NAME

Net::DNS::Packet – DNS protocol packet

SYNOPSIS

```
use Net::DNS::Packet;

$query = new Net::DNS::Packet( 'example.com', 'MX', 'IN' );

$reply = $resolver->send( $query );
```

DESCRIPTION

A Net::DNS::Packet object represents a DNS protocol packet.

METHODS

new

```
$packet = new Net::DNS::Packet( 'example.com' );
$packet = new Net::DNS::Packet( 'example.com', 'MX', 'IN' );

$packet = new Net::DNS::Packet();
```

If passed a domain, type, and class, **new()** creates a Net::DNS::Packet object which is suitable for making a DNS query for the specified information. The type and class may be omitted; they default to A and IN.

If called with an empty argument list, **new()** creates an empty packet.

```
$packet = new Net::DNS::Packet( \$data );
$packet = new Net::DNS::Packet( \$data, 1 );        # debug
```

If passed a reference to a scalar containing DNS packet data, a new packet object is created by decoding the data. The optional second boolean argument enables debugging output.

Returns undef if unable to create a packet object.

Decoding errors, including data corruption and truncation, are collected in the $@ ($EVAL_ERROR) variable.

```
( $packet, $length ) = new Net::DNS::Packet( \$data );
```

If called in array context, returns a packet object and the number of octets successfully decoded.

Note that the number of RRs in each section of the packet may differ from the corresponding header value if the data has been truncated or corrupted during transmission.

data

```
$data = $packet->data;
$data = $packet->data( $size );
```

Returns the packet data in binary format, suitable for sending as a query or update request to a nameserver.

Truncation may be specified using a non-zero optional size argument.

header

```
$header = $packet->header;
```

Constructor method which returns a Net::DNS::Header object which represents the header section of the packet.

edns

```
$edns    = $packet->edns;
$version = $edns->version;
$UDPsize = $edns->size;
```

Auxiliary function which provides access to the EDNS protocol extension OPT RR.

reply

```
$reply = $query->reply( $UDPmax );
```

Constructor method which returns a new reply packet.

The optional UDPsize argument is the maximum UDP packet size which can be reassembled by the local network stack, and is advertised in response to an EDNS query.

question, zone

```
@question = $packet->question;
```

Returns a list of Net::DNS::Question objects representing the question section of the packet.

In dynamic update packets, this section is known as **zone()** and specifies the DNS zone to be updated.

answer, pre, prerequisite

```
@answer = $packet->answer;
```

Returns a list of Net::DNS::RR objects representing the answer section of the packet.

In dynamic update packets, this section is known as **pre()** or **prerequisite()** and specifies the RRs or RRsets which must or must not preexist.

authority, update

```
@authority = $packet->authority;
```

Returns a list of Net::DNS::RR objects representing the authority section of the packet.

In dynamic update packets, this section is known as **update()** and specifies the RRs or RRsets to be added or deleted.

additional

```
@additional = $packet->additional;
```

Returns a list of Net::DNS::RR objects representing the additional section of the packet.

print

```
$packet->print;
```

Prints the entire packet to the currently selected output filehandle using the master file format mandated by RFC1035.

string

```
print $packet->string;
```

Returns a string representation of the packet.

from

```
print "packet received from ", $packet->from, "\n";
```

Returns the IP address from which this packet was received. This method will return undef for user-created packets.

size

```
print "packet size: ", $packet->size, " octets\n";
```

Returns the size of the packet in octets as it was received from a nameserver. This method will return undef for user-created packets (use length($packet->data) instead).

push

```
$ancount = $packet->push( prereq => $rr );
$nscount = $packet->push( update => $rr );
$arcount = $packet->push( additional => $rr );

$nscount = $packet->push( update => $rr1, $rr2, $rr3 );
$nscount = $packet->push( update => @rr );
```

Adds RRs to the specified section of the packet.

Returns the number of resource records in the specified section.

Section names may be abbreviated to the first three characters.

unique_push

```
$ancount = $packet->unique_push( prereq => $rr );
$nscount = $packet->unique_push( update => $rr );
$arcount = $packet->unique_push( additional => $rr );

$nscount = $packet->unique_push( update => $rr1, $rr2, $rr3 );
$nscount = $packet->unique_push( update => @rr );
```

Adds RRs to the specified section of the packet provided that the RRs are not already present in the same section.

Returns the number of resource records in the specified section.

Section names may be abbreviated to the first three characters.

pop

```
my $rr = $packet->pop( 'pre' );
my $rr = $packet->pop( 'update' );
my $rr = $packet->pop( 'additional' );
```

Removes a single RR from the specified section of the packet.

sign_tsig

```
$query = Net::DNS::Packet->new( 'www.example.com', 'A' );

$query->sign_tsig(
            'Khmac-sha512.example.+165+01018.private',
            fudge => 60
            );

$reply = $res->send( $query );

$reply->verify( $query ) || die $reply->verifyerr;
```

Attaches a TSIG resource record object, which will be used to sign the packet (see RFC 2845).

The TSIG record can be customised by optional additional arguments to **sign_tsig()** or by calling the appropriate Net::DNS::RR::TSIG methods.

If you wish to create a TSIG record using a non-standard algorithm, you will have to create it yourself. In all cases, the TSIG name must uniquely identify the key shared between the parties, and the algorithm name must identify the signing function to be used with the specified key.

```
$tsig = Net::DNS::RR->new(
            name            => 'tsig.example',
            type            => 'TSIG',
            algorithm       => 'custom-algorithm',
            key             => '<base64 key text>',
            sig_function    => sub {
                                    my ($key, $data) = @_;
                                          ...
                                }
            );

$query->sign_tsig( $tsig );
```

The historical simplified syntax is still available, but additional options can not be specified.

```
$packet->sign_tsig( $key_name, $key );
```

The response to an inbound request is signed by presenting the request in place of the key parameter.

```
$response = $request->reply;
$response->sign_tsig( $request, @options );
```

Multi-packet transactions are signed by chaining the **sign_tsig()** calls together as follows:

```
$opaque  =  $packet1->sign_tsig( 'Kexample.+165+13281.private' );
$opaque  =  $packet2->sign_tsig( $opaque );
            $packet3->sign_tsig( $opaque );
```

The opaque intermediate object references returned during multi-packet signing are not intended to be accessed by the end-user application. Any such access is expressly forbidden.

Note that a TSIG record is added to every packet; this implementation does not support the suppressed signature scheme described in RFC2845.

verify and verifyerr

```
$packet->verify()           || die $packet->verifyerr;
$reply->verify( $query )    || die $reply->verifyerr;
```

Verify TSIG signature of packet or reply to the corresponding query.

```
$opaque  =  $packet1->verify( $query ) || die $packet1->verifyerr;
$opaque  =  $packet2->verify( $opaque );
$verifed =  $packet3->verify( $opaque ) || die $packet3->verifyerr;
```

The opaque intermediate object references returned during multi-packet **verify()** will be undefined (Boolean false) if verification fails. Access to the object itself, if it exists, is expressly forbidden. Testing at every stage may be omitted, which results in a BADSIG error on the final packet in the absence of more specific information.

sign_sig0

SIG0 support is provided through the Net::DNS::RR::SIG class. The requisite cryptographic components are not integrated into Net::DNS but reside in the Net::DNS::SEC distribution available from CPAN.

```
$update = new Net::DNS::Update('example.com');
$update->push( update => rr_add('foo.example.com A 10.1.2.3'));
$update->sign_sig0('Kexample.com+003+25317.private');
```

Execution will be terminated if Net::DNS::SEC is not available.

verify SIG0

```
$packet->verify( $keyrr )           || die $packet->verifyerr;
$packet->verify( [$keyrr, ...] )    || die $packet->verifyerr;
```

Verify SIG0 packet signature against one or more specified KEY RRs.

sigrr

```
$sigrr = $packet->sigrr() || die 'unsigned packet';
```

The sigrr method returns the signature RR from a signed packet or undefined if the signature is absent.

truncate

The truncate method takes a maximum length as argument and then tries to truncate the packet and set the TC bit according to the rules of RFC2181 Section 9.

The smallest length limit that is honoured is 512 octets.

COPYRIGHT

LICENSE

SEE ALSO

perl, Net::DNS, Net::DNS::Update, Net::DNS::Header, Net::DNS::Question, Net::DNS::RR, Net::DNS::RR::TSIG, RFC1035 Section 4.1, RFC2136 Section 2, RFC2845