Network Working Group   Bill Croft (Stanford University)
Request for Comments: 951   John Gilmore (Sun Microsystems)
September 1985

BOOTSTRAP PROTOCOL (BOOTP)

1. Status of this Memo

This RFC suggests a proposed protocol for the ARPA-Internet community, and requests discussion and suggestions for improvements. Distribution of this memo is unlimited.

2. Overview

This RFC describes an IP/UDP bootstrap protocol (BOOTP) which allows a diskless client machine to discover its own IP address, the address of a server host, and the name of a file to be loaded into memory and executed. The bootstrap operation can be thought of as consisting of TWO PHASES. This RFC describes the first phase, which could be labeled 'address determination and bootfile selection'. After this address and filename information is obtained, control passes to the second phase of the bootstrap where a file transfer occurs. The file transfer will typically use the TFTP protocol [9], since it is intended that both phases reside in PROM on the client. However BOOTP could also work with other protocols such as SFTP [3] or FTP [6].

We suggest that the client's PROM software provide a way to do a complete bootstrap without 'user' interaction. This is the type of boot that would occur during an unattended power-up. A mechanism should be provided for the user to manually supply the necessary address and filename information to bypass the BOOTP protocol and enter the file transfer phase directly. If non-volatile storage is available, we suggest keeping default settings there and bypassing the BOOTP protocol unless these settings cause the file transfer phase to fail. If the cached information fails, the bootstrap should fall back to phase 1 and use BOOTP.

Here is a brief outline of the protocol:

1. A single packet exchange is performed. Timeouts are used to retransmit until a reply is received. The same packet field layout is used in both directions. Fixed length fields of maximum reasonable length are used to simplify structure definition and parsing.

2. An 'opcode' field exists with two values. The client broadcasts a 'bootrequest' packet. The server then answers with a 'bootreply' packet. The bootrequest contains the client's hardware address and its IP address, if known.

3. The request can optionally contain the name of the server the client wishes to respond. This is so the client can force the boot to occur from a specific host (e.g. if multiple versions of the same bootfile exist or if the server is in a far distant net/domain). The client does not have to deal with name / domain services; instead this function is pushed off to the BOOTP server.

4. The request can optionally contain the 'generic' filename to be booted. For example 'unix' or 'ethertip'. When the server sends the bootreply, it replaces this field with the fully qualified path name of the appropriate boot file. In determining this name, the server may consult his own database correlating the client's address and filename request, with a particular boot file customized for that client. If the bootrequest filename is a null string, then the server returns a filename field indicating the 'default' file to be loaded for that client.

5. In the case of clients who do not know their IP addresses, the server must also have a database relating hardware address to IP address. This client IP address is then placed into a field in the bootreply.

6. Certain network topologies (such as Stanford's) may be such that a given physical cable does not have a TFTP server directly attached to it (e.g. all the gateways and hosts on a certain cable may be diskless). With the cooperation of neighboring gateways, BOOTP can allow clients to boot off of servers several hops away, through these gateways. See the section 'Booting Through Gateways' below. This part of the protocol requires no special action on the part of the client. Implementation is optional and requires a small amount of additional code in gateways and servers.

3. Packet Format

All numbers shown are decimal, unless indicated otherwise. The BOOTP packet is enclosed in a standard IP [8] UDP [7] datagram. For simplicity it is assumed that the BOOTP packet is never fragmented. Any numeric fields shown are packed in 'standard network byte order', i.e. high order bits are sent first.

In the IP header of a bootrequest, the client fills in its own IP source address if known, otherwise zero. When the server address is unknown, the IP destination address will be the 'broadcast address' 255.255.255.255. This address means 'broadcast on the local cable, (I don't know my net number)' [4].

The UDP header contains source and destination port numbers. The BOOTP protocol uses two reserved port numbers, ’BOOTP client’ (68) and ’BOOTP server’ (67). The client sends requests using ’BOOTP server’ as the destination port; this is usually a broadcast. The server sends replies using ’BOOTP client’ as the destination port; depending on the kernel or driver facilities in the server, this may or may not be a broadcast (this is explained further in the section titled ’Chicken/Egg issues’ below). The reason TWO reserved ports are used, is to avoid ’waking up’ and scheduling the BOOTP server daemons, when a bootreply must be broadcast to a client. Since the server and other hosts won’t be listening on the ’BOOTP client’ port, any such incoming broadcasts will be filtered out at the kernel level. We could not simply allow the client to pick a ’random’ port number for the UDP source port field; since the server reply may be broadcast, a randomly chosen port number could confuse other hosts that happened to be listening on that port.

The UDP length field is set to the length of the UDP plus BOOTP portions of the packet. The UDP checksum field can be set to zero by the client (or server) if desired, to avoid this extra overhead in a PROM implementation. In the ’Packet Processing’ section below the phrase ’[UDP checksum.]’ is used whenever the checksum might be verified/computed.

FIELD	BYTES	DESCRIPTION
op	1	packet op code / message type. 1 = BOOTREQUEST, 2 = BOOTREPLY
htype	1	hardware address type, see ARP section in "Assigned Numbers" RFC. ’1’ = 10mb ethernet
hlen	1	hardware address length (eg ’6’ for 10mb ethernet).
hops	1	client sets to zero, optionally used by gateways in cross-gateway booting.
xid	4	transaction ID, a random number, used to match this boot request with the responses it generates.
secs	2	filled in by client, seconds elapsed since client started trying to boot.

--	2	unused
ciaddr	4	client IP address; filled in by client in bootrequest if known.
yiaddr	4	'your' (client) IP address; filled by server if client doesn't know its own address (ciaddr was 0).
siaddr	4	server IP address; returned in bootreply by server.
giaddr	4	gateway IP address, used in optional cross-gateway booting.
chaddr	16	client hardware address, filled in by client.
sname	64	optional server host name, null terminated string.
file	128	boot file name, null terminated string; 'generic' name or null in bootrequest, fully qualified directory-path name in bootreply.
vend	64	optional vendor-specific area, e.g. could be hardware type/serial on request, or 'capability' / remote file system handle on reply. This info may be set aside for use by a third phase bootstrap or kernel.

4. Chicken / Egg Issues

How can the server send an IP datagram to the client, if the client doesnt know its own IP address (yet)? Whenever a bootreply is being sent, the transmitting machine performs the following operations:

1. If the client knows its own IP address ('ciaddr' field is nonzero), then the IP can be sent 'as normal', since the client will respond to ARPs [5].

2. If the client does not yet know its IP address (ciaddr zero), then the client cannot respond to ARPs sent by the transmitter of the bootreply. There are two options:

   a. If the transmitter has the necessary kernel or driver hooks to 'manually' construct an ARP address cache entry, then it can fill in an entry using the 'chaddr' and 'yiaddr' fields. Of course, this entry should have a timeout on it, just like any other entry made by the normal ARP code itself. The transmitter of the bootreply can then simply send the bootreply to the client's IP address. UNIX (4.2 BSD) has this capability.

b. If the transmitter lacks these kernel hooks, it can simply send the bootreply to the IP broadcast address on the appropriate interface. This is only one additional broadcast over the previous case.

5. Client Use of ARP

The client PROM must contain a simple implementation of ARP, e.g. the address cache could be just one entry in size. This will allow a second-phase-only boot (TFTP) to be performed when the client knows the IP addresses and bootfile name.

Any time the client is expecting to receive a TFTP or BOOTP reply, it should be prepared to answer an ARP request for its own IP to hardware address mapping (if known).

Since the bootreply will contain (in the hardware encapsulation) the hardware source address of the server/gateway, the client MAY be able to avoid sending an ARP request for the server/gateway IP address to be used in the following TFTP phase. However this should be treated only as a special case, since it is desirable to still allow a second-phase-only boot as described above.

6. Comparison to RARP

An earlier protocol, Reverse Address Resolution Protocol (RARP) [1] was proposed to allow a client to determine its IP address, given that it knew its hardware address. However RARP had the disadvantage that it was a hardware link level protocol (not IP/UDP based). This means that RARP could only be implemented on hosts containing special kernel or driver modifications to access these 'raw' packets. Since there are many network kernels existent now, with each source maintained by different organizations, a boot protocol that does not require kernel modifications is a decided advantage.

BOOTP provides this hardware to IP address lookup function, in addition to the other useful features described in the sections above.

7. Packet Processing

7.1. Client Transmission

Before setting up the packet for the first time, it is a good idea to clear the entire packet buffer to all zeros; this will place all fields in their default state. The client then creates a packet with the following fields.

The IP destination address is set to 255.255.255.255. (the broadcast address) or to the server's IP address (if known). The IP source address and 'ciaddr' are set to the client's IP address if known, else 0. The UDP header is set with the proper length; source port = 'BOOTP client' port destination port = 'BOOTP server' port.

'op' is set to '1', BOOTREQUEST. 'htype' is set to the hardware address type as assigned in the ARP section of the "Assigned Numbers" RFC. 'hlen' is set to the length of the hardware address, e.g. '6' for 10mb ethernet.

'xid' is set to a 'random' transaction id. 'secs' is set to the number of seconds that have elapsed since the client has started booting. This will let the servers know how long a client has been trying. As the number gets larger, certain servers may feel more 'sympathetic' towards a client they don't normally service. If a client lacks a suitable clock, it could construct a rough estimate using a loop timer. Or it could choose to simply send this field as always a fixed value, say 100 seconds.

If the client knows its IP address, 'ciaddr' (and the IP source address) are set to this value. 'chaddr' is filled in with the client's hardware address.

If the client wishes to restrict booting to a particular server name, it may place a null-terminated string in 'sname'. The name used should be any of the allowable names or nicknames of the desired host.

The client has several options for filling the 'file' name field. If left null, the meaning is 'I want to boot the default file for my machine'. A null file name can also mean 'I am only interested in finding out client/server/gateway IP addresses, I dont care about file names'.

The field can also be a 'generic' name such as 'unix' or 'gateway'; this means 'boot the named program configured for my machine'. Finally the field can be a fully directory qualified path name.

The 'vend' field can be filled in by the client with vendor-specific strings or structures. For example the machine hardware type or serial number may be placed here. However the operation of the BOOTP server should not DEPEND on this information existing.

If the 'vend' field is used, it is recommended that a 4 byte 'magic number' be the first item within 'vend'. This lets a server determine what kind of information it is seeing in this field. Numbers can be assigned by the usual 'magic number' process --you pick one and it's magic. A different magic number could be used for bootreply's than bootrequest's to allow the client to take special action with the reply information.

[UDP checksum.]

7.2. Client Retransmission Strategy

If no reply is received for a certain length of time, the client should retransmit the request. The time interval must be chosen carefully so as not to flood the network. Consider the case of a cable containing 100 machines that are just coming up after a power failure. Simply retransmitting the request every four seconds will inundate the net.

As a possible strategy, you might consider backing off exponentially, similar to the way ethernet backs off on a collision. So for example if the first packet is at time 0:00, the second would be at :04, then :08, then :16, then :32, then :64. You should also randomize each time; this would be done similar to the ethernet specification by starting with a mask and 'and'ing that with with a random number to get the first backoff. On each succeeding backoff, the mask is increased in length by one bit. This doubles the average delay on each backoff.

After the 'average' backoff reaches about 60 seconds, it should be increased no further, but still randomized.

Before each retransmission, the client should update the 'secs' field. [UDP checksum.]

7.3. Server Receives BOOTREQUEST

[UDP checksum.] If the UDP destination port does not match the ’BOOTP server’ port, discard the packet.

If the server name field (sname) is null (no particular server specified), or sname is specified and matches our name or nickname, then continue with packet processing.

If the sname field is specified, but does not match ’us’, then there are several options:

1. You may choose to simply discard this packet.

2. If a name lookup on sname shows it to be on this same cable, discard the packet.

3. If sname is on a different net, you may choose to forward the packet to that address. If so, check the ’giaddr’ (gateway address) field. If ’giaddr’ is zero, fill it in with my address or the address of a gateway that can be used to get to that net. Then forward the packet.

If the client IP address (ciaddr) is zero, then the client does not know its own IP address. Attempt to lookup the client hardware address (chaddr, hlen, htype) in our database. If no match is found, discard the packet. Otherwise we now have an IP address for this client; fill it into the ’yiaddr’ (your IP address) field.

We now check the boot file name field (file). The field will be null if the client is not interested in filenames, or wants the default bootfile. If the field is non-null, it is used as a lookup key in a database, along with the client’s IP address. If there is a default file or generic file (possibly indexed by the client address) or a fully-specified path name that matches, then replace the ’file’ field with the fully-specified path name of the selected boot file. If the field is non-null and no match was found, then the client is asking for a file we dont have; discard the packet, perhaps some other BOOTP server will have it.

The ’vend’ vendor-specific data field should now be checked and if a recognized type of data is provided, client-specific actions should be taken, and a response placed in the ’vend’ data field of the reply packet. For example, a workstation client could provide an authentication key and receive from the server a capability for remote file access, or a set of configuration options, which can be passed to the operating system that will shortly be booted in.

Place my (server) IP address in the ’siaddr’ field. Set the ’op’ field to BOOTREPLY. The UDP destination port is set to ’BOOTP client’. If the client address ’ciaddr’ is nonzero, send the packet there; else if the gateway address ’giaddr’ is nonzero, set the UDP destination port to ’BOOTP server’ and send the packet to ’giaddr’; else the client is on one of our cables but it doesnt know its own IP address yet --use a method described in the ’Egg’ section above to send it to the client. If ’Egg’ is used and we have multiple interfaces on this host, use the ’yiaddr’ (your IP address) field to figure out which net (cable/interface) to send the packet to. [UDP checksum.]

7.4. Server/Gateway Receives BOOTREPLY

[UDP checksum.] If ’yiaddr’ (your [the client’s] IP address) refers to one of our cables, use one of the ’Egg’ methods above to forward it to the client. Be sure to send it to the ’BOOTP client’ UDP destination port.

7.5. Client Reception

Don’t forget to process ARP requests for my own IP address (if I know it). [UDP checksum.] The client should discard incoming packets that: are not IP/UDPs addressed to the boot port; are not BOOTREPLYs; do not match my IP address (if I know it) or my hardware address; do not match my transaction id. Otherwise we have received a successful reply. ’yiaddr’ will contain my IP address, if I didnt know it before. ’file’ is the name of the file name to TFTP ’read request’. The server address is in ’siaddr’. If ’giaddr’ (gateway address) is nonzero, then the packets should be forwarded there first, in order to get to the server.

8. Booting Through Gateways

This part of the protocol is optional and requires some additional code in cooperating gateways and servers, but it allows cross-gateway booting. This is mainly useful when gateways are diskless machines. Gateways containing disks (e.g. a UNIX machine acting as a gateway), might as well run their own BOOTP/TFTP servers.

Gateways listening to broadcast BOOTREQUESTs may decide to forward or rebroadcast these requests ’when appropriate’. For example, the gateway could have, as part of his configuration tables, a list of other networks or hosts to receive a copy of any broadcast BOOTREQUESTs. Even though a 'hops' field exists, it is a poor idea to simply globally rebroadcast the requests, since broadcast loops will almost certainly occur.

The forwarding could begin immediately, or wait until the 'secs' (seconds client has been trying) field passes a certain threshold.

If a gateway does decide to forward the request, it should look at the 'giaddr' (gateway IP address) field. If zero, it should plug its own IP address (on the receiving cable) into this field. It may also use the 'hops' field to optionally control how far the packet is reforwarded. Hops should be incremented on each forwarding. For example, if hops passes '3', the packet should probably be discarded. [UDP checksum.]

Here we have recommended placing this special forwarding function in the gateways. But that does not have to be the case. As long as some 'BOOTP forwarding agent' exists on the net with the booting client, the agent can do the forwarding when appropriate. Thus this service may or may not be co-located with the gateway.

In the case of a forwarding agent not located in the gateway, the agent could save himself some work by plugging the broadcast address of the interface receiving the bootrequest into the 'giaddr' field. Thus the reply would get forwarded using normal gateways, not involving the forwarding agent. Of course the disadvantage here is that you lose the ability to use the 'Egg' non-broadcast method of sending the reply, causing extra overhead for every host on the client cable.

9. Sample BOOTP Server Database

As a suggestion, we show a sample text file database that the BOOTP server program might use. The database has two sections, delimited by a line containing an percent in column 1. The first section contains a 'default directory' and mappings from generic names to directory/pathnames. The first generic name in this section is the 'default file' you get when the bootrequest contains a null 'file' string.

The second section maps hardware addresstype/address into an ipaddress. Optionally you can also overide the default generic name by supplying a ipaddress specific genericname. A 'suffix' item is also an option; if supplied, any generic names specified by the client will be accessed by first appending 'suffix' to the 'pathname' appropriate to that generic name. If that file is not found, then the plain 'pathname' will be tried. This 'suffix' option allows a whole set of custom generics to be setup without a lot of effort. Below is shown the general format; fields are delimited by one or more spaces or tabs; trailing empty fields may be omitted; blank lines and lines beginning with '#' are ignored.

```
# comment line

homedirectory
genericname1    pathname1
genericname2    pathname2
...

% end of generic names, start of address mappings

hostname1 hardwaretype hardwareaddr1 ipaddr1 genericname suffix
hostname2 hardwaretype hardwareaddr2 ipaddr2 genericname suffix
...
```

Here is a specific example. Note the 'hardwaretype' number is the same as that shown in the ARP section of the 'Assigned Numbers' RFC. The 'hardwaretype' and 'ipaddr' numbers are in decimal; 'hardwareaddr' is in hex.

```
# last updated by smith

/usr/boot
vmunix          vmunix
tip             ethertip
watch           /usr/diag/etherwatch
gate            gate.

% end of generic names, start of address mappings

hamilton        1 02.60.8c.06.34.98     36.19.0.5
burr            1 02.60.8c.34.11.78     36.44.0.12
101-gateway     1 02.60.8c.23.ab.35     36.44.0.32      gate 101
mjh-gateway     1 02.60.8c.12.32.bc     36.42.0.64      gate mjh
welch-tipa      1 02.60.8c.22.65.32     36.47.0.14      tip
welch-tipb      1 02.60.8c.12.15.c8     36.46.0.12      tip
```

In the example above, if 'mjh-gateway' does a default boot, it will get the file '/usr/boot/gate.mjh'.

10. Acknowledgements

Ross Finlayson (et. al.) produced two earlier RFC’s discussing TFTP bootstraping [2] using RARP [1].

We would also like to acknowledge the previous work and comments of Noel Chiappa, Bob Lyon, Jeff Mogul, Mark Lewis, and David Plummer.

REFERENCES

1. Ross Finlayson, Timothy Mann, Jeffrey Mogul, Marvin Theimer. A Reverse Address Resolution Protocol. RFC 903, NIC, June, 1984.

2. Ross Finlayson. Bootstrap Loading using TFTP. RFC 906, NIC, June, 1984.

3. Mark Lottor. Simple File Transfer Protocol. RFC 913, NIC, September, 1984.

4. Jeffrey Mogul. Broadcasting Internet Packets. RFC 919, NIC, October, 1984.

5. David Plummer. An Ethernet Address Resolution Protocol. RFC 826, NIC, September, 1982.

6. Jon Postel. File Transfer Protocol. RFC 765, NIC, June, 1980.

7. Jon Postel. User Datagram Protocol. RFC 768, NIC, August, 1980.

8. Jon Postel. Internet Protocol. RFC 791, NIC, September, 1981.

9. K. R. Sollins, Noel Chiappa. The TFTP Protocol. RFC 783, NIC, June, 1981.