DNSOP Working Group
Internet-Draft
Updates: 2308, 4033, 4034, 4035 (if approved)
Intended status: Informational
Expires: January 22, 2020

J. Woodworth
D. Ballew
CenturyLink, Inc.
S. Bindinganaveli Raghavan
Hughes Network Systems
D. Lawrence
Oracle

July 21, 2019

BULK DNS Resource Records
draft-woodworth-bulk-rr-09

Abstract

The BULK DNS resource record type defines a method of pattern-based creation of DNS resource records based on numeric substrings of query names. The intent of BULK is to simplify generic assignments in a memory-efficient way that can be easily shared between the primary and secondary nameservers for a zone.

Ed note

Text inside square brackets ([]) is additional background information, answers to frequently asked questions, general musings, etc. They will be removed before publication. This document is being collaborated on in GitHub at <https://github.com/ioneyez/bulk-rr>. The most recent version of the document, open issues, etc should all be available here. The authors gratefully accept pull requests.

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 https://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."

This Internet-Draft will expire on January 22, 2020.

Table of Contents

1. Introduction

The BULK DNS resource record defines a pattern-based method for on-the-fly resource record generation. It is essentially an enhanced wildcard mechanism, constraining generated resource record owner names to those that match a pattern of variable numeric substrings. It is also akin to the $GENERATE master file directive [bind-arm] without being limited to numeric values and without creating all possible records in the zone data.

For example, consider the following record:

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

It will answer requests for pool-A-0-0.example.com through pool-A-255-255.example.com with the IPv4 addresses 10.55.0.0 through 10.55.255.255.

Much larger record sets can be defined while minimizing the associated requirements for server memory and zone transfer network bandwidth.

This record addresses a number of real-world operational problems that authoritative DNS service providers experience. For example, operators who host many large reverse lookup zones, even for only IPv4 space in in-addr.arpa, would benefit from the disk space, memory size, and zone transfer efficiencies that are gained by encapsulating a simple record-generating algorithm versus enumerating all of the individual records to cover the same space.

Production zones of tens of thousands of pattern-generated records currently exist, that could be reduced to just one BULK RR. These zones can look deceptively small on the primary nameserver and balloon to 100MB or more when expanded,

BULK also allows administrators to more easily deal with singletons, records in the pattern space that are an exception to the normal data generation rules. Whereas a mechanism like $GENERATE may need to be adjusted to account for these individual records, the processing rules for BULK have explicit records more naturally override the dynamically generated ones. This collision problem is not just a

theoretical concern, but a real source of support calls for providers.

Pattern-generated records are also not only for the reverse DNS space. Forward zones also occasionally have entries that follow patterns that would be well-addressed by the BULK RR.

1.1. Background and Terminology

The reader is assumed to be familiar with the basic DNS and DNSSEC concepts described in [RFC1034], [RFC1035], [RFC4033], [RFC4034], and [RFC4035]; subsequent RFCs that update them in [RFC2181] and [RFC2308]; and DNS terms in [RFC7719].

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] when, and only when, they appear in all capitals, as shown here.

2. The BULK Resource Record

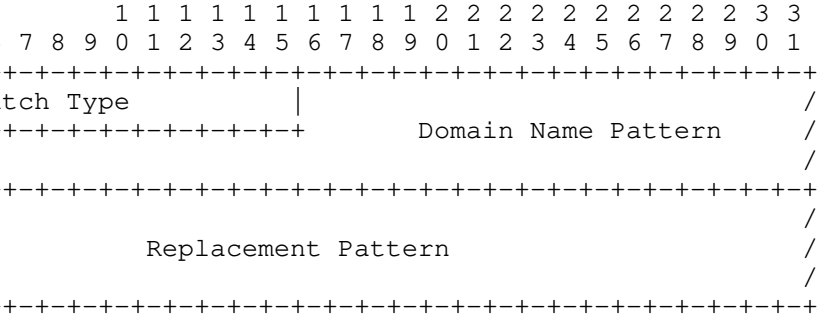
The BULK resource record enables an authoritative nameserver to generate RRs for other types based upon the query received.

The Type value for the BULK RR type is TBD.

The BULK RR is class-independent.

2.1. BULK RDATA Wire Format

The RDATA for a BULK RR is as follows:



Match Type identifies the type of the RRset to be generated by this BULK record. It is two octets corresponding to an RR TYPE code as specified in [RFC1035], Section 3.2.1.

Domain Name Pattern consists of a pattern encoded as a wire-format fully qualified domain name. The full name is used so that numeric substrings above the zone cut can be captured in addition to those in the zone. It needs no length indicator for the entire field because the root label marks its end.

Special characters are interpreted as per the following Augmented Backus-Naur Form (ABNF) notation from [RFC5234].

```
match         =  1*(range / string)

range         =  "[" [decnum "-" decnum] "]" /
                  "<" [hexnum "-" hexnum] ">"
                       ; create references for substitution
                       ; limit of 32 references
                       ; [] is syntactic sugar for 0-255
                       ; <> is syntactic sugar for 00-ff

string        =  1*(ctext / quoted-char)

decnum        =  1*decdigit
                       ; constrained to 65535 maximum.

hexnum        =  1*hexdigit
                       ; constrained to ffff maximum.

octet         =  %x00-FF

decdigit      =  %x30-39
                       ; 0-9
hexdigit      =  decdigit / 0x41-0x46 / 0x61-66
                       ; 0-9, A-F, a-f

ctext         =  <any octet excepting "\">

quoted-char   = "\" octet
                        ; to allow special characters as literals
```

Interpretation of the Domain Name Pattern is described in detail in the "BULK Replacement" section. Note that quoted-char must be stored in the wire format to preserve its semantics when the BULK RR is interpreted by nameservers.

The limit of 32 references is meant to simplify implementation details. It is largely but not entirely arbitrary, as it could capture every individual character of the text representation of a full IPv6 address.

Replacement Pattern describes how the answer RRset MUST be generated for the matching query. It needs no length indicator because its end can be derived from the RDATA length minus Match Type and Domain Name Pattern lengths. It uses the following additional ABNF elements:

```
replace       =   1*(reference / string)

reference     =   "$" "{" (positions / "*") [options] "}"

positions     =   (position / posrange) 0*("," (position / posrange))

posrange      =   position "-" position

position      =   1*decnum

options       =   delimiter [interval [padding]]

delimiter     =   "|" 0*(ctext | quoted-char)
                        ; "\|" to use "|" as delimiter
                        ; "\\" to use "\" as delimiter

interval      =   "|" *2decdigit

padding       =   "|" *2decdigit
```

[Is the formatting complexity beyond simple ${1}, ${2}, etc, really worth it? I definitely see how it could make for shorter replacement patterns, but does it enhance their clarity and usability, adding a feature someone really wants?]

The Replacement Pattern MUST end in the root label if it is intended to represent a fully qualified domain name.

2.2. The BULK RR Presentation Format

Match Type is represented as an RR type mnemonic or with [RFC3597]'s generic TYPE mechanism.

Domain Name Pattern is represented as a fully qualified domain name as per [RFC1035] Section 5.1 rules for encoding whitespace and other special characters.

Replacement Pattern is represented by the standard <character-string> text rules for master files as per [RFC1035] section 5.1.

It is suggested that lines longer than 80 characters be wrapped with parenthetical line continuation, per [RFC1035] Section 5.1, starting after Match Type and ending after Replacement Pattern.

3. BULK Replacement

When a BULK-aware authoritative nameserver receives a query for which it does not have a matching name or a covering wildcard, it MUST then look for BULK RRs at the zone apex, selecting all BULK RRs with a Match Type that matches the query type and a Domain Name Pattern that matches the query name. Note that query type ANY will select all Match Types, and all query types match a CNAME or DNAME Match Type. One or more answer RRs will be generated per the replacement rules below. Examples are provided in an appendix.

By only triggering the BULK algorithm when the query name does not exist, administrators are given the flexibility to explicitly override the behaviour of specific names that would otherwise match the BULK record's Domain Name Pattern. This is unlike BIND's $GENERATE directive, which adds the generated RRs to any existing names.

3.1. Matching the Domain Name Pattern

A query name matches the Domain Name Pattern if the characters that appear outside the numeric ranges match exactly and those within numeric ranges have values that fall within the range. Numeric matches MUST be of the appropriate decimal or hexadecimal type as specified by the delimiters in the pattern. For example, if a range is given as [0-255], then FF does not match even though its value as a hexadecimal number is within the range. Leading zeros in the numeric part(s) of the qname MUST be ignored; for example, 001.example.com, 01.example.com and 1.example.com would all match [].example.com.

When a query name matches a Domain Name Pattern, the value in each numeric range is stored for use by the Replacement Pattern, with reference numbers starting at 1 and counting from the left. For example, matching the query name host-24-156 against host-[0-255]-[0-255] assigns 24 to ${1} and 156 to ${2}.

3.2. Record Generation using Replacement Pattern

The Replacement Pattern generates the record data by replacing the ${...} references with data captured from the query name, and copying all other characters literally.

The simplest form of reference uses only the reference number between the braces, "{" and "}". The value of the reference is simply copied directly from the matching position of the query name.

The next form of reference notation uses the asterisk, "*". With ${*}, all captured values in order of ascending position, delimited by its default delimiter (described below), are placed in the answer. The commercial-at, "@" symbol captures in the same way only in order of descending position.

Numeric range references, such as ${1-4}, replaces all values captured by those references, in order, delimited by the default delimiter described below. To reverse the order in which they are copied, reverse the upper and lower values, such as ${4-1}. This is useful for generating PTR records from query names in which the address is encoded in network order.

Similar to range references, separating positions by commas creates sets for replacement. For example, ${1,4} would be replaced by the first and fourth captured values, delimited its default delimiter. This notation may be combined with the numeric range form, such as ${3,2,1,8-4}.

3.2.1. Delimiters

A reference can specify a delimiter to use by following a vertical bar, "|", with zero or more characters. Zero characters, such as in ${1-3|}, means no delimiter is used, while other characters up to an unescaped vertical bar or closing brace are copied between position values in the replacement. The default delimiter is the hyphen, "-".

3.2.2. Delimiter intervals

A second vertical bar in the reference options introduces a delimiter interval. The default behavior of a multi-position reference is to combine each captured value specified with a delimiter between each. With a delimiter interval the delimiters are only added between every Nth value. For example, ${*|-|4} adds a hyphen between every group of four captured positions. 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 0 value for the delimiter interval MUST be interpreted as the default value of 1.

3.2.3. Padding length

The fourth and final reference option determines the field width of the copied value. Shorter values MUST be padded with leading zeroes ("0") and longer values MUST be truncated to the width.

The default behavior, and that of an explicit empty padding length, is that the captured query name substring is copied exactly. A width of zero "0" is a signal to "unpad", and any leading zeros MUST be removed. [Unnecessary complexity?]

If a delimiter interval greater than 1 is used, 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 or truncate them to a width of 4 characters.

[If this is kept, the element/feature should probably be renamed from "padding" since it is just as likely to truncate.]

3.2.4. Final processing

The string that results from all replacements is converted to the appropriate RDATA format for the record type. If the conversion fails, the SERVFAIL rcode MUST be set on the response, representing a misconfiguration that the server was unable to perform. [The EDNS extended-error code would be useful here.]

The TTL of each RR generated by a BULK RR is the TTL of the corresponding BULK record itself. [BULK should probably have its own TTL field because using that of the record itself feels like bad design. On the other hand, if BULK is never meant to be queried for directly and only appears in authoritative data, its own TTL is pretty useless normally.]

The class for the RRSet is the class of the BULK RR.

If the generated record type is one that uses domain names in its resource record data, such as CNAME, a relative domain names MUST be fully qualified with the origin domain of the BULK RR.

4. Known Limitations

This section defines known limitations of the BULK resource type.

4.1. Unsupported Nameservers

Authoritative nameservers that do not understand the semantics of the new record type will not be able to deliver the intended answers even when the type appears in their zone data This significantly affects the interoperability of primary versus secondary authorities that are not all running the same software. Adding new RRs which affect handling by authoritative servers, or being unable to add them, is an issue that needs to be explored more thoroughly within dnsop.

5. Security Considerations

Two known security considerations exist for the BULK resource record, DNSSEC and DDOS attack vectors.

5.1. DNSSEC Signature Strategies

DNSSEC was designed to provide validation for DNS resource records, requiring each tuple of owner, class, and type 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 [npn-draft] provides the most flexible.

5.1.1. On-the-fly Signatures

A significant design goal of DNSSEC was to be able to do offline cryptographic signing of zone contents, keeping the key material more secure.

On-the-fly processing requires authoritative nameservers to sign generated records as they are created. Not all authoritative nameserver implementations offer on-the-fly signatures, and even with those that do not all operators will want to keep signing keys online. 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.

One possibly mitigation for addressing the risk of keeping the zone signing key online would be to continue to keep the key for signing positive answers offline and introduce a second key for online signing of negative answers.

No changes to validating resolvers is required to support this solution.

5.1.2. Alternative Signature Scheme

Previous versions of this draft proposed a new signature scheme using a Numeric Pattern Normalization (NPN) RR. It was a method to support offline signatures for BULK records, with the drawback that is required updates to DNSSEC-aware resolvers.

That mechanism is not specific to BULK and has been removed from the current draft. If there is further interest in pursuing it, it can be reopened as a separate draft.

5.1.3. 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 resolvers.

5.2. 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 seconds) in its examples, values between 300 to 900 seconds are likely more appropriate and is RECOMMENDED. What is ultimately deemed appropriate may differ from zone to zone and administrator to administrator.

[I am unclear how this helps DDOS mitigation against anyone at all, and suspect this section should be removed..]

5.3. Implications of Large-Scale DNS Records

The production of such large-scale records in the wild may have some unintended side-effects. These side-effects could be of concern or add unexpected complications to DNS based security offerings or forensic and anti-spam measures. While outside the scope of this document, implementers of technology relying on DNS resource records for critical decision making must take into consideration how the existence of such a volume of records might impact their technology.

Solutions to the magnitude problem for BULK generated RRs are expected be similar if not identical to that of existing wildcard records, the core difference being the resultant RDATA will be unique for each requested Domain Name within its scope.

The authors of this document are confident that by careful consideration, negative_side-effects produced by implementing the features described in this document can be eliminated from any such service or product.

6. Privacy Considerations

The BULK record does not introduce any new privacy concerns to DNS data.

7. IANA Considerations

IANA is requested to assign numbers for the BULK RR.

8. 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.

A special thanks is extended for the kindness, wisdom and technical advice of Robert Whelton (CenturyLink, Inc.) and Gary O'Brien (Secure64 Software Corp).

9. References

9.1. Normative References

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <https://www.rfc-editor.org/info/rfc1034>.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/info/rfc1035>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2181] Elz, R. and R. Bush, "Clarifications to the DNS Specification", RFC 2181, DOI 10.17487/RFC2181, July 1997, <https://www.rfc-editor.org/info/rfc2181>.

[RFC2308] Andrews, M., "Negative Caching of DNS Queries (DNS NCACHE)", RFC 2308, DOI 10.17487/RFC2308, March 1998, <https://www.rfc-editor.org/info/rfc2308>.

[RFC2317] Eidnes, H., de Groot, G., and P. Vixie, "Classless IN-ADDR.ARPA delegation", BCP 20, RFC 2317, DOI 10.17487/RFC2317, March 1998, <https://www.rfc-editor.org/info/rfc2317>.

[RFC3597] Gustafsson, A., "Handling of Unknown DNS Resource Record (RR) Types", RFC 3597, DOI 10.17487/RFC3597, September 2003, <https://www.rfc-editor.org/info/rfc3597>.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, DOI 10.17487/RFC4033, March 2005, <https://www.rfc-editor.org/info/rfc4033>.

[RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, DOI 10.17487/RFC4034, March 2005, <https://www.rfc-editor.org/info/rfc4034>.

[RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, DOI 10.17487/RFC4035, March 2005, <https://www.rfc-editor.org/info/rfc4035>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

9.2. Informative References

[bind-arm] Internet Systems Consortium, "BIND 9 Configuration Reference", 2016, <https://ftp.isc.org/isc/bind9/cur/9.9/doc/arm/Bv9ARM.html>.

[npn-draft] Internet Systems Consortium, "Numeric Pattern Normalization (NPN)", 2019, <https://github.com/ioneyez/npn>.

[RFC7719] Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS Terminology", RFC 7719, DOI 10.17487/RFC7719, December 2015, <https://www.rfc-editor.org/info/rfc7719>.

Appendix A. BULK Examples

A.1. Example 1

```
$ORIGIN 2.10.in-addr.arpa.
@ 86400 IN BULK PTR (
          [0-255].[0-255].[0-255].[0-255].in-addr.arpa.
          pool-${4-1}.example.com.
        )
```

A query received for the PTR of 4.3.2.10.in-addr.arpa will create the references ${1} through ${4} with the first four labels of the query name. The ${4-1} reference in the replacement pattern will then substitute them in reverse with the default delimiter of hyphen between every character and no special field width modifications. The TTL of the BULK RR is used for the generated record, making the response:

4.3.2.10.in-addr.arpa 86400 IN PTR pool-10-2-3-4.example.com.

A.2. Example 2

```
$ORIGIN 2.10.in-addr.arpa.
@ 86400 IN BULK PTR (
          [0-255].[0-255].[0-255].[0-255].in-addr.arpa.
          pool-${2,1|||3}.example.com.
        )
```

Example 2 is similar to Example 1, except that it modifies the replacement pattern. The empty option after the first vertical bar causes no delimiters to be inserted, while the second empty option that would keep the delimiter interval as 1. The latter is relevant because the final value, padding of 3, is applied over each delimiter interval even when no delimiter is used. Not all captures from the substring are required to be used in the response.

The result is that a query for the PTR of 4.3.2.10.in-addr.arpa generates this response:

4.3.2.10.in-addr.arpa 86400 IN PTR pool-003004.example.com.

[Admittedly you can’t do this very effectively without the field width complexity. Is this sort of name common? Does it need

support? Admittedly $GENERATE had the feature, but is that reason enough?]

[Change this to a hex matching example?]

A.3. Example 3

```
$ORIGIN 0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa.
@ 86400 IN BULK PTR (
          <>.<>.<>.<>.<>.<>.<>.<>.<>.<>.<>.<>.<>.<>.<>.<>
          poolAA-${16-8|-|4}.example.com.
        )
```

This example introduces IPv6 where 16 individual nibbles are captured and the last 8 are combined into 2 blocks of 4, separated by a hyphen.

A query for the IP of 2001:db8::dead:beef results in a PTR RR with the value of poolAA-dead-beef.example.com.

A.4. Example 4

```
$ORIGIN example.com.
@ 86400 IN BULK AAAA (
          poolAA-<0-ffff>-<0-ffff>.example.com.
          ${@|.|1}.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa.
        )
```

This example performs the reverse of example 3, where a query of poolAA-dead-beef.example.com captures "dead" and "beef", reversing the nibbles and using a dot (.) as the delimiter to form a valid AAAA record.

A.5. Example 5

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.
@    7200 IN BULK CNAME [0-255].[0-3] ${*|.}.0-3
0-3 86400 IN NS ns1.sub.example.com.
```

A query for the PTR of 25.2.2.10.in-addr.arpa is received and the BULK record with the CNAME Match Type matches all query types. 25 and 2 are captured as references, and joined in the answer by the period (".") character as a delimiter, with ".0-3" then appended

literally and fully qualified by the origin domain. The final synthesized record is:

25.2.2.10.in-addr.arpa 7200 IN CNAME 25.2.0-3.2.10.in-addr.arpa.

[Without $* and options complexity, the pattern to get the same result is just ${1}.{$2}.0-3 which is not really significantly onerous to enter, and slightly less arcane looking to comprehend.]

Authors' Addresses

John Woodworth
CenturyLink, Inc.
4250 N Fairfax Dr
Arlington VA 22203
USA

Email: John.Woodworth@CenturyLink.com

Dean Ballew
CenturyLink, Inc.
2355 Dulles Corner Blvd, Ste 200 300
Herndon VA 20171
USA

Email: Dean.Ballew@CenturyLink.com

Shashwath Bindinganaveli Raghavan
Hughes Network Systems
11717 Exploration Lane
Germantown MD 20876
USA

Email: shashwath.bindinganaveliraghavan@hughes.com

David C Lawrence
Oracle

Email: tale@dd.org