated RDATA which MUST be ignored by the normalization process. Normalization Processing will be described further in a later section.

4.1.5. The Right Ignore field

The Right Ignore field defines the length of characters as counted from the right-hand side of the generated RDATA which MUST be ignored by the normalization process. Normalization Processing will be described further in a later section.

4.2. The NPN RR Presentation Format

The Match Type field is represented as an RR type mnemonic. When the mnemonic is not known, the TYPE representation as described in [RFC3597], Section 5, MUST be used.

The Flags field MUST be presented as a string of characters representing each flag bit. This document defines only the period ".", hyphen "-" and hexadecimal "X" flags. Flags MAY appear in any order. For example; all three flags could appear as "-9." or ".f-" (without the quotes). If all bits are zero all default values (if defined) would be presented ("9" as currently defined).

All Ignore fields MUST be presented as an unsigned decimal integers and fall within the 0-255 range available to a single octet.

4.3. Normalization Processing of NPN RRs

This document provides a minor yet significant modification to DNSSEC regarding how RRsets will be signed or verified. Specifically the Signature Field of [RFC2535], Section 4.1.8. Prior to processing into canonical form, signed zones may contain associated RRs where; owner, class and type of a non NPN RR directly corresponds with an NPN RR matching owner, class and Match Type. If this condition exists the NPN RR's RDATA defines details for processing the associated RDATA into a "Normalized" format. Normalized data is based on pre-canonical formatting and zero padded for "A" and "AAAA" RR types for acceptable precision during the process. This concept will become clearer in the NPN pseudocode and examples provided in the sections to follow.

The rules for this transformation are simple:

For RR's Owner field, characters from the beginning to the index of the Owner Ignore value or the final string of characters belonging to the zone's ORIGIN MUST NOT be modified by this algorithm. While the Owner Ignore value is not used for BULK records but is included with the expectation other pattern-based resource records may emerge and leverage NPN records for their DNSSEC support requirements.

For RR's RDATA field, character from beginning to the index of Left Ignore value or characters with index of Right Ignore value to the end MUST NOT be modified by this algorithm.

In the remaining portion of both Owner and RDATA strings of numeric data, defined as character "0" through "f" or "0" through "9" depending on whether or not the Hexadecimal flag is set or not, MUST be consolidated to a single character and set to the highest value defined by the Hexadecimal flag. Examples may be found in the following section. If period-as-number or hyphen-as-number flags are set whichever are used ("." or "-") would be treated as part of the number and consolidated where appropriate.

Once the normalization has been performed the signature will continue processing into canonical form using the normalized RRs in the place of original ones.

One thing to keep in mind when calculating values for the Ignore fields is the Label Pattern and Replacement Pattern fields are considered relative unless terminated by a period. When processing

NPN records the fully-qualified Patterns will be used for determining which characters should be ignored.

NPN RRs MAY be included in the "Additional" section to provide a hint for NPN processing required for verification path.

It is important to note, properly sizing the Ignore fields is critical to minimizing the risk of spoofed signatures. Never intentionally set all Ignore values to zero in order to make validation easier as it places the validity of zone data at risk. Only accompany RRs which are pattern derived (such as BULK) with NPN records as doing so may unnecessarily reduce the confidence level of generated signatures.

4.3.1. Pseudocode for NPN Normalization Processing

This section provides a simple demonstration of process flow for NPN rdata normalization and DNSSEC signatures.

The pseudocode provided below assumes all associated RRs are valid members of a DNSSEC compatible RRset (including BULK generated ones).

```
for rr in rrset
    if (has_NPN<rr.owner, rr.class, rr.type>)
        rr.rdata_normal = NPN_normalize<rr.rdata>
        add_to_sigrrset<NPN.owner, rr.class, rr.type,
            rr.rdata_normal>
        next
    else
        add_to_sigrrset<rr.owner, rr.class, rr.type, rr.rdata>
        next

process_canonical_form<sigrrset>

dnssec_sign<sigrrset>
```

Similar logic MUST be used for determining DNSSEC validity of RRsets in verification (validation) nameservers for signatures generated based on NPN normalization.

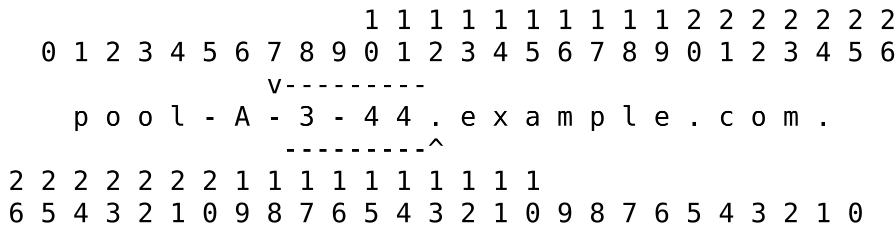
4.3.2. NPN Normalization Processing examples

EXAMPLE 1
For this example the query is defined as a PTR record for "10.2.3.44" with an origin of "2.10.in-addr.arpa." and the evaluating BULK and NPN RR as:

```
-.2.10.in-addr.arpa. 86400 IN BULK PTR (
                                  [0-255].[0-10]
                                  pool-A-${1}-${2}.example.com.
                                  )
*.2.10.in-addr.arpa. 86400 IN NPN  PTR 9 0 7 13
```

As shown previously in BULK RR examples the query would enter the nameserver with an owner of "44.3.2.10.in-addr.arpa." and a "PTR" RR with the RDATA of "pool-A-3-44.example.com." would be generated.

By protecting the "Ignore" characters from the generated RR's RDATA the focus for normalization becomes "3-44" as illustrated below.



Everything to the left of "3-44" will remain intact as will everything to its right. The remaining characters will be processed for numbers between "0" and "9" as indicated by the NPN record's hexadecimal flag "9" and each run replaced by the single character "9". The final Normalized RDATA would therefore become "pool-A-9-9.example.com." and its signature would be based on this "normalized" RDATA field. This new "normalized" string would be used as an RDATA for the wildcard label of "*.2.10.in-addr.arpa." now encompassing all possible permutations of the "pool-A-${1}-${2}.example.com." pattern.

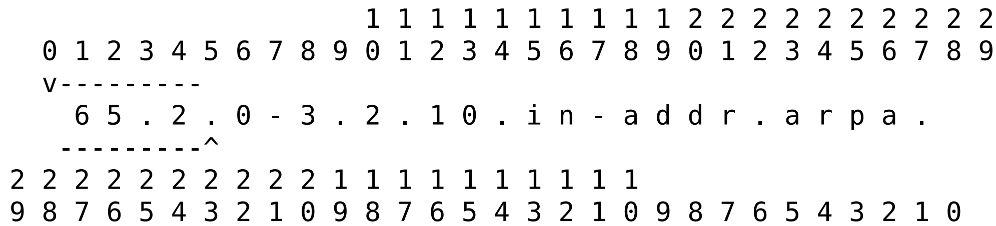
Since the verification (validation) nameserver would use the identical NPN record for processing and comparison, all RRs generated by the BULK record can now be verified with a single wildcard signature.

EXAMPLE 2
This example contains a classless IPv4 delegation on the /22 CIDR boundary as defined by [RFC2317]. The network for this example is "10.2.0/22" delegated to a nameserver "ns1.sub.example.com.". RRs for this example are defined as:

```
$ORIGIN 2.10.in-addr.arpa.
0-3 86400 IN      NS    ns1.sub.example.com.
-   86400 IN BULK CNAME [0-255].[0-3] ${*|.}.0-3
*   86400 IN NPN  CNAME 9 0 0 23
```

For this example, a query of "10.2.2.65" would enter the nameserver as "65.2.2.10.in-addr.arpa." and a "CNAME" RR with the RDATA of "65.2.0-3.2.10.in-addr.arpa." would be generated.

By protecting the "Ignore" characters from the generated RR's RDATA the focus for normalization becomes "65.2" as illustrated below.



```
                           1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
       v---------
         6 5 . 2 . 0 - 3 . 2 . 1 0 . i n - a d d r . a r p a .
        ---------^
     2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
     9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
```

Everything to the left of "65.2" will remain intact as will everything to its right. The remaining characters will be processed for numbers between "0" and "9" as indicated by the NPN record's hexadecimal flag "9" and each run replaced by the single character "9". The final Normalized RDATA would therefore become "9.9.0-3.2.10.in-addr.arpa." and its signature would be based on this "normalized" RDATA field. This new "normalized" string would be used as an RDATA for the wildcard label of "*.2.10.in-addr.arpa." now encompassing all possible permutations of the "${*|.}.0-3.2.10.in-addr.arpa." pattern.

As in example 1, the verification (validation) nameserver would use the same NPN record for comparison.

EXAMPLE 3
This example provides reverse logic for example 1 by providing an IPv4 "A" record for a requested hostname. For this example the query is defined as an "A" record for "pool-A-3-44.example.com." with an origin of "example.com.". RRs for this example are defined as:

```
-.example.com. 86400 IN BULK A (
                                   pool-A-[0-10]-[0-255]
                                   10.2.${*}
                                  )
*.example.com. 86400 IN NPN  A 9 0 8 0
```

By protecting the "Ignore" characters from the generated RR's RDATA the focus for normalization becomes "003.044" as illustrated below.

```
                                  1 1 1 1 1 1 1 1 1
                  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8
                                v--------------
                    0 1 0 . 0 0 2 . 0 0 3 . 0 4 4
                                 ---------------^
              1 1 1 1 1 1 1 1 1
              8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
```

This example illustrates a key point about NPN records; since they are pre-canonical they MUST operate on a strict subset of WIRE formatted data. For "A" and "AAAA" records this means the "Ignore" fields are based on zero padded data. In this example our generated record MUST be converted into "010.002.003.044" (shown above) prior to processing. After processing, wire format would become "0x0A02032C" (shown in hexadecimal). This format would be too imprecise for normalization so padded decimal is used.

Everything to the left of "003.044" will remain intact as will everything to its right. The remaining characters will be processed for numbers between "0" and "9" as indicated by the NPN record's hexadecimal flag "9" and each run replaced by the single character "9". The final Normalized RDATA would therefore become "10.2.9.9" and its signature would be based on this "normalized" RDATA field. This new "normalized" "A" RR would be used as an RDATA for the wildcard label of "*.example.com." now encompassing all possible permutations of the "10.2.${*}" pattern.

EXAMPLE 4
This example provides similar logic for an IPv6 AAAA record. For this example the query is defined as an "AAAA" record for "pool-A-ff-aa.example.com." with an origin of "example.com.". RRs for this example are defined as:

```
-.example.com. 86400 IN BULK AAAA (
                                   pool-A-[0-ffff]-[0-ffff]
                                   fc00::${1}:${2}
                                  )
*.example.com. 86400 IN NPN  AAAA X 0 30 0
```

By protecting the "Ignore" characters from the generated RR's RDATA the focus for normalization becomes "00ff:00aa" as illustrated below.

```
                      1 1 1 1 1 1 1 1 1 1 2 2
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

    f c 0 0 : 0 0 0 0 : 0 0 0 0 : 0 0 0 0 : -/-/

  4 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 1
  0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9
```

```
      /-/-/- . . . . . . . . . . . . . . . . . . . . . . . . -/-/-/
                             2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 4
                             1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
                                               v------------------
                        /-/- 0 0 0 0 : 0 0 0 0 : 0 0 f f : 0 0 a a
                                                ------------------^
                           2 1 1 1 1 1 1 1 1 1 1
                           0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
```

This example reinforces the point on pre-canonical processing of NPN records; they MUST operate on a strict subset of WIRE formatted data. For "A" and "AAAA" records this means the "Ignore" fields are based on zero padded data. In this example our generated record MUST be converted into "fc00:0000:0000:0000:0000:0000:00ff:00aa" (shown above) prior to processing. After processing, wire format would become "0xFC000000000000000000000000FF00AA" (shown in hexadecimal). This format is slightly misleading as it is truly only 16 bytes of WIRE data and would be too imprecise for normalization so padded hexadecimal is used.

Everything to the left of "00ff:00aa" will remain intact as will everything to its right. The remaining characters will be processed for numbers between "0" and "f" as indicated by the NPN record's hexadecimal flag "X" and each run replaced by the single character "f". The final Normalized RDATA would therefore become "fc00::f:f" and its signature would be based on this "normalized" RDATA field. This new "normalized" "AAAA" RR would be used as an RDATA for the wildcard label of "*.example.com." now encompassing all possible permutations of the "fc00::${1}:${2}" pattern.

5. Positive Side-Effects

This section highlights positive side effects of some architectural decisions regarding the BULK RR design.

5.1. Record Superimposition

The main side-effect of the BULK RR design is superimposition. RRs created by the BULK generation process generally rely on the logic of wildcard assignment. This logic only provides answers where no others exist. This means in the reverse DNS world (network assignment) HUGE blocks of addresses can be assigned a single BULK record and where delegated to another customer or SWIP will be automatically overridden.

When compared with bind's $GENERATE statement, if a singleton record such as CNAME appears within a $GENERATE range, either the CNAME or $GENERATE becomes invalid. While a BULK record range would

automatically notch out the CNAME without user intervention or creating a potential management problem for the future when two $GENERATES create a hole where the CNAME no longer exists. BULK RRs would again automatically reassign the missing record to one of its own.

5.2. Pattern Based DNSSEC support

The NPN resource record can be used to support other dynamic RR types which do not currently exist.

6. Known Limitations

This section defines known limitations of the BULK resource type.

6.1. Increased CPU utilization for NXDOMAIN RRs

Nameserver requirements to support BULK RRs will minimally increase CPU utilization requirements compared to most RR types. However, since the inception of DNSSEC more is expected of DNS servers at a system resource level and it is the authors' belief the benefit outweighs the sacrifice.

A quick comparison of BULK versus bind's $GENERATE expansion reveals much more memory would be sacrificed with $GENERATES to save the CPU cycles required to support BULK records. Additionally, $GENERATES cannot be transferred (i.e. AXFR) without expansion and an IPv6 CIDR even as small as /96 would be simply impossible. BULK on the other hand can easily support IPv6 CIDRs of /64 and much larger with very little effort.

6.2. Pre-Adoption Nameserver Implications

While there is an added demand on authoritative nameservers, there are no new requirements to recursive (caching) resolvers for non-DNSSEC record handling. Even authoritative nameservers are able to transfer to and from supporting nameservers with no requirement (although would be unable to return BULK generated records without support).

Prior to widespread adoption on the authoritative side all generated records would be invisible if served on nameservers lacking support. Since generated records are generally NOT service impacting records this should be understood but not of great concern.

Once adoption has reached an appreciable level on the producer (authoritative) side only DNSSEC requirements remain for the consumer (resolver) side. This behavior is fully expected.

7. Security Considerations

Two known security considerations exist for the BULK resource record, DNSSEC and DDOS attack vectors. Both are addressed in the following sections.

7.1. DNSSEC Signature Strategies

DNSSEC was designed to provide verification (validation) for DNS resource records. In a nutshell this requires each (owner, class, type) tuple to have its own signature. This essentially defeats the purpose of providing large generated blocks of RRs in a single RR as each generated RR would require its own legitimate RRSIG record.

In the following sections several options are discussed to address this issue. Of the options, on-the-fly provides the most secure solution and NPN provides the most flexible.

7.1.1. On-the-fly (Live) Signatures

This solution requires authoritative nameservers to sign generated records _as_they_are_generated_. Not all authoritative nameserver implementations offer on-the-fly (realtime) signatures so this solution would either require all implementations to support on-the-fly signing or be ignored by implementations which can not or will not comply.

No changes to recursive (resolving) nameservers is required to support this solution.

7.1.2. Normalized (NPN Based) Signatures

This solution provides the most flexible solution as nameservers without on-the-fly signing capabilities can still support signatures for BULK records. The down side to this solution is it requires recursive (resolving) nameserver support. Unless a recursive nameserver can verify the signature it is unverifiable.

NPN records are likely to be a topic of great debate as to their own security limitations. It is, however, the authors' belief; while any logic which limits the input of digital signatures, lessens the validity of such signatures, the limitation is minimal and the gain is significant. The main reason for this is as a general rule, RRs used in a generic manner such as conventional $GENERATE RRs or scripted mass pattern generated RRs have a lesser importance than other RRs in managed zones. These therefore inherently pose less risk by means of attack and have a much less reward by defeating security measures.

This being said, care must still be taken to set the Ignore fields appropriately to minimize exposure and only use NPN RRs to secure pattern-based records such as BULK.

7.1.3. Non-DNSSEC Zone Support Only

As a final option zones which wish to remain entirely without DNSSEC support may serve such zones without either of the above solutions and records generated based on BULK RRs will require zero support from recursive (resolving) nameservers.

7.2. DNSSEC Verifier Details

Verification of DNSSEC signed BULK generated RRs may be performed against on-the-fly signatures with zero modification to their behavior. However, verification against NPN records would require changes to the logic to incorporate processing RDATA generated by BULK logic as described above so the results will be compatible.

7.3. DDOS Attack Vectors and Mitigation

As an additional defense against Distributed Denial Of Service (DDOS) attacks against recursive (resolving) nameservers it is highly recommended shorter TTLs be used for BULK RRs than others. While disabling caching with a zero TTL is not recommended (as this would only result in a shift of the attack target) a balance will need to be found. While this document uses 24 hours (86400) in its examples values between 300 to 900 are likely more appropriate and is RECOMMENDED. What is ultimately deemed appropriate may differ from zone to zone and administrator to administrator.

8. IANA Considerations

IANA is requested to assign numbers for two DNS resource record types identified in this document; BULK and NPN.

9. Acknowledgments

This document was created as an extension to the DNS infrastructure. As such, many people over the years have contributed to its creation and the authors are appreciative to each of them even if not thanked or identified individually.

10. References

10.1. Normative References

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2181] Elz, R. and R. Bush, "Clarifications to the DNS Specification", RFC 2181, July 1997.

[RFC2308] Andrews, M., "Negative Caching of DNS Queries (DNS NCACHE)", RFC 2308, March 1998.

[RFC2536] Eastlake 3rd, D., "DSA KEYs and SIGs in the Domain Name System (DNS)", RFC 2536, March 1999.

[RFC2931] Eastlake 3rd, D., "DNS Request and Transaction Signatures (SIG(0)s)", RFC 2931, September 2000.

[RFC3110] Eastlake 3rd, D., "RSA/SHA-1 SIGs and RSA KEYs in the Domain Name System (DNS)", RFC 3110, May 2001.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 2005.

[RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, March 2005.

10.2. Informative References

[RFC2535] Eastlake 3rd, D., "Domain Name System Security Extensions", RFC 2535, March 1999.

Authors' Addresses

John Woodworth
4250 North Fairfax Drive
Arlington, VA 22203
USA

EMail: John.Woodworth@CenturyLink.com

Dean Ballew
2355 Dulles Corner Boulevard Suite 200 300
Herndon, VA 20171
USA

EMail: Dean.Ballew@CenturyLink.com

Shashwath Bindinganaveli Raghavan
2355 Dulles Corner Boulevard Suite 200 300
Herndon, VA 20171
USA

EMail: Shashwath.Bindinganaveliraghavan@CenturyLink.com

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.