Inter-Office Memorandum

To	Communication Protocols	Date	February 13, 1979
From	Ed Taft and David Boggs	Location	Palo Alto
Subject	Alto Boot Protocol	Organization	Parc/CSL

XEROX

Filed on: [Maxc1]<Pup>AltoBoot.press

This memo describes the protocol by which Altos are boot-loaded over the Ethernet, the protocol for discovering what boot files are available, the protocol for distributing the most recent version of a boot file to all boot servers, and the protocol for getting a boot server's statistics.

Because gateways are up 24 hours a day and are often located at places in the internet where many Ethernets come together, most gateways contain a boot server. However it is important to understand that boot servers and gateways are logically two very different things which are only physically co-located for convenience. There are gateways which aren't boot servers (e.g. Maxc1), and boot servers which aren't gateways (e.g. Peek).

Breath of Life

A Boot server periodically (every 5 seconds or so) sends a BreathOfLife packet on each directly-connected Ethernet. This is not a Pup: it is a raw Ethernet packet with the Ethernet destination address set to a special value. The remainder of the packet is an Alto Ethernet boot loader program.

When an Alto is booted with the BS key depressed, the boot microcode enables the Ethernet receiver to accept packets directed to host 377B and copies them into memory beginning at location 1. When a packet of type 602B is received without error, the Alto then begins executing instructions at location 3.

The current Alto Ethernet boot loader is contained in <AltoSource>EtherBoot.asm.

Mayday

An Alto which wants to be boot-loaded broadcasts a Pup of type BootFileRequest to Miscellaneous Services sockets of all hosts on the directly-connected Ethernet. The low-order 16 bits of the Pup ID is the number of the boot file desired. The Pup source port is the one to which the Alto wants the boot file sent.

Since the BootFileRequest Pup may be lost, it should be periodically retransmitted up to some maximum number of retries if no response is received at first. EtherBoot retransmits about once a second for about 30 seconds. The reason for giving up after a while is that perhaps it is getting no answer because its Ethernet interface is broken and is polluting the net whenever it transmits.

The standard boot loader, when started at its normal address (3), reads one of the Alto keyboard words to determine the desired boot file number. All keys up (except BS) corresponds to boot file number 0. One-bits in this word are selected by depressing combinations of the following keys, listed most-significant bit first:

3 2 W Q S A 9 I X O L , "] BLANK-MIDDLE BLANK-TOP

EFTP

The actual transfer of a boot file is accomplished using the Pup EFTP protocol. A boot server receiving a request for a boot file it is willing to supply simply attempts to EFTP that file to the port from which the BootFileRequest Pup arose.

Since several boot servers may respond to a single request, a server should be prepared for the EFTP transmit attempt to fail. When the Ethernet boot loader receives the first EFTPData Pup in response to its BootFileRequest, it locks on to that source and ignores Pups from everywhere else. Due to space limitations (254 words), it is unable to respond to other EFTP transmissions with EFTPAbort Pups, as specified by protocol.

There are two timeouts of interest here: the *abort timeout*, within which an ack must be received or the transfer is aborted, and the *retransmission timeout*, after which if an ack has not arrived the current data block is sent again. Ideally the retransmission timeout should be adaptive: about 2 times the average response time, exponentially aged over the last 8 samples. In any case it should be such as to retransmit a few times before the abort timeout takes. The abort timeout should be a function of the available bandwidth of the path between the sender and receiver. The table below lists recommended values.

	First Block		Subsequent Blocks	
	Abort	Retrans	Abort	Retrans
Fast net	500	100	5000	1000
Slow net	10000	2500	10000	2500

The timeouts for a slow net are suitable down to about 2400 bits/sec. The retransmission timeouts listed are for EFTP implementations which do not use an adaptive algorithm; the initial adaptive retransmission timeout may have to be reduced from its default value (typically 1 second) for the first block on a fast net. A reasonable simplification is to assume that all nets except Ethernets are slow. Even on an unloaded 9600 bits/sec line it takes several minutes to send a full core image boot file. Boot servers should be able to boot an Alto over an Ethernet while simultaneously updating a boot server at the end of a slow phone line.

Boot File Names and Numbers

String names and 16 bit numbers are both used to refer to boot files. Servers deal mostly in boot file numbers: requests to send a boot file refer to it by number; servers compare the creation dates of files with identical numbers when distributing new versions. The NetExec sorts its directory by name, keeping the number, date, and server host address as auxiliary information.

Boot file numbers less than 100000B have a uniform meaning throughout the network, are updated automatically and are assigned by administrative fiat. The remaining numbers are available for local use and do not propagate. [Ivy]<Portola>BootDirectory.txt is the master directory of registered boot files.

Most gateways have a boot file with the name <gatewayName>.boot, with number 100000B. This is intended for use by people developing new boot files. A test version of a boot file stored there won't propagate, and there can't be any doubt about which boot server it came from.

Boot Directory Information

When the EtherBoot mechanism was first developed it was only expected to handle a small number of files -- DMT, Scavenger, FTP and a few others -- and key combinations were picked that were easy to remember and convenient for people with two hands and ten fingers. Even so, it was difficult to remember the keys and the number of files grew to the point where this scheme was getting out of hand, so the NetExec was developed. The NetExec was assigned one of the last convenient key combinations and it is now the standard way for humans to invoke other boot files.

The NetExec discovers what boot files are available by broadcasting a Pup of type BootDirRequest to Miscellaneous Services sockets on all hosts on the directly-connected Ethernet. Hosts that are boot servers respond with packets of type BootDirReply containing <boot file number, date, boot file name> tuples. A boot server with lots of boot files may fill several BootDirReply Pups. The NetExec builds a directory of <boot file name, boot file number, date, host> tuples from these responses.

Boot File Update

At present there are two programs which implement boot servers: Gateways and Peek. There are about 20 gateways in operation, and the number is growing. It takes a few minutes per gateway to update one boot file (most gateways are at the end of slow phone lines). Not all gateways are up all of the time. There are probably 50 Peek disks in the world, each with some subset of the boot files that existed when the disk was built. The owners of Peek disks are exhorted to rebuild their disks about once a month. The result of this anarchy is that old versions of boot files persisted in the internet for years.

A Boot file now includes the date on which it was built. Boot servers periodically exchange boot file directories which include these dates. When a new version of a boot file is stored onto any boot server, all other boot servers will soon discover this and automatically update their local copies. The protocol is similar to that used by name servers to update the network directory.

About once an hour and each time a new boot file arrives, each boot server broadcasts its boot file directory in a BootDirReply Pup to miscellaneous services sockets on all directly connected networks. When a boot server receives one of these, it compares the dates of its local boot files with the dates in the Pup. If the sender has a more recent version, then the receiver requests a copy using a SendBootFile Pup. If the receiver has a more recent version, then it should send a BootDirReply to cause the sender to update. The same EFTP protocol that is used to boot an Alto is used to move new versions among boot servers.

If the date comparison is unguarded, a damaged file with a bogus date far in the future could propagate everywhere and it would be impossible to purge. To protect against this, a file which claims to have been created in the future should be treated as if it had a date of zero, thus making it elegible for update by anyone.

Server Statistics

Boot servers may optionally keep statistics on their activities and make them available through the net. A program requests a boot server's statistics by sending a Pup of type BootStatsRequest to the miscellaneous services socket, and the boot server responds by sending a Pup of type BootStatsReply containing the statistics. The first word of the reply is a format version number which is incremented whenever the format changes.

Pointers to other Documentation

When an Alto is hardware-booted over the Ethernet, all three of the steps (BreathOfLife, MayDay, EFTP) are executed. A software-initiated boot may be accomplished by copying the boot loader into page 0 and jumping into it, thereby starting at the "Mayday" stage with the boot file number and host as optional arguments. Further information may be found in the "EtherBoot" package documentation.

The standard Ethernet boot loader can load only B-format boot files. A boot server must transform S0-format files into B-format files (by rearranging pages) before sending them. Further information may be found in the "BuildBoot" subsystem documentation.

Details

A Breath of life packet is a raw *(non-Pup)* Ethernet packet:

Destination:	377B
Type:	602B
Contents:	A boot loader program.

The starting address of the boot loader is the the third word of the packet (first content word) which will be address 3 in Alto memory. The total packet length must not exceed 256 words.

Boot servers listen on the Miscellaneous Services socket (4) and handle some or all of the Pup types listed below.

BootFileRequest

Pup Type:	244B
Pup ID:	Low 16 bits are the boot file number desired
Pup SPort:	The port to which the boot file should be EFTPed
Pup DPort:	Miscellaneous services
Pup Contents:	None

This packet is generated in two contexts: 1) by the Ether boot loader while booting an Alto, and 2) by a boot server to update a local copy of a boot file.

BootDirRequest

Pup Type:	257B
Pup DPort:	Miscellaneous services
Pup Contents:	None

This packet is generated by the NetExec to discover who the boot servers are and what files they have.

BootDirReply

Pup Type:	260B
Pup ID:	if it is in reply to a BootDirRequest, the ID should match the request.
Pup Contents:	1 or more blocks of the following format: A boot file number (the number that goes in the low 16 bits of a BootFileRequest Pup), an Alto format date (2 words), a boot file name in BCPL string format.

This packet is generated 1) in response to a BootDirRequest, 2) gratuitously broadcast every hour, and 3) in response to a BootDirReply advertising an older version of a local file.

KissOfDeath

Pup Type:	247B
Pup DPort:	Miscellaneous services
Pup SPort:	The BootFileRequest Pup is sent to SPort.host on the local Ethernet. If SPort.host is zero, it is broadcast.

Pup ID:	The low 16 bits contain the boot file number to put in the BootFileRequest Pup.
Pup Contents:	None

DMT is the only program which currently responds to KissOfDeath Pups and is used now only to run tests on the Ethernet. We have a multiprocessor with about 125 6-MIP CPUs on the second floor of Parc which is idle 16 hours a day just waiting for someone to figure out how to use it.

BootStatsRequest

Pup Type:	253B
Pup DPort:	Miscellaneous services
Pup Contents:	None

BootStatsReply

Pup Type:	254B
PupID:	same as the BootStatsRequest that triggered it.
Pup Contents:	A version number to identify the format of the following words (current version = 1). Followed by the number of boot files sent, followed by the number of boot directories sent, both in BCPL double precision format.

Revision History

October 17, 1976

First release

March 9, 1978

Boot directory protocol added.

December 31, 1978

Automatic update protocol added.

February 13, 1979

Boot server statistics protocol added.