Global Knowledge ™

Expert Reference Series of White Papers

How Does Ping Really Work?

How Does Ping Really Work?

George Mays, Global Knowledge Course Director, CCISP, CCNA, A+, Network+, Security+, I-Net+

Introduction

Ping is a basic Internet program that most of us use daily, but did you ever stop to wonder how it really worked? I don't know about you, but it bugs me when I do not know how something really works. The purpose of this paper is to resolve any lingering questions you may have about ping and to take your understanding to the next level. If you do not happen to be a programmer, please do not be frightened off! I am not going to tell you how to write your own version of ping; trust me.

I am guessing that you know basically how the TCP/IP ping utility works. It sends an ICMP (Internet Control Message Protocol) Echo Request to a specified interface on the network and, in response, it expects to receive an ICMP Echo Reply. By doing this, the program can test connectivity, gauge response time, and report a variety of errors.

ICMP is a software component of the Internetworking layer of TCP/IP; essentially, it is a companion at that level to IP (Internet Protocol) itself. In fact, ICMP relies on IP for transport across the network. If you observe this sort of network traffic, say on an Ethernet network, then your protocol analyzer would capture an Ethernet frame transporting an IP datagram with an ICMP message inside.

Enter the problem: Since the ping program executes at the Application layer, how does it make ICMP do these tricks? You may recall, if you are a student of TCP/IP, that the Host-to-Host layer is sandwiched between these entities. Is that bypassed? If so, then how? Who is responsible for formatting these messages (Echo Request and Echo Reply)?

More vexingly, when unexpected ICMP responses, other than the customary Echo Reply, result from the Echo Request, how is it that they find their way to the ping program? This last question may seem obvious, but it is not. ICMP messages contain no addressing information that allows the TCP/IP protocol stack to discern the program that is to receive the message. TCP and UDP use port numbers for this purpose. So, how does this work?

Background

The TCP/IP protocol stack is organized as a four-layer model (see Figure 1). The lowest layer, commonly called the Network Interface or Network Access layer, is analogous to OSI layers 1 and 2, the Physical and Data Link Control layers. This includes things like media, connectors, signaling, physical addressing, error detection, and managing shared access to the media. For most of us this translates into Ethernet and our cabling system.

Figure 1

TCP/IP Model
Process or Application Layer
Host-to-Host Layer
Internetworking Layer
Network Interface or Network Access Layer
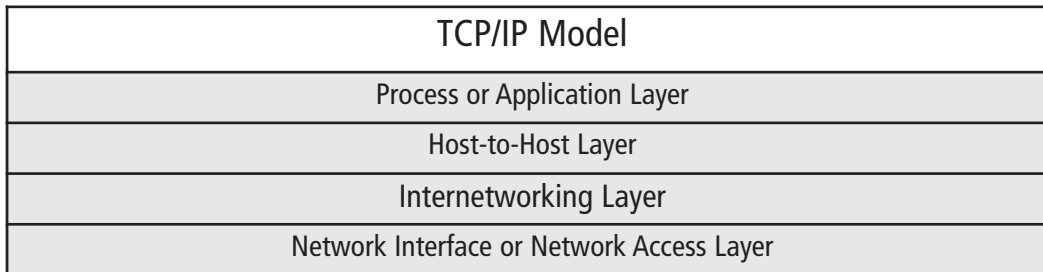
The layer above the Network Access layer, the Internetworking layer, is best likened to OSI layer 3, the Network layer. Here we expect to find logical addressing and routing: things that facilitate communication across network boundaries. This is where IP and its addressing mechanisms reside, as does ICMP.

ICMP is a necessary component of any TCP/IP implementation. It does not exist to provide information to the higher-layer protocols (like TCP and UDP) so that they may be more reliable. Rather, ICMP provides network diagnostic capabilities and feedback to those responsible for network administration and operation. See RFC 792, if you are really interested.

Above the Internetworking layer is the Host-to-Host layer, which is the counterpart of OSI layer 4, the Transport layer. I like to think that this also includes some of the Session layer (5) functionality as well. This is where we expect to find facilities for reliable end-to-end data exchange, additional error checking, and the means to discriminate one program from another (using port numbers). TCP and UDP reside at this level.

At the top of the stack, the Application or Process layer, we find high-level protocols (like SMTP, HTTP, and FTP) implemented. This is where applications execute as well. So when you do a ping, the ping program should be perceived to function at this level.

A Minor Mystery

With ICMP operating at the Internetworking layer and the ping program at the Application layer, how is the Host-to-Host layer bypassed? The answer lies in an understanding of what are known as "raw" sockets.

Well, for openers, what is a socket, right? Abstractly, a socket is an endpoint to communication, usually thought to consist of an IP address and port number, which identify a particular host and program, respectively. But a programmer has a slightly different perspective on a socket. From his vantage point, "socket" is a system function that allocates resources that enable the program to interact with the TCP/IP protocol stack beneath. The addressing information is associated with this only after the socket call is made. (Again, if you are interested, this is the role of the "bind" function.) So, take note, it is possible to allocate a socket and not overtly associate any addressing information with it.

There are three commonly encountered types of sockets: stream, datagram, and raw. TCP uses the stream type and UDP uses the datagram type. Raw sockets are used by any application that needs to interact directly with IP, bypassing TCP and UDP in doing so. Customers include routing protocol implementations like routed and gated (that implement RIP and OSPF). It also includes our friend ping.

There are some special considerations in using raw sockets. Since you are circumventing the facilities of the Host-to-Host layer, you forego the program addressing mechanism, the port numbering scheme. This means that programs that employ raw sockets must sift through all incoming packets presented to them in order to find those packets that are of interest.

What Actually Goes On

When the ping program begins execution, it opens a raw socket sensitive only to ICMP. This means two things:

- **On output:** the sending of ICMP Echo Requests, the program is required to format the ICMP message. The system will provide the IP header and the Ethernet (usually) header.
- **On input:** the program must examine all ICMP messages coming in and cull out the items of interest. The expected input is ICMP Echo Replies.

Let us take these things in turn.

On the outbound side, the Echo Requests are formatted in the manner shown in Figure 2. The message type is always the coded value eight (8). The code field always contains zero. The checksum is used for error detection. The ICMP message header and data are included in its computation. The ping program performs this calculation and fills in the blank. The identification field follows and is supposed to contain the process ID (PID) that uniquely identifies that execution of the ping program to the operating system. On Windows systems, this field contains the constant value 256. Next is the sequence number field, which starts at 0 and is bumped by one on each Echo Request sent. After these required fields, optional test data will follow. In the ping implementation that I examined (Slackware Linux), this included a timestamp used in the round-trip time calculation upon receipt of the Echo Reply.

Figure 2

ICMP Echo Request	
Type (8)	Code (0)
Checksum	
Identification	
Sequence	
Test Data	

As for inbound ICMP messages, ping's task is a bit more complex. Because ping is using a raw ICMP socket, the program is presented with a copy of all incoming ICMP messages, except for a few special cases like incoming Echo Requests generated by other people pinging us (the latter are handled by the system). This means that ping sees not only the expected Echo Replies when they arrive but also things like Destination Unreachable, Source Quench, and Time Exceeded messages. (Figure 3 summarizes the ICMP message types.)

Figure 3

ICMP Message Types		
Type	Codes	Description
0/8	0	Echo Reply/Echo Request
3	0-15	Destination Unreachable
4	0	Source Quench
5	0-3	Redirect
9/10	0	Router Advertisement
11	0-1	Time Exceeded
12	0	Parameter Problem
13/14	0	Timestamp Request/Timestamp Reply
17/18	0	Address Mask Request/Address Mask Reply

Now think about this for a moment. If you have two copies of the ping program running at the same time, then they are each going to see one another's Echo Replies and any other "nastygrams" that might show up. Each instance of the program must identify the messages that are relevant to it. If you guessed that this is what the PID (identification) field is used for then you are absolutely right.

How does the Windows flavor of ping accomplish this feat without the PID? You got me. That sounds like a topic for a future article. Let me get back to you on that.

Interestingly, the messages coming in are handed to ping with the IP header still intact. So, the program has access to important things there like the time-to-live (TTL) value and record route information (if the latter option is turned on).

Summary

At this point, you should have a fairly complete understanding of the cycle of processing associated with ping. Let me recapitulate the essential elements:

- As the ping program initializes, it opens a raw ICMP socket so that it can employ IP directly, circumventing TCP and UDP.
- Ping formats an ICMP type 8 message, an Echo Request, and sends it (using the "sendto" function) to the designated target address. The system provides the IP header and the data link layer envelope.
- ¶ As ICMP messages are received, ping has the opportunity to examine each packet to pick out those items that are of interest.
- The usual behavior is to siphon off ICMP type 0 messages, Echo Replies, which have an identification field value that matches the program PID.

- Ping uses the timestamp in the data area of the Echo Reply to calculate a round-trip time. It also reports the TTL from the IP header of the reply.
- When things do not work normally, ping may report some of the other ICMP message types that show up in the inbox. This includes things like Destination Unreachable and Time Exceeded messages.

Learn More

Learn more about how you can improve productivity, enhance efficiency, and sharpen your competitive edge. Check out the following Global Knowledge courses:

Network+ Boot Camp
TCP/IP Networking

For more information or to register, visit www.globalknowledge.com or call 1-800-COURSES to speak with a sales representative.

Our courses and enhanced, hands-on labs offer practical skills and tips that you can immediately put to use. Our expert instructors draw upon their experiences to help you understand key concepts and how to apply them to your specific work situation. Choose from our more than 700 courses, delivered through Classrooms, e-Learning, and On-site sessions, to meet your IT and management training needs.

About the Author

George Mays has over 35+ years experience in computing, data communications, and networking. His experience includes: mainframe systems programmer, Fortune 500 DBA, management of systems programming, data communications, IT operations, engineering, software development, and networking. He is also the author and course director for Global Knowledge's Network+ Boot Camp and has contributed to several hacking and security books. George holds various industry certifications including: CISSP, CCNA, A+, Network+, Security+, I-Net+. Past certification: MCSE. He is an instructor in TCP/IP, Troubleshooting, Network Protocols, Network Fundamentals, A+, Security+ and CISSP. In addition to teaching for Global Knowledge, Mr. Mays also acts as a consultant in the fields of general networking and security.