Network Working Group
INTERNET-DRAFT

P. Francis
S. Thomson
Bellcore
June 1993

Use of DNS with Pip

Status of this Memo

Abstract

Pip is an internet protocol intended as the replacement for IP version 4. Pip is a general purpose internet protocol, designed to handle all forseeable internet protocol requirements. This specification describes the use of DNS to support Pip. Because Pip carries IDs and addresses separately, and because Pip Addresses are variable length, DNS must be modified to support Pip. Also, Pip addresses are more likely to change than IP addresses. To enable DNS to still cache aggressively in the presence of address changes, we propose adding functionality to DNS to allow resolvers to ask for later versions of information when previously returned information is found to be out-of-date. In addition to these necessary modifications, we have chosen to add new information to DNS to support transition and to support Pip features, such as policy routing, mobile hosts and routing through Public Data Networks. Later multicast support will be added as well.

Acknowledgements

The authors would like to acknowledge Bob Smart and Garrett Wollman for their initial work on Pip in DNS.

Conventions

All functions in this specification are mandatory.

1. INTRODUCTION

Pip is an internet protocol intended as the replacement for IP version 4. Pip is a general purpose internet protocol, designed to handle all forseeable internet protocol requirements. This specification describes the use of DNS to support Pip. Because Pip carries IDs and addresses separately, and because Pip Addresses are variable length, DNS must be modified to support Pip. Also, Pip addresses are more likely to change than IP addresses. To enable DNS to still cache aggressively in the presence of address changes, we propose adding functionality to DNS to allow resolvers to ask for later versions of information when previously returned information is found to be out-of-date. In addition to these necessary modifications, we have chosen to add new information to DNS to support transition and to support Pip features, such as policy routing, mobile hosts and routing through Public Data Networks. Later multicast support will be added as well.

In spite of this additional functionality, we retain the fundamental DNS paradigm of source-independency (a resource record is returned to the requestor no matter who the requestor is). We also retain the fundamental DNS paradigm that the information stored by DNS does not change often.

This document is meant to be read in conjunction with RFC 1034 and RFC 1035, and is an extension of the latter. The last draft of this document defined new resource records to support the functionality mentioned in the above paragraphs. Some of these definitions have been updated in this draft and they are motivated in more detail. In addition, we discuss not only how DNS is to be used to support the transition of hosts from using IP to using Pip, but also how DNS is to make the transition itself from storing only IP information to storing both IP and Pip information. In this draft, we also propose a

scheme for requesting later versions of resource records using what we call "timestamped queries".

This draft is still rough, and subject to change and expansion. Comments are very welcome.

2. SUMMARY OF PIP DNS INFORMATION

The fundamental information that needs to be stored in DNS to support Pip includes the following:

1. One or more Pip identifiers[1] (Pip IDs) per host. Pip identifiers are 64 bits in length and are used to denote the source and destination in a Pip packet, typically hosts. Initially, there will be one Pip identifier per host. Ultimately, there may be several Pip identifiers per host, as they may be used to denote per-host entities such as processes rather than just the host itself. In any case, Pip identifiers name a host uniquely. At the time of writing, it is an open issue whether Pip identifiers have hierarchical structure. We assume they do have structure here, and that the structure is encoded in the identifier, that is, the Pip identifier is self-describing.

2. Multiple Pip addresses per host[3]. Pip addresses are hierarchical and provider-rooted. They have a provider part and a subscriber part, and each part may have multiple levels of hierarchy. Typically, a host will have one private address (used by hosts within the same subscriber network) and one or more external addresses, one per provider.

In Pip, we choose to use DNS to support transition, mobility, routing through Public Data Networks and policy routing[3]. The information needed is as follows:

1. The address of a Pip/IP translator positioned at the border between the Pip domain and the IP domain. This is used in communications between Pip and IP hosts. When the destination is an IP-only host, the Pip host must address the Pip packet to the Pip/IP translator positioned at the border between the IP domain and the Pip domain. Note that in the case where both source and destination are IP-only with Pip in the middle, the entry Pip/IP

translator must do a DNS query to obtain this information.

2. One or more Mobile Address Servers. Pip supports mobility using non-mobile hosts or routers to keep track of the location of mobile hosts. DNS is used to store the name of the mobile address server for each mobile host. In the case where a host is predominantly mobile (that is, doesn't have a "normal" reachable address), the host may have no Pip addresses stored in DNS. The host's mobile address server may be used directly to determine the current address of the host.

3. Public data network address. This indicates what the public data network address is for hosts connected via a public data network (PDN). This address can be put in an option of the Pip header, and subsequently used by the PDN entry Pip router.

4. Descriptive information associated with the backbone represented by (the provider part of) the Pip Address. The purpose of this information is to allow the source to make a policy decision. The descriptive informatation includes backbone type (e.g. internet) restriction class (e.g. commercial, research), available Type of Service (TOS) (e.g. full-motion video, voice, telnet), and provider name (ANS, AT&T, etc.). The intention of the information is that it be useful and sufficient for the large majority of users, but not necessarily satisfy every possible policy requirement. The information will allow the source to choose the best destination provider given a user or policy preference for the use of a particular source address.

3. NEW CLASS AND RESOURCE RECORD (RR) DEFINITIONS

A new class is introduced to support Pip queries. The class supports all existing non IP-specific resource records in unmodified form, and replaces the definition of IP-specific records, notably the IP address record (type A), with RR types storing a Pip identifier and a Pip address. New RR types are introduced to support new information that has no counterpart in IP, such as backbone descriptors and PDN addresses.

3.1. Storing Pip Identifiers and Address Information

The fundamental additions to DNS are the resource records storing Pip identifiers and addresses. An important requirement for efficiency is that the Pip identifier and addresses for a particular host be retrievable in one DNS query. Another requirement is that Pip addresses should be retrievable in one query given either a hostname or Pip identifier. (Logically, a Pip identifier can be thought of as just another name for a host.) The latter requirement means that we cannot store the Pip identifiers and addresses based on the domain name of a host, since looking up on a Pip ID would then require two queries to DNS, one to map the Pip ID to a hostname (using a PTR query), and one to map a hostname to a set of Pip addresses. Thus, we define Pip identifiers to be indexed by canonical domain name, while Pip addresses are stored per Pip identifier. These mappings have the additional advantage that communication with IP-only hosts during transition is supported transparently. (See Section 3.5). A drawback of this approach is that if hosts are allocated multiple Pip IDs, addresses will be stored redundantly, once for each Pip ID. If this is a problem, we can introduce canonical Pip IDs and aliases as is done for host domain names. We assume for now that hosts only have one Pip ID.

Thus, two resource records are needed: one to store a Pip identifier and one to store a Pip address. The IP type A RR is replaced by a resource record that stores a Pip identifier instead of an IP address. We define a Pip type A query to return the Pip ID as an answer and the Pip addresses associated with the Pip ID as additional information. The type A resource record for the Pip class has the following format (more detailed definitions of the data fields of this and other Pip RRs are given in the Appendix):

<owner> <ttl> Pip A <Pip ID>

The <owner> of the RR is the domain name of a host. In a RR, <Pip ID> is a 64-bit word. In the master file, a Pip identifier is represented in its textual format, which consists of eight 2-digit hex numbers separated by dots[1].

The Pip address RR for the Pip class is defined as follows:

<owner> <ttl> Pip ADDR <subtype> <data>

The <owner> is an inverse lookup name based on a Pip identifier.

(Inverse lookup domains are discussed in the following section.) Several types of address information can be stored in this resource record. The <subtype> field is a 16-bit integer and is used to distinguish between them. The most common data field comprises a Pip address, denoted by subtype 1. Pip addresses are encoded in RRs as a sequence of 32-bit words, as described more fully in the Appendix. In the master file, a subtype is written as an integer, and the Pip address in its textual format, which is a list of numbers, separated by commas (to indicated changes in metalevel) and dots (to indicate changes in level of hierarchy or different providers in a route fragment)[2].

The above definitions are used to store the fundamental information: Pip identifiers and addresses. As already mentioned, we choose to use DNS to store other types of addressing information too, to support mobility and use of Public Data Networks. Moreover, we would like all address information to be returned in one query. Thus, the ADDR field has been defined to include a subtype field as shown above to enable an ADDR query to return more information than just "ordinary" Pip addresses. (If, after experimentation, this is found to be unnecessary, the subtype field should be removed and separate resource record types used.) A subtype of 2 indicates that the data field is the Pip ID of a mobile address server for the <owner>. Type ADDR additional section processing is performed on the Pip ID of the mobile address server to yield its addresses. A mobile address server should not itself be mobile, but if the database indicates that it is, this should be ignored by the name server when answering an ADDR query (to avoid circularities). To support the storage of PDN addresses, Pip addresses are defined to cause additional section processing to yield a PDN address. PDN resource records are discussed below.

3.2. Storing Public Data Network Addresses

The PDNA RR has the following form:

<owner> <ttl> <class> PDNA <PDN-addr>

PDNA records are stored per (subscriber part of) a Pip address. The <owner> is an inverse lookup name based on a Pip address. (Inverse lookup domains are discussed in Section 3.4). In a master file, the PDN address is written as a string of decimal digits. This record

causes no additional section processing.

3.3. Storing Backbone Information

A new resource record is added to store information about a provider, called a backbone descriptor (type BBD). At the time of writing, the record is expected to contain at least the following 4 fields for each provider: the name, type, usage restriction classes and types of service available. Since it is not clear at this stage precisely what backbone information needs to be stored, we define the resource record to be extensible by making the fields self-describing.

The BBD RR has the following form:

<owner> <ttl> <class> BBD <mnemonic field0> <mnemonic field1> <...>

BBD records are stored per (top-level component of) a Pip address. The <owner> is an inverse lookup name based on a Pip address. See below for address domains. In a RR, <mnemonic> is an 8-bit integer and <field> a <character-string>. In a master file, the BBD fields are prefixed by a mnemonic integer which indicates the type of the field that follows. The fields are character strings. This record causes no additional section processing.

3.4. PIP-ADDR.ARPA and PIP-ID.ARPA domains

The two special domains, PIP-ADDR.ARPA and PIP-ID.ARPA are used to map Pip addresses and Pip identifiers to regular domain names, respectively. As with IP addresses, the PTR resource record type is used to query this mapping. The PTR query type causes no additional section processing.

Pip address domain names are represented by a sequence of hex labels in reverse order with the suffix PIP-ADDR.ARPA. The labels correspond to each component of an address and are separated by dots and commas, in the same way as defined for the textual representation of Pip addresses[2]. Pip identifier domain names are represented by hex labels in reverse order, with the suffix PIP-ID.ARPA. These labels are determined from the hierarchical structure of the Pip

identifier[1]. Identifier labels are separated by dots.

The PIP-ID.ARPA domain is also used to map Pip identifiers to Pip addressing information (ADDR type), while the PIP-ADDR.ARPA domain is also used to map Pip addresses to backbone descriptors (BBD type) and PDN attachment point addresses (PDNA type).

3.5. DNS Support for IP/Pip transition.

NOTE: This section assumes IPAE transition. A new transition scheme is being proposed for Pip, and this will be taken account of in a later draft.

We mentioned in Section 2 that we choose to use DNS to store the addresses of translating routers for IP hosts. The name service can be used to support IP/Pip host transition by assigning IP hosts special "IP-only" Pip IDs (identifiers with a well-known prefix indicating the named host is IP-only, and containing the IP address in the low-order 32 bits) and using the address of the appropriate translating router as their "normal" Pip address. Note that no extra RR types are needed to perform this mapping; the translator addresses are stored in place of a host's Pip addresses, transparently to DNS and the resolver. (If the name server or resolver does need to know that a host is IP for any reason, this can be determined from the special prefix of the Pip identifier given to the host.) Also, note that storing the addresses of the translator as the Pip addresses of IP hosts (rather than storing the Pip ID of the translator and having that map onto the translator addresses) does not duplicate information in the database. In Pip, the addresses used for translation are different from those used to address the translating machine as a destination host.

4. TRANSITION FROM IP NAME SERVICE TO PIP NAME SERVICE

During transition, we expect that zones with a Pip host have a Pip name server. (A Pip name server is one that is authoritative for Pip information. Note that this does not necessarily mean the name server

can communicate using Pip.) However, we do not expect zones that contain only IP hosts to deploy a Pip name server, at least not immediately. Moreover, we do not think it practical to expect other sites to maintain Pip name servers for IP sites. (As stated in the section above, servers are needed to store Pip addresses of translating routers for IP zones, but we expect that the amount of information needed for this purpose is significantly smaller than the database of each IP zone.) Thus, the Pip naming tree will not be complete for some period of time. This has a number of repercussions for resolvers.

First, recursive resolvers (typically, found in name servers) must have access to both Pip and IP name servers and must be able to determine which name servers support Pip and which IP. Class information is provided in name server (NS) resource records. In order to obtain NS RRs of a particular class, however, a recursive resolver must ask parent name servers the appropriate class of query. Such information must be gleaned by trial and error. (NS resource records are not usually explicitly requested - they are returned as a by-product of a query for some other resource record to a non-authoritative server. When a Pip query is made to a non-authoritative name server, Pip NS RRs will be returned in the authoritative section of a response. When an IP query is made, IP NS RRs will be returned in the authoritative section of a response.) Note that in general an address request might require 3 queries if the host being looked up is IP-only, assuming a Pip class query is made first: failure of a Pip class query will require a separate IP class query. Since we are using DNS to store addresses of translating routers for IP hosts, another lookup is required to get the Pip address of the translating router.

The efficiency of recursive resolvers can be improved in a number of ways. The fact that a zone does NOT support a certain class should be cached and timed out in the same way as name server resource records are. This prevents future queries during the time-to-live period from having to rediscover this (negative) information. Also, for IP hosts in zones with Pip name servers, the number of queries can be reduced by assigning the IP hosts special Pip identifiers (as explained in Section 3.5) and storing these Pip identifiers and the addresses of the appropriate translating router in the Pip database. Pip address queries for these hosts will succeed in one query.

The fact that the Pip naming tree is not complete also affects the efficiency of operation of stub resolvers, typically found in Pip hosts. (A stub resolver relies on a name server (with a recursive

resolver) to perform resolution as it is, by definition, incapable of following referrals. It usually does not cache any information.) Stub resolvers are not as amenable to optimization as they do not in general cache name server RRs and thus do not have any information to indicate what class of query to make when looking up the address of a host. A Pip stub resolver can be assisted by having the name server it relies on to resolve queries to decide on which class of query to perform. As explained above, a name server can potentially do a much better job of determining which class of query to make, since it caches name server records and knows what classes of requests a name server supports, or at least knows how to find out in the absence of such information. Hence, it is more efficient for stub resolvers on Pip hosts to always make Pip class queries and have the name servers decide what class of query to make in the event of a referral and perform any translation necessary to return a valid Pip response. When the name server is referred to a Pip name server, no translation is required. When the name server is referred to an IP-only name server, the query must be translated into an IP query. On receiving a response from the IP name server, the query must be translated back to that of the original class, and the resource records formatted to appear as valid Pip records. In particular, all IP addresses must be translated to appear as Pip identifiers with their associated Pip addresses. To achieve this, the IP hosts are given special Pip identifiers (based on their IP addresses). These identifiers are then looked up separately to retrieve the corresponding Pip addresses for the IP hosts. The addresses are the addresses of the IP hosts' translating routers.

5. TIMESTAMPED QUERIES

In Pip, hosts are able to tell when the address of a destination host is out-of-date. This will typically happen when a destination changes provider (its address prefix will change). A host is able to tell that the destination address has changed since, in Pip, the provider's routers are configured to return a PCMP Packet Not Delivered Message for an address with an old prefix for some period of time after the change occurs[4]. To enable DNS to still allow high enough time-to-live values for efficient operation, but yet still maintain connectivity with hosts whose addresses have changed, we propose that DNS be enhanced to allow a resolver to ask for later versions of resource records than it currently has cached.

The proposed scheme involves assigning version numbers to resource records, which are incremented whenever the information is changed. In a query, a resolver specifies the version number of the RRs requested, and the name server responds with RRs that have version numbers greater than that specified. If the name server does not have the data locally or has an old version, it must refer the request to name servers "closer" to the answer in the normal way. If the name server is authoritative for RRs requested, it always answers the question independent of the version number requested.

The version numbers could be timestamps, in which case they may have additional semantic significance, e.g. they may indicate when the RR was added to the database or the last time the RR was modified. Alternatively, they could be sequence numbers similar to zone serial numbers. Other proposed extensions to DNS also require RR version information, such as incremental zone transfers mentioned recently on the namedroppers mailing list. This scheme requires that version information be comparable with zone serial numbers. If both timestamped queries and incremental zone transfers are to be added to DNS, then it would make sense to use a common definition. Whatever the form of version numbers chosen, there must be a way for a requestor to ask for any version of a RR, that is, the first version found. Such a query would be used by a resolver when requesting a resource record for the first time.

5.1. Changes to Message Format

According to RFC 1035, there are three sets of queries that can be made to DNS, standard queries, inverse queries and status queries. When a query is made, a field in the header, called the operation code, indicates what kind of query it is. To enable timestamped queries, a new version of the standard query is proposed.

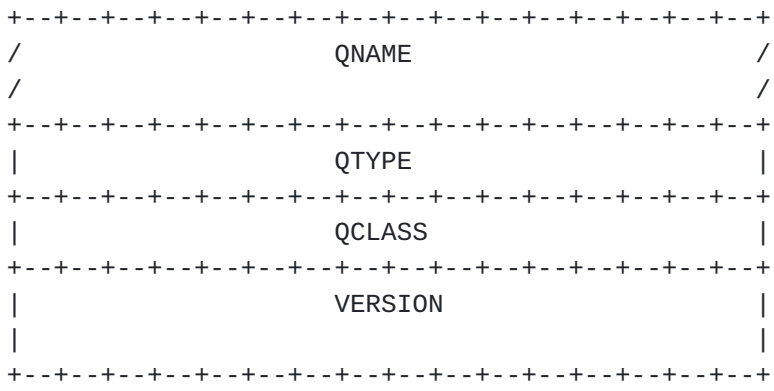
The following new operation code is thus defined:

Op Code	Value and Meaning
QUERY2	4 Timestamped queries

The format of messages will need to change to support timestamped queries. The header will remain the same to maintain backwards

compatibility, but the new operation code will indicate that the format of the question section and the definitions of RRs have been extended.

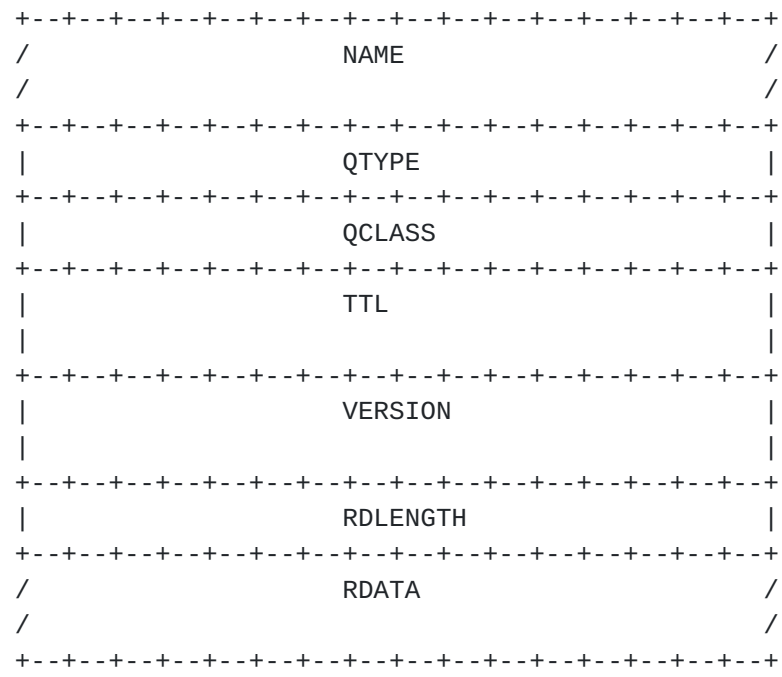
The current definition of the question section consist of three fields: the name of the domain being queried, the type of the query (typically a RR type) and the class of the query (typically Internet). An extra field will be added to include the version information. The precise form of this field is not yet defined, but it will likely be a 32-bit field. The question section has the following format (original from RFC 1035):



where:

QNAME	<domain-name>
QTYPE	A 16-bit field specifying the type of the query
QCLASS	A 16-bit field specifying the class of the query
VERSION	A 32-bit field specifying that the answer must be a later version than that specified. A version number of zero is used to indicate that a version number is not known or that a resource record with any version number is requested.

RRs will also be extended to include the version information field. Currently, RRs consists of the name of the domain the record describes, the type of the record, its class, the time-to-live field, which indicates how long the RR can be cached, as well as the data and its length. A 32-bit version field could follow the TTL field as follows (original RR format from RFC 1035):



where:

NAME	A <domain-name> to which this RR pertains
QTYPE	A 16-bit field specifying the type of the RR. This field specifies the meaning of the data in the RDATA field.
QCLASS	A 16-bit field specifying the class of the data
TTL	A 32-bit unsigned integer that specifies the time interval (in seconds) that the resource record may be cached before it should be discarded.
VERSION	A 32-bit field specifying the version number of the data in the RDATA field.
RDLENGTH	An unsigned 16-bit integer specifying the length of the octets of the RDATA field.
RDATA	A variable-length string of octets that describes the resource. The format of this information varies according to the TYPE and CLASS of the resource record.

6. APPENDIX

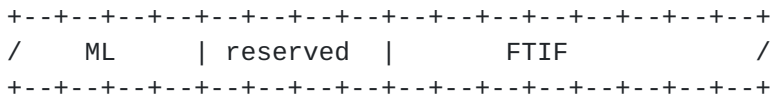
The Pip class is chosen to have value 5. The following types of resource records are added to DNS (or redefined) to store Pip information. The type A and ADDR resource records are Pip-specific. The

other resource records can be used by any class.

Type	Value and Meaning	
A	0	Redefined to store Pip identifier
ADDR	64	Different types of address information, such as Pip addresses and names of mobile address servers
BBD	65	Backbone descriptor
PDNA	66	PDN attachment point address

6.1. New Resource Record (RR) data definitions

Below we define the data fields for each of the new Pip resource records. Some data fields are defined in terms of <domain-name>s and <character-string>s. These two field types have the same definitions as stated in RFC 1035. We also define an <octet-string>, which is similar to a <character-string>, but consists of a string of unsigned 8-bit fields only. Two other new fields include the Pip identifier and the Pip address. A <pip-identifier> is a 64-bit field containing a Pip ID represented in its modified ASN.1 notation[1]. A <pip-address> consists of a sequence of 16-bit numbers called FTIFs (Forwarding Table Index Fields), each representing a level of the hierarchy, or a provider in a route fragment. A Pip address is represented in a RR by a sequence of 32-bit words, each containing an FTIF, and the metalevel, if appropriate, as shown below:

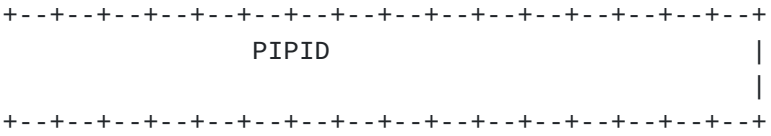


where:

ML	An 8-bit field containing the metalevel of this FTIF. This field is only set in the first FTIF of each metalevel (zero otherwise).
<reserved>	An 8-bit field reserved for later use. Should be zero.
FTIF	A 16-bit field containing the FTIF value.

The data format for Pip identifiers (type A), addresses (type ADDR), backbone descriptors (type BBD) and PDN addresses (type PDNA) follow:

A data format

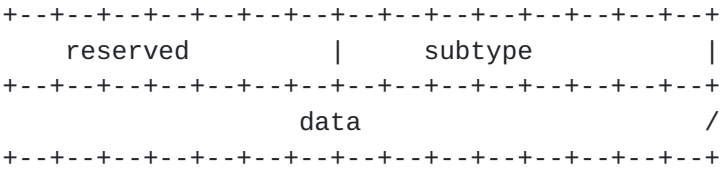


where:

PIPID A <pip-identifier> associated with the specified <domain-name>

Type A RRs cause type ADDR additional section processing.

ADDR data format



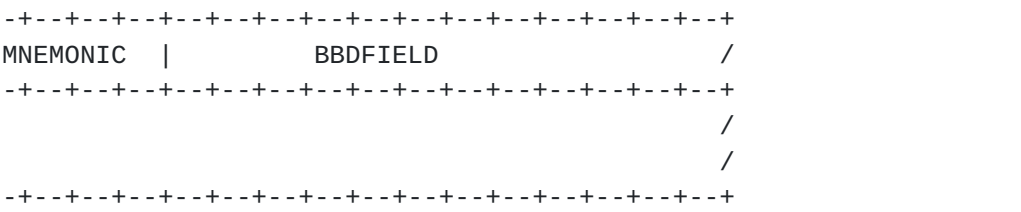
where:

<reserved> A 16-bit field reserved for later use. Should be zero.
<subtype> A 16-bit integer indicating the type of data to be found in the data field.
<data> In the case of subtype 1, a <pip-address> of the specified owner.
In the case of subtype 2, the <pip-identifier> of the mobile address server for the specified owner.

ADDR queries do cause additional section processing. In the case of subtype 1, PDNA queries are performed on Pip addresses. In the case of subtype 2, ADDR queries are performed on Pip identifiers. Note that an ADDR query will not be performed again as part of additional section processing if the mobile server itself has a

mobile address server.

BBD data format



where:

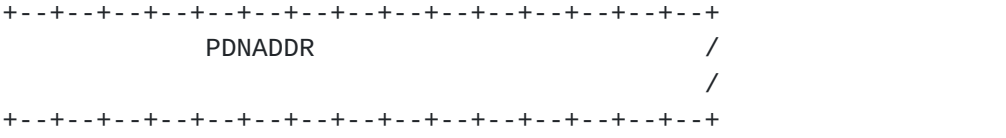
MNEMONIC An 8-bit field indicating the type of information contained in BBDFIELD.

BBDFIELD A <character-string> that contains information of the corresponding type.

Standard values for mnemonics must still be defined.

This record causes no additional section processing.

PDNA data format



where:

PDNADDR An <octet-string> that specifies the public data network attachment point address in encoded form as a sequence of 4-bit digits.

This record causes no additional section processing.

References

[1] Francis, "Pip Identifiers", Internet-Draft
[2] Francis, "Pip Address Conventions", Internet-Draft
[3] Francis, "Pip Near-term Architecture", Internet-Draft
[4] Francis, "Pip Host Operation", Internet-Draft