The ATA over Ethernet Protocol

Brantley Coile
Sam Hopkins

Coraid, Inc.
565 Research Drive
Athens, GA 30605
{brantley|sah}@coraid.com

1. Introduction

AoE is an open standards based protocol that allows direct network access to disk drives by client hosts such as web servers, mail servers, or cluster servers. These systems can use the AoE protocol to access an arbitrarily large array of AoE servers through the use of commodity Ethernet switches. The AoE servers could be small processors and disk drives on a small printed circuit board plugged into a small rack mounted chassis. These small boards are called 'blades.'

The protocol is described in the AoE specification. This document is meant to provide an introduction to the protocol, and aid in understanding the protocol features. In this document, host means the client host that is *using* the disk drives, and the server is the network node that is *providing* block access to the disk.

The protocol consists of request messages sent to the AoE server and reply messages returned to the host. Some messages contain ATA commands, and any data associated with the transaction. Other messages relate to the Config/Query feature of the protocol, to set and query a small amount of out-of-band data. The format of these message are simple and have two forms: ATA messages, and Config/Query messages. Both share a common header format that facilitates network delivery.

The structure of this paper reflects that structure of the protocol. The next section talks about the common header, and the following section discusses the ATA messages, with the section after than talking about the Config/Query messages. The next to last section talks about one way the AoE protocol can be used. It describes some aspects of our Linux device drivers. We conclude with some observations about the protocol.

2. The Common Header

While the two classes of messages, ATA and Config/Query, each have their own fields, they both share a common format for the first 24 bytes of a message. This common header provides enough information to send messages between client hosts and AoE servers. The common header has four functions. First, it provides a way to correlate responses with requests. Second, it provides a way to discover the Ethernet address of an AoE server at some physical location in a rack storage blades. Third, the common header identifies requests from the responses. Lastly, the header contains error information.

The header is shown in Figure 1. As can be seen the 14 byte Ethernet header is described for convenience. It is the *E* of AoE.

AoE messages can be queued in AoE servers. This allows disks to remain busy, because as soon as they finish one disk request they can start on the next. AoE uses Ethernet's best effort delivery, and the client host software is responsible for resending request messages that have not been responded to in a reasonable amount of time. To match responses with requests and to check for responses that were never received, the client host can use the *Tag* field. AoE servers copy the *Tag* field from requests to the

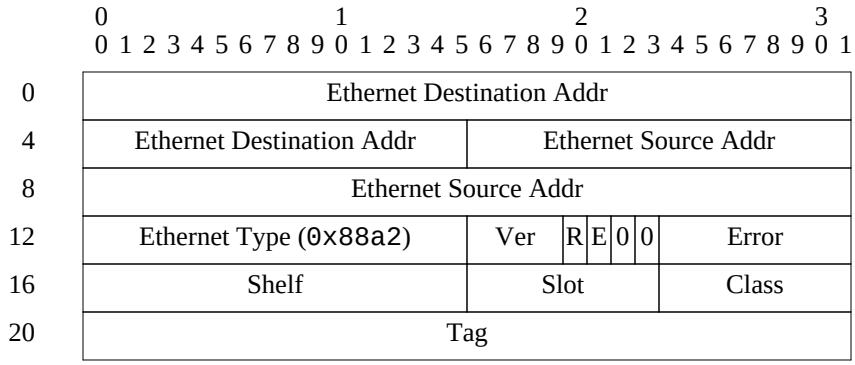

Figure 1. The Common AoE Header

responses unchanged, so the requester can put correlation information in the tag field. When the response is received, an internal table of tags can be scanned to find the associated request. Timeout routines can scan this same table looking for requests that have not been responded to. In this way, AoE provides all the information needed for reliable operation.

The next feature provides a way to find specific AoE servers. AoE server blades are little larger than 3.5" disk drives. They are inserted into slots in a small shelf. A number of these shelves can be bolted into simple relay racks, and many shelves connected to a local network segment. The common header has fields to help keep track of blade locations. Each shelf has dip switches giving it a unique identity. Each blade can read these dip switches as well as its slot number. These two numbers are represented in the common header as the shelf and slot address. (In the AoE specification they are referred to as *major* and *minor* numbers, not to be confused with values of the same name in Unix and Linux special devices.) These fields can be used to find the Ethernet address of a specific blade.

This is done with Ethernet broadcast messages. Ethernet has a reserved address, consisting of all ones, for broadcasting messages to all devices on the local network segment. AoE protocol requires that only request messages that have its unique shelf and slot number be processed, dropping all others. If we wish to find the Ethernet address of the blade in, say, shelf 6 slot 4, we would broadcast a message with the shelf and slot fields set to 6 and 4 respectively. Only that blade would respond. The response would have the Ethernet address of that blade.

The shelf and slot numbers also have their own broadcast values. Addresses of all ones will cause the AoE server to accept the request. This allow some message to be processed by more than one AoE server. If a shelf is all ones and the slot number a specific value, only the blades in that slot in every shelf would process the message. A slot value of all ones with a specific shelf address will cause only the blades in that shelf to respond. This could also be used to query for all the blades currently on the network segment.

Two flags are also used in the header. In order to keep from accidentally processing a response AoE servers will only process messages that contain a zero in the *R* bit. Client servers set this bit to zero in requests, and AoE servers set this bit to one in responses. This was added to keep blades from executing responses that may have been broadcast on the local network segment.

The other flag bit is used to indicate an error. The *E* bit in the header is set to one in a response when the request cannot be completed for some reason. The *Error* field will in that case contain an error message. See the AoE protocol specification for the current list of error codes.

Lastly, the *Class*, or *Cmd* as it's called in the AoE specification, contains a zero for ATA messages and a one for Config/Query messages.

3. ATA Class Messages

The ATA class messages are described in this section. The Advanced Technology Attachment is a standard that has evolved out of the ST506 interface and the Western Digital 1010 disk controller chip of the early 1980's. These chips were embedded in host bus adaptors (HBA) used in IBM Personal Computer compatible systems.

The chip had a number of registers that controlled disk data transfers. The set of registers held the cylinder, track and sector information, a sector count register that specified the number of sectors to be read or written, and a command register. When the command register was written with an operation code, the disk would initiate a data transfer. A status register and an error register would report any problems or the successful completion of the command.

As an example, in the case of a read command, data would be transferred from the disk into a buffer on the HBA, and the HBA would notify the host that data was available to be read. The host would then move the data into its local memory.

With the advent of In Disk Electronics (IDE), the controller function was moved from the HBA to the drive itself. The specification for the speed of transferring the data between the internal buffer and the host's memory was increased several times. This was all finally codified into the Advanced Technology Attachment or ATA standard. The ATA standard covers both the physical connections to the drive and the logical interface.

The original parameter registers using the three dimensional coordinates in the form of cylinder, track and sector, were replaced with a 24-bit logical block address (LBA). Since disks are addressed as 512 byte sectors, 24 bit LBAs limited disks to 137GB. When disk drives exceeded this limit 48-bit addressing was adopted. This was accomplished by adding new 48-bit commands, and allowing the old parameter registers to be double loaded. That is, one could store two values into the same register and both values would be used as part of the address. AoE also allows the use of 48-bit LBA commands, allowing disks of 144,115GB, which should be enough for a while.

The ATA messages contain requests and response to perform ATA transactions. In ATA transactions, there are three possibilities: no data will be transferred, data will be written into the disk, or data will be read from the disk. Nothing in the parameter registers indicates which one of these operations will occur. When the *W* flag in the ATA message is set to one it indicates that there will be a transfer *to* the disk drive.

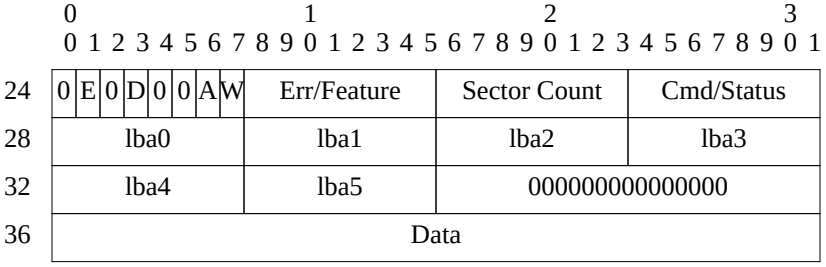

Figure 2. The ATA Header

The ATA message contains values to put into the registers, and the flags field to controls how the values are used. See Figure 2. If the *E* bit in Aflags is zero, 24-bit addressing is used. The lba values are copied into the address registers, the feature and sector count value are copied into the appropriate registers. Last, the command is copied into the command register which starts the disk transaction. The AoE server watches the status register, and after the completion of the command, copies all of the register into the response header. The error register goes into the Err/Feature field and the status register is copied into the Cmd/Status field. The response from an ATA message will contain the values of the status and error registers, along with any residue of the parameter registers.

If the *E* bit in the ATA message is set, the 48 bit double loading of the parameter registers are used.

Since the ATA commands are carried over Ethernet, and since there is a constraint on the size of the

frames on an Ethernet network, we can't send more than two 512 byte sectors in a single message. This means that the sector count will be less than or equal to two. Even with the 1GE and 10GE jumbo frames, we can't move more than 8192 bytes, or 16 sectors. So, in 48-bit mode we don't bother with the upper byte of the sector count register. This may seem overly restrictive, but one of the authors has written an NFS client and is sensitive to the number of zero bits encoded in a protocol.

The *A* bit, when set to one, indicates to the AoE server that it can respond as soon as a write command is received. Queuing the writes to the disk can improve performance by allow more data transfers to be done in parallel, and allow for the possibility of the AoE server transferring data from multiple write requests in a single disk transaction.

This is all there is to the ATA messages. It simply exports that ATA interface on AoE servers to the client hosts.

4. Config/Query Information

The naming scheme using shelf and slot numbers described above is more than adequate for a few client hosts hooked to dozens of AoE servers. But as the number of client hosts and AoE servers increases this scheme becomes less attractive. The second class of AoE messages, the Config/Query messages, enables a more advanced naming scheme.

On each AoE server there is a small amount of nonvolatile memory that can be used by the client servers to save configuration data. This configuration data can be any arbitrary binary values, and have a length from up to 1024 bytes. This information has no meaning to the AoE server; it is merely a place for client hosts to stash information. This data is not stored on the disk to avoid any interaction with any server software. Disks from client hosts, for example, can be moved to an AoE server and still be accessed, without having to reform the information. The Config/Query messages use this data in two ways. First, the config data can be conditionally or unconditionally set. Second, there are various ways to query for the data.

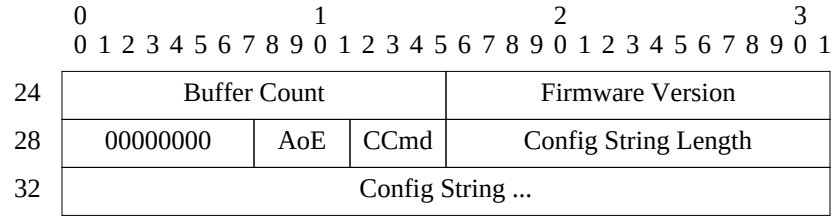

Figure 3. The Config/Query Header

Setting the config data can be done in two ways. First, a set request will set the config memory to a value *only* if the current config data is zero length. If the config memory already has non-zero length data, the set request will fail. A second command can force the AoE server to set the config memory. The intended config values can be zero length thereby allowing the config data to be reset.

Query messages can contain query data that is used to optionally match against the stored config information. The data can be queried in three different ways. First, the config data can be unconditionally read. Second, the config data can be returned only if the query data matches the prefix of the config on the AoE server. That is, the query has a number of bytes that is less than or equal to the length of data in the server, and the bytes must match the bytes in the server. The AoE server responds with the complete config information.

The last query request requires an exact match of the stored information. The data in the request and the data in the server have to match both in content and length. This is useful for broadcast queries where one is looking for a specific AoE server.

So, a possible scenario is as follows. A pool of AoE servers are introduced into the system, all having zero length config data. A client host wants to provision more storage, so it broadcasts for an exact match with a query of zero length. All of the new blades will respond since they have no config data. The client host then randomly chooses a server and sends some config information to be conditionally set.

Normally the data is stored and the client host gets a positive response. However, if two client hosts simultaneous query for zero length config data and see the same responses, both might pick the same server to claim. They would both send the conditional set config command, but because of the serial nature of the Ethernet, only one request would be first to the AoE server and succeed in setting the config data. The second set request would fail since the length of the data would no longer be zero. In this way, the blades themselves do simple yet sufficient arbitration.

Next, consider RAID software that has claimed a set of servers to form a RAID device. The RAID software could set each AoE servers config string to a two part string. The first part could identify the RAID set, and the last part could identify the AoE servers position in that set. When client hosts later reboot they could send a prefix query with just the unique RAID name and would receive responses from all the blades in the set. These responses would have all the config data so each response would identify their location in the set.

These are just two ways the config/query features of AoE can be used. It is hoped that others will find more uses for them. These features were designed to be flexible without being so general they would be hard to use.

Table 1 Config Commands

Value	Description
0	read config string
1	test config string exact match
2	test config string prefix match
3	set config string
4	force set config string

5. Device drivers

The intention of the AoE protocol is to provide local access to disks on a local area network. To this end, block device drivers are used to access the protocol. This is different that other networking protocols in that to the host, the network service appears as simply a local disk. There are no Sockets. There are no socket system calls. Just block special files in the `/dev` directory.

In the Linux 2.6 AoE drivers, the special files appear in the directory `/dev/etherd`. The names of these files link them to specific physical locations of AoE blades. The name, for example, for the blade in shelf 3, slot 6 would be `/dev/etherd/e3.6`, and the second partition on that blade would be `/dev/etherd/e3.6p2`. This allows a very simple way to manage the blades in a network.

In addition there are a couple of character special files in the directory. The current status of AoE blades can be listed by reading the file `/dev/etherd/status`. /dev/etherd/error `will copy out any logged driver error messages.`

6. Conclusion

The AoE protocol is a flexible yet simple protocol that allows ATA disks to be directly connected to a local area network. It is simple to implement, so the costs of the AoE servers can be very low. Because of that, there can be a lot of AoE servers on a single local network segment. The use of commodity network to bind disk drives with client hosts allows for true network storage instead of the expensive Fibre Channel and systems both closed and with high common equipment costs.

The use of local Linux and BSD drivers has worked out well, and the performance has been encouraging. Keeping the contents of the disk transparent to the AoE server allows users to know exactly what is on the disk and move them from direct attached to the client hosts to AoE servers without having to copy or reformat data. They can always get their data off the blades.

The decision to use simple Ethernet frames requires some explanation. We view the use of storage blades as local storage, not Internet storage. As a result we use the simple data link services of a local area network to transport the messages. We are not trying to make storage available directly across an internetwork as there are already solutions for that problem. For internetwork storage the use of iSCSI or NFS

would be more appropriate.