NAME

Net::Ping - check a remote host for reachability

SYNOPSIS

```perl
use Net::Ping;

$p = Net::Ping->new();
print "$host is alive.\n" if $p->ping($host);
$p->close();


$p = Net::Ping->new("icmp");
$p->bind($my_addr); # Specify source interface of pings
foreach $host (@host_array)
{
    print "$host is ";
    print "NOT " unless $p->ping($host, 2);
    print "reachable.\n";
    sleep(1);
}
$p->close();


$p = Net::Ping->new("tcp", 2);
# Try connecting to the www port instead of the echo port
$p->{port_num} = getservbyname("http", "tcp");
while ($stop_time > time())
{
    print "$host not reachable ", scalar(localtime()), "\n"
        unless $p->ping($host);
    sleep(300);
}
undef($p);


# Like tcp protocol, but with many hosts
$p = Net::Ping->new("syn");
$p->{port_num} = getservbyname("http", "tcp");
foreach $host (@host_array) {
  $p->ping($host);
}
while (($host,$rtt,$ip) = $p->ack) {
  print "HOST: $host [$ip] ACKed in $rtt seconds.\n";
}


# High precision syntax (requires Time::HiRes)
$p = Net::Ping->new();
$p->hires();
($ret, $duration, $ip) = $p->ping($host, 5.5);
printf("$host [ip: $ip] is alive (packet return time: %.2f ms)\n", 1000
 * $duration)
    if $ret;
$p->close();


# For backward compatibility
print "$host is alive.\n" if pingecho($host);
```

DESCRIPTION

This module contains methods to test the reachability of remote hosts on a network. A ping object is first created with optional parameters, a variable number of hosts may be pinged multiple times and then the connection is closed.

You may choose one of six different protocols to use for the ping. The "tcp" protocol is the default. Note that a live remote host may still fail to be pingable by one or more of these protocols. For example, www.microsoft.com is generally alive but not "icmp" pingable.

With the "tcp" protocol the ping() method attempts to establish a connection to the remote host's echo port. If the connection is successfully established, the remote host is considered reachable. No data is actually echoed. This protocol does not require any special privileges but has higher overhead than the "udp" and "icmp" protocols.

Specifying the "udp" protocol causes the ping() method to send a udp packet to the remote host's echo port. If the echoed packet is received from the remote host and the received packet contains the same data as the packet that was sent, the remote host is considered reachable. This protocol does not require any special privileges. It should be borne in mind that, for a udp ping, a host will be reported as unreachable if it is not running the appropriate echo service. For Unix-like systems see *inetd(8)* for more information.

If the "icmp" protocol is specified, the ping() method sends an icmp echo message to the remote host, which is what the UNIX ping program does. If the echoed message is received from the remote host and the echoed information is correct, the remote host is considered reachable. Specifying the "icmp" protocol requires that the program be run as root or that the program be setuid to root.

If the "external" protocol is specified, the ping() method attempts to use the `Net::Ping::External` module to ping the remote host. `Net::Ping::External` interfaces with your system's default `ping` utility to perform the ping, and generally produces relatively accurate results. If `Net::Ping::External` if not installed on your system, specifying the "external" protocol will result in an error.

If the "syn" protocol is specified, the ping() method will only send a TCP SYN packet to the remote host then immediately return. If the syn packet was sent successfully, it will return a true value, otherwise it will return false. NOTE: Unlike the other protocols, the return value does NOT determine if the remote host is alive or not since the full TCP three-way handshake may not have completed yet. The remote host is only considered reachable if it receives a TCP ACK within the timeout specifed. To begin waiting for the ACK packets, use the ack() method as explained below. Use the "syn" protocol instead the "tcp" protocol to determine reachability of multiple destinations simultaneously by sending parallel TCP SYN packets. It will not block while testing each remote host. demo/fping is provided in this distribution to demonstrate the "syn" protocol as an example. This protocol does not require any special privileges.

Functions

Net::Ping->new([$proto [, $def_timeout [, $bytes [, $device [, $tos]]]]]);

Create a new ping object. All of the parameters are optional. $proto specifies the protocol to use when doing a ping. The current choices are "tcp", "udp", "icmp", "stream", "syn", or "external". The default is "tcp".

If a default timeout ($def_timeout) in seconds is provided, it is used when a timeout is not given to the ping() method (below). The timeout must be greater than 0 and the default, if not specified, is 5 seconds.

If the number of data bytes ($bytes) is given, that many data bytes are included in the ping packet sent to the remote host. The number of data bytes is ignored if the protocol is "tcp". The minimum (and default) number of data bytes is 1 if the protocol is "udp" and 0 otherwise. The maximum number of data bytes that can be specified is 1024.

If $device is given, this device is used to bind the source endpoint before sending the ping

packet. I beleive this only works with superuser privileges and with udp and icmp protocols at this time.

If $tos is given, this ToS is configured into the soscket.

$p->ping($host [, $timeout]);

> Ping the remote host and wait for a response. $host can be either the hostname or the IP number of the remote host. The optional timeout must be greater than 0 seconds and defaults to whatever was specified when the ping object was created. Returns a success flag. If the hostname cannot be found or there is a problem with the IP number, the success flag returned will be undef. Otherwise, the success flag will be 1 if the host is reachable and 0 if it is not. For most practical purposes, undef and 0 and can be treated as the same case. In array context, the elapsed time as well as the string form of the ip the host resolved to are also returned. The elapsed time value will be a float, as retuned by the Time::HiRes::time() function, if hires() has been previously called, otherwise it is returned as an integer.

$p->source_verify({ 0 | 1 });

> Allows source endpoint verification to be enabled or disabled. This is useful for those remote destinations with multiples interfaces where the response may not originate from the same endpoint that the original destination endpoint was sent to. This only affects udp and icmp protocol pings.

> This is enabled by default.

$p->service_check({ 0 | 1 });

> Set whether or not the connect behavior should enforce remote service availability as well as reachability. Normally, if the remote server reported ECONNREFUSED, it must have been reachable because of the status packet that it reported. With this option enabled, the full three-way tcp handshake must have been established successfully before it will claim it is reachable. NOTE: It still does nothing more than connect and disconnect. It does not speak any protocol (i.e., HTTP or FTP) to ensure the remote server is sane in any way. The remote server CPU could be grinding to a halt and unresponsive to any clients connecting, but if the kernel throws the ACK packet, it is considered alive anyway. To really determine if the server is responding well would be application specific and is beyond the scope of Net::Ping. For udp protocol, enabling this option demands that the remote server replies with the same udp data that it was sent as defined by the udp echo service.

> This affects the "udp", "tcp", and "syn" protocols.

> This is disabled by default.

$p->tcp_service_check({ 0 | 1 });

> Depricated method, but does the same as service_check() method.

$p->hires({ 0 | 1 });

> Causes this module to use Time::HiRes module, allowing milliseconds to be returned by subsequent calls to ping().

> This is disabled by default.

$p->bind($local_addr);

> Sets the source address from which pings will be sent. This must be the address of one of the interfaces on the local host. $local_addr may be specified as a hostname or as a text IP address such as "192.168.1.1".

> If the protocol is set to "tcp", this method may be called any number of times, and each call to the ping() method (below) will use the most recent $local_addr. If the protocol is "icmp" or "udp", then bind() must be called at most once per object, and (if it is called at all) must be called before the first call to ping() for that object.

$p->open($host);

> When you are using the "stream" protocol, this call pre-opens the tcp socket. It's only necessary to do this if you want to provide a different timeout when creating the connection, or remove the overhead of establishing the connection from the first ping. If you don't call open(), the connection is automatically opened the first time ping() is called. This call simply does nothing if you are using any protocol other than stream.

$p->ack([$host]);

> When using the "syn" protocol, use this method to determine the reachability of the remote host. This method is meant to be called up to as many times as ping() was called. Each call returns the host (as passed to ping()) that came back with the TCP ACK. The order in which the hosts are returned may not necessarily be the same order in which they were SYN queued using the ping() method. If the timeout is reached before the TCP ACK is received, or if the remote host is not listening on the port attempted, then the TCP connection will not be established and ack() will return undef. In list context, the host, the ack time, and the dotted ip string will be returned instead of just the host. If the optional $host argument is specified, the return value will be partaining to that host only. This call simply does nothing if you are using any protocol other than syn.

$p->nack($failed_ack_host);

> The reason that host $failed_ack_host did not receive a valid ACK. Useful to find out why when ack($fail_ack_host) returns a false value.

$p->close();

> Close the network connection for this ping object. The network connection is also closed by "undef $p". The network connection is automatically closed if the ping object goes out of scope (e.g. $p is local to a subroutine and you leave the subroutine).

pingecho($host [, $timeout]);

> To provide backward compatibility with the previous version of Net::Ping, a pingecho() subroutine is available with the same functionality as before. pingecho() uses the tcp protocol. The return values and parameters are the same as described for the ping() method. This subroutine is obsolete and may be removed in a future version of Net::Ping.

NOTES

There will be less network overhead (and some efficiency in your program) if you specify either the udp or the icmp protocol. The tcp protocol will generate 2.5 times or more traffic for each ping than either udp or icmp. If many hosts are pinged frequently, you may wish to implement a small wait (e.g. 25ms or more) between each ping to avoid flooding your network with packets.

The icmp protocol requires that the program be run as root or that it be setuid to root. The other protocols do not require special privileges, but not all network devices implement tcp or udp echo.

Local hosts should normally respond to pings within milliseconds. However, on a very congested network it may take up to 3 seconds or longer to receive an echo packet from the remote host. If the timeout is set too low under these conditions, it will appear that the remote host is not reachable (which is almost the truth).

Reachability doesn't necessarily mean that the remote host is actually functioning beyond its ability to echo packets. tcp is slightly better at indicating the health of a system than icmp because it uses more of the networking stack to respond.

Because of a lack of anything better, this module uses its own routines to pack and unpack ICMP packets. It would be better for a separate module to be written which understands all of the different kinds of ICMP packets.

INSTALL

The latest source tree is available via cvs:

```
  cvs -z3 -q -d
:pserver:anonymous@cvs.roobik.com.:/usr/local/cvsroot/freeware checkout
Net-Ping
  cd Net-Ping
```

The tarball can be created as follows:

```
  perl Makefile.PL ; make ; make dist
```

The latest Net::Ping release can be found at CPAN:

```
  $CPAN/modules/by-module/Net/
```

1) Extract the tarball

```
  gtar -zxvf Net-Ping-xxxx.tar.gz
  cd Net-Ping-xxxx
```

2) Build:

```
  make realclean
  perl Makefile.PL
  make
  make test
```

3) Install

```
  make install
```

Or install it RPM Style:

```
  rpm -ta SOURCES/Net-Ping-xxxx.tar.gz


  rpm -ih RPMS/noarch/perl-Net-Ping-xxxx.rpm
```

BUGS

For a list of known issues, visit:

https://rt.cpan.org/NoAuth/Bugs.html?Dist=Net-Ping

To report a new bug, visit:

https://rt.cpan.org/NoAuth/ReportBug.html?Queue=Net-Ping

AUTHORS

```
  Current maintainer:
    bbb@cpan.org (Rob Brown)


  External protocol:
    colinm@cpan.org (Colin McMillen)


  Stream protocol:
```

```
    bronson@trestle.com (Scott Bronson)


  Original pingecho():
    karrer@bernina.ethz.ch (Andreas Karrer)
    pmarquess@bfsec.bt.co.uk (Paul Marquess)


  Original Net::Ping author:
    mose@ns.ccsn.edu (Russell Mosemann)
```

COPYRIGHT

$Id: Ping.pm,v 1.86 2003/06/27 21:31:07 rob Exp $