M.I.T. Laboratory for Computer Science

Request for Comments No. 219
March 19, 1982

VAX Remote Virtual Disk Protocol

by Michael Greenwald

This is a draft of the proposed VAX Remote Virtual Disk Protocol. It is vague in many places where exact details have yet to be worked out. It is a working guide rather than a protocol specification.

Comments are solicited.

1. Introduction

1.1. Motivation

The VAX Remote Virtual Disk Protocol is designed for use in an environment consisting of many identical VAXen. It is intended to provide the ability to dynamically attach arbitrary disks to arbitrary computers. It is especially useful when the VAXen are physically remote, or user intervention is impractical (local disks are non-removable).

1.2. Scope

The VAX Remote Virtual Disk Protocol provides no more functionality than simply allowing additional disk drives to be attached to your machine. It does not implement sharing, or any form of object storage, or naming mechanism, other than that found on any disk drive.

It does not guarantee reliable writes to the disk and depends on higher level applications for all but the most rudimentary reliability checks.

1.3. Interfaces

This protocol is layered on top of the DOD IP protocol. An IP protocol number has yet to be assigned.

This protocol is intended to be used by the device driver for a disk on the user machine. The server presumably is running this protocol in a separate module.

For example, to be used in VAX UNIX, the device driver (for device VD01 - virtual disk 1) would convert the standard disk reads and writes to protocol requests.

Currently, our plan for the server is to implement it as a stand-alone module in a VAX with several large disks.

2. Specification

There are two categories of packets - requests and responses. There are four types of requests:

~ SPIN-UP

~ SPIN-DOWN

~ READ

~ WRITE

There are four types of responses:

~ ACK

~ ERROR

~ BLOCK

~ WRITEACK

The format of individual packets follow.

2.1. Packet Format: SPIN-UP

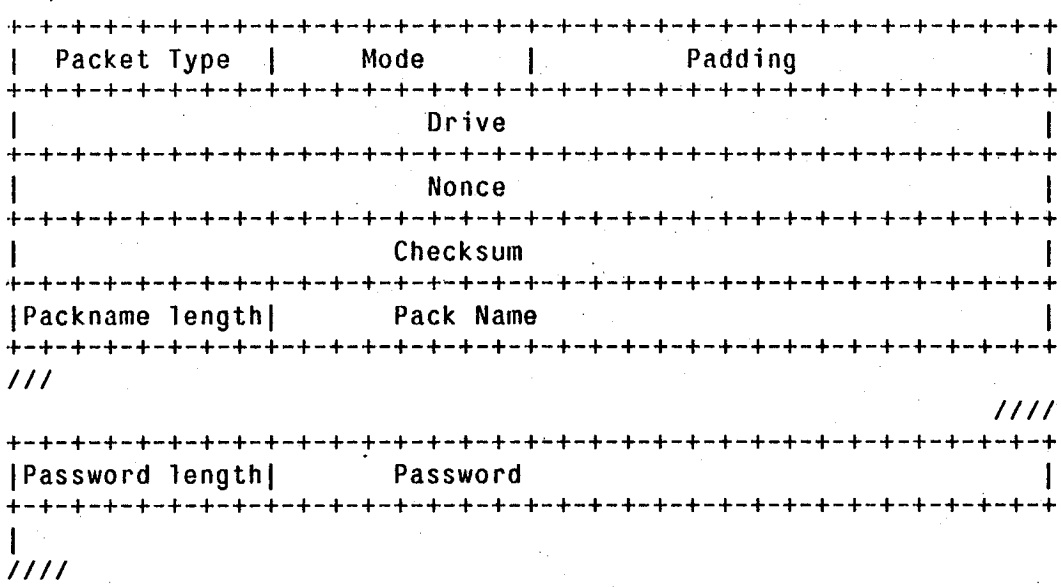

Figure 1: Format of SPIN-UP Packet

Packet Type should be SPIN-UP.

Mode should be one of READ-ONLY, SHARED, or EXCLUSIVE.

If a user wishes to spin up virtual disk "Foo" in his local drive 12, then he sends a SPIN-UP packet to the server. He fills in Packet Type with SPIN-UP, and Mode with a valid mode. He fills in Drive with the 32 bit integer 12. Nonce is meant to be a Unique ID (UID).[1] The Checksum is the sum of the 32 bit words in the packet, modulo 2 to the 31. If the packet does not end on an even 32 bit boundary, then *for the purpose of the checksum only* we assume that the packet was filled out by zero bytes.

In this case PackName Length would be 3, and PackName would be "Foo", with each character converted to an 8 bit byte. If the mode were, for example, READ-ONLY, then the user would fill in Password with the READ-ONLY password for this diskpack. If the password were "bar", then Password Length would be 3, and Password would be "bar".

Upon receipt of the SPIN-UP packet, the server would attempt to fullfill the request. If

---

[1] Since it is only a 32 bit number, it is only unique over a fixed period of time.

everything is correct, (the disk exists, the password is correct and so on), then the server associates virtual disk "Foo", with drive 12 from host H. From now on, any reference from host H to drive 12 will refer to virtual disk "Foo". The server then sends an ACK packet back to the user.

If the server detects an error, it sends an ERROR packet back to the user. This ERROR packet can be cause by many different classes of errors. First, "real" disk errors - for example the physical disk containing "Foo" is trashed, or any error you might get accessing a physical disk. This is different than the typical case of a computer connected to a physical disk, because there, you would not detect the error until you tried to perform an operation on the disk.

There can also be errors because of bad arguments - a non-existant virtual disk, a bad password, or a bad checksum. There can also be inconsistent arguments - you already have a disk spun up in drive 12, or someone else has disk "Foo" spun-up in READ-ONLY mode.

2.2. Packet Format: SPIN-DOWN

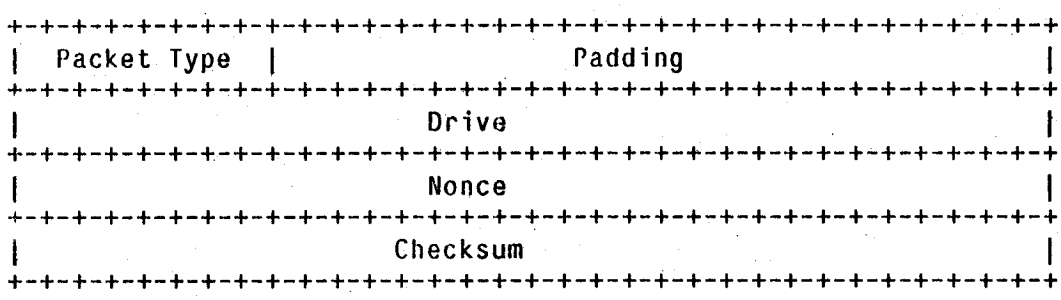

Figure 2: Format of SPIN-DOWN Packet

Packet Type should be SPIN-DOWN.

If a user wishes to spin down the virtual disk in his local drive 12, then he simply sends a SPIN-DOWN packet to the server. He fills in Packet Type with SPIN-DOWN, and he fills in drive with the 32 bit integer 12. Nonce is meant to be a Unique ID (UID). The Checksum is over the entire packet. For purposes of computing the checksum, the Checksum field is considered to be zero.

Upon receipt of the SPIN-DOWN packet, the server would attempt to terminate the association between host H, drive 12, and virtual disk "Foo". If this worked correctly, the server sends and ACK back to the user. If the drive was not spun up, or there were some other error, then the server sends back an ERROR packet. It is not polite for a user to consider his disk spun down until he receives the ACK from the server.

2.3. Packet Format: READ

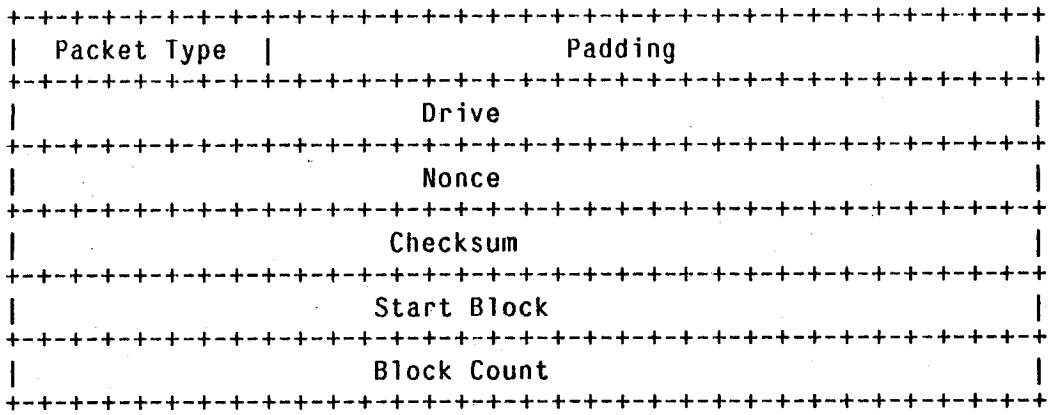

Figure 3: Format of READ Packet

Packet Type should be READ.

Drive should be the drive that you wish to read from.

Nonce should be a UID.

Checksum should be the 32 bit checksum of the entire packet.

Start Block should be the number of the first block that you wish to read.

Block Count should be the number of blocks that you wish to read.

If a user wants to read data from local drive number 12, (which is associated with virtual disk "Foo"), then his request gets translated into a READ packet with the appropriate values. The Drive field is filled with the drive number (12 in this case), and the server knows to associate host H, drive 12, with virtual disk "Foo". Start Block is the number of the block on the virtual disk that he wants to start reading from, and Block Count is the number of blocks that he wants to read.

Upon receipt of a READ packet, the server tries to send the data requested to the user. If the drive is spun up, and the Start Block is within the bounds of the disk, the the server responds with a BLOCK packet. A BLOCK packet contains a block of data, a block number, and a 32 bit status word.

If the server detects an error, he still sends a BLOCK packet, although it has a non-zero status

word. If the status word is non-zero, then the data is undefined. Errors can be the result of physical errors on the disk, or any of the assorted things that can go wrong on a disk.

The only time that the server sends an ERROR packet in response to a READ, is in the case of a malformed packet. Basically, protocol errors cause ERROR packets, and typical disk errors cause non-zero status word.

READs can timeout. This protocol does not guarantee delivery of packets. We assume that most packets will reach their intended destination, but we make no guarantees. It is up to the user to treat READ timeouts the same way he would treat physical disk timeouts.

2.4. Packet Format: WRITE

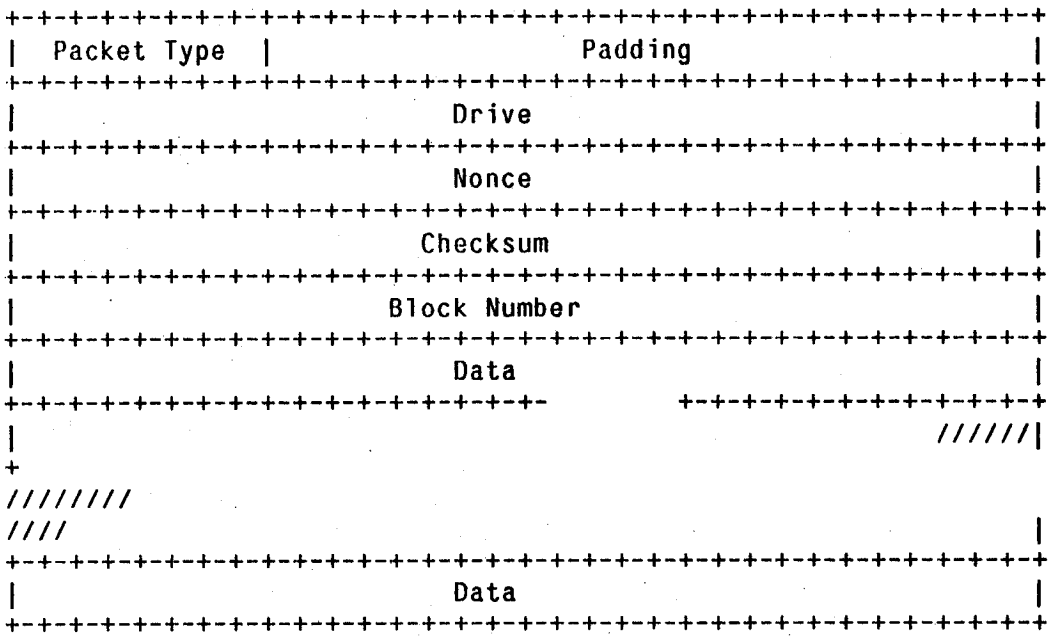

Figure 4: Format of WRITE Packet

Packet Type	should be WRITE.
Drive	should be the local drive number that you wish to write to.
Checskum	should be a checksum of the entire packet *including the data.* If a WRITE packet arrives successfully then the data is assumed to be valid.
Block Number	is the number of the block that you wish the data to be written into.
Data	is 512 bytes of data that are to be written into the specified block.

If a user wishes to write data to the virtual disk, he sends a WRITE packet to the server. The WRITE packet includes both the request for a write, and the data in the same packet.

The checksum for the packet is taken to be the "guarantee" that the data is uncorrupted.

The server sends a WRITEACK if the write was a success. Our tentative plan is to send the WRITEACK *after* the data is successfully written to physical disk. If this proves to be an efficiency bottleneck, then we will probably institute two separate write requests, one guaranteed to return only when the data is physically written, and the other returning as long as the server is willing to buffer the data to be written at some indeterminate time in the future.

The WRITEACK includes a 32 bit status word. In the case of most errors, a WRITEACK with a non-zero status word will be returned. Some errors, however, will cause ERROR packets to be sent. This will generally be because of inconsistent arguments.

WRITEs can timeout. Again, the user is expected to deal with a WRITE timeout in the same way in which he would deal with a disk timeout.

2.5. Packet Format: ACK

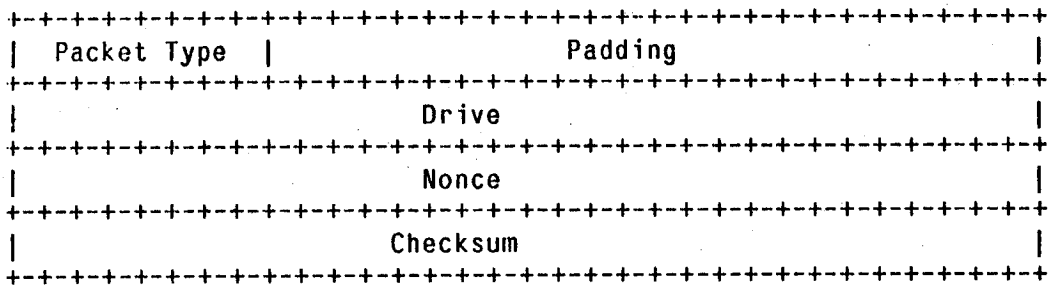

Figure 5: Format of ACK Packet

Packet Type — should be ACK.

Nonce — should be the value of the Nonce in the packet to which you are responding.

Drive — should be the drive number specified in the packet that you are acknowledging.

Checksum — is the 32 bit checksum of the entire packet.

2.6. Packet Format: ERROR

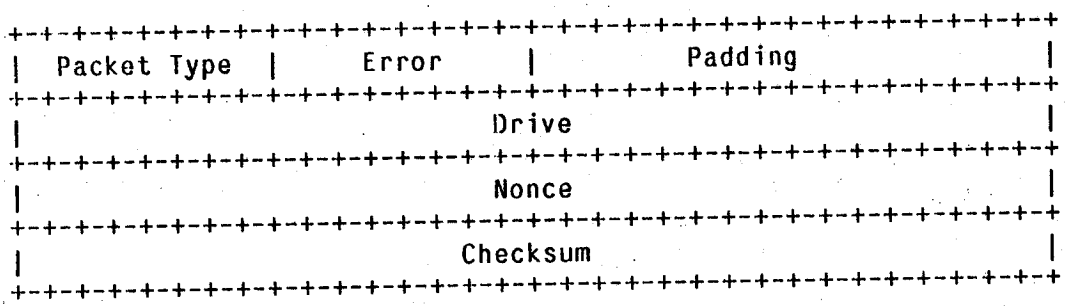

Figure 6: Format of ERROR Packet

Packet Type — should be ERROR.

Error — should be the type of error. It must be one of the constants defined in section 2.10.3.

Drive — should be the drive number specified in the packet that generated the error.

Nonce — should be the same as the value of Nonce in the packet that generated the error.

ERROR packets are sent out when the server wants to tell the user that some error has occured. They are usually sent when the error was in the protocol, or some high level thing wrong with the virtual disk system. Errors that are roughly equivalent to those that a physical disk would give are typically returned in the 32 bit status word that is part of BLOCK and WRITEACK.

2.7. Packet Format: BLOCK

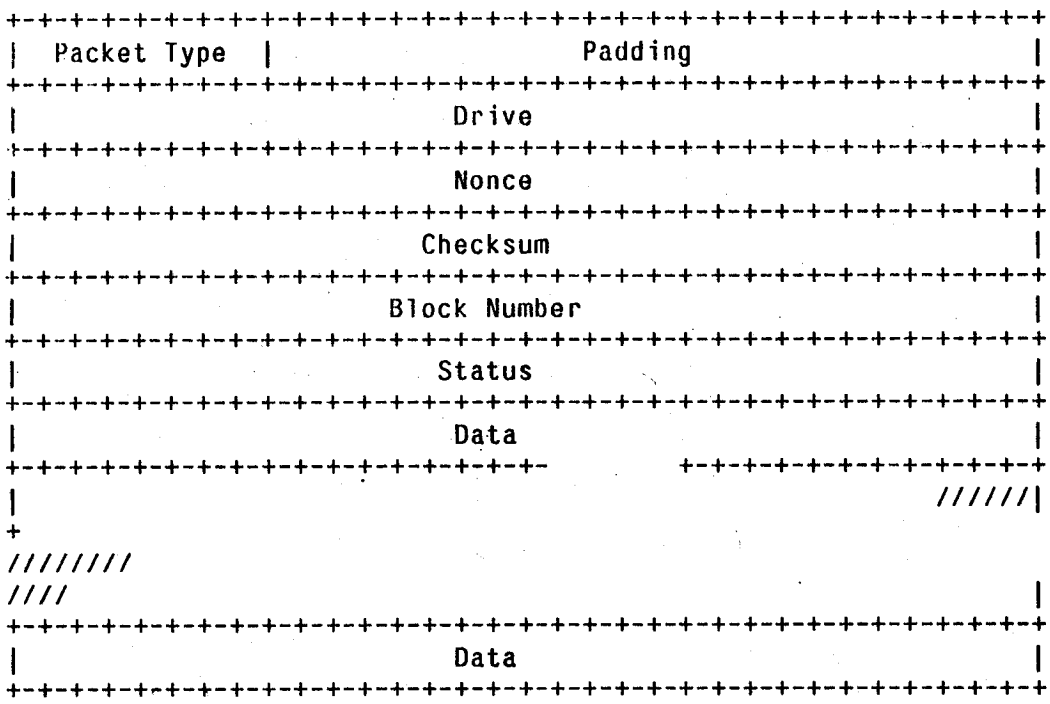

Figure 7: Format of BLOCK Packet

Packet Type	should be BLOCK.
Drive	should be the disk drive specified in the READ request that this BLOCK is being transmitted in response to.
Nonce	should be the same as in the original READ request. (Even in cases where the read was for multiple blocks.)
Block Number	identifies this block in the virtual disk.
Status	is a 32 bit status word describing the status of thvirtual disk. Most errors are reported through this word.
Data	is a 512 byte block of data, read from the virtual disk.

The BLOCK packet is the response to the READ request. It includes the data requested, and enough information to allow the user to determine which request this is in response to. (Note that there can be many outstanding READ requests, even in the case where we restrict access to a drive to a single outstanding request - a user can always have requests out to more than one drive).

The checksum is over the entire packet and is the only check we have on the validity of the data. We assume that if the checksum is correct then the data is the same as on the disk.

In case of an error, the server fills in the approriate bits in the Status word.

If the server receives a malformed READ request then the server responds with an ERROR packet.

2.8. Packet Format: WRITEACK

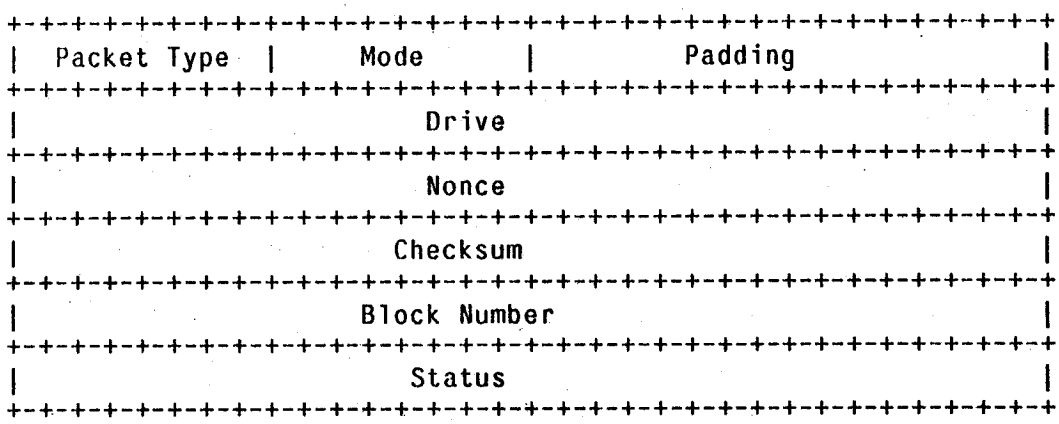

Figure 8: Format of WRITEACK Packet

Packet Type — should be WRITEACK.

Drive — should be the same as the drive specified in the WRITE request.

Nonce — should be the same as the Nonce specified in the WRITE request.

Checksum — is the 32 bit checksum of the entire packet.

Block Number — is the number of the block that you are acknowledging having written.

Status — is the 32 bit Status word that represents the state of the disk, and reports errors in the write procedure.

WRITEACK is the response to the WRITE request. It signals the completion of the WRITE request. Currently the WRITEACK is not sent until the after the block was written out to the physical disk.

Errors are reported through the Status word.

2.9. Definition of Fields

Block Count — A 32 bit integer representing the number of blocks that you wish to read. Legitimate values are from 1 to (NUMBER-OF-BLOCKS-ON-DISK - Start-Block + 1).

Block Number — A 32 bit integer representing the number of the block of data enclosed in the packet. Disk blocks are numbered from 0 to $2^{31} - 1$.

Checksum — A 32 bit checksum of the packet. The checksum is computed by adding together all the 32 bit words in the packet. The Checksum field is considered to be zero for this computation. If the packet does not end on a 32 bit boundary, then the checksum computation assumes that the packet is padded out by zeros. The low order 32 bits are then used as the Checksum (The 32 bit sum is taken modulo 2^{31}).

Data — A 512 byte block of data. The bytes are ordered in the packet exactly the way they are ordered on disk. The bits in a byte are ordered according to the IP specs.

Drive — An index that represents the drive number on the users machine. It is an integer between 0 and $2^{31} - 1$, and is encoded as a 32 bit binary integer. Drive numbers are unique on a given host, but will probably be duplicated over the net. The virtual disk is specified by the host-drive pair.

Error — An 8 bit byte representing the type of error that the ERROR packet is reporting. The values are defined in "Error Types", Section 2.10.3.

Mode — An 8 bit description of the mode that the virtual disk will be opened in. Mode can currently be one of three values - READ-ONLY, SHARED, or EXCLUSIVE.[2] Opening in SHARED mode, gives you read and write access to the disk, and allows other users to also access it in SHARED mode. Opening in EXCLUSIVE mode, gives you read and write access to the disk, but locks the disk, so that no other user can touch the disk while you have it spun up. READ-ONLY gives you read access to the disk, allowing anyone else to share it with you, as long as they too, are in READ-ONLY mode.

Nonce — A 32 bit unique identifier. The Nonce is an integer between 0 and $2^{31} - 1$, encoded as a 32 bit binary integer. It is assume to be unique for an interval of time several times the lifetime of a packet.

Packet Type — An 8 bit byte describing the type of packet. Currently defined values are: SPIN-UP, SPIN-DOWN, READ, WRITE, ACK, ERROR, BLOCK, and WRITEACK.

---

[2] See "Constants", Section 2.10

Pack Name — An ASCII string represeting the name of the virtual disk pack that you wish to associate with the specified local disk drive.

Packname Length — The length of the ASCII string representing the Pack Name. The length field is a byte-count that only includes characters in the string. The length field itself is *not* included in the byte – count.

Padding — Zero bytes used to fill fields out to even 32 bit boundaries.

Password — An ASCII string representing the capability to access the Virtual disk pack in the specified mode.

Password Length — The length of the ASCII string representing Password. The length field is a byte-count that only includes characters in the string. The length field itself is *not* included in the byte – count.

Start Block — A 32 bit integer representing the number of the first block of data that you wish to read. Disk blocks are numbered from 0 to 2^{31} - 1.

Status — A 32 bit word representing the status of the disk drive, and used to convey error information back to the user. Each bit will represent a different condition. The meaning of these bits will be defined later - in the next draft of this document. They are meant to cover the same status and error information that one would get back from a physical disk drive.

2.10. Constants

The following are the values of the constants referenced in the text.

2.10.1. Packet Types

SPIN-UP	1
SPIN-DOWN	2
READ	3
WRITE	4
ACK	17
ERROR	18

BLOCK	19
WRITEACK	20

2.10.2. Modes

READ-ONLY	1
SHARED	2
EXCLUSIVE	3

2.10.3. Error Types

These values will be defined in a subsequent draft.