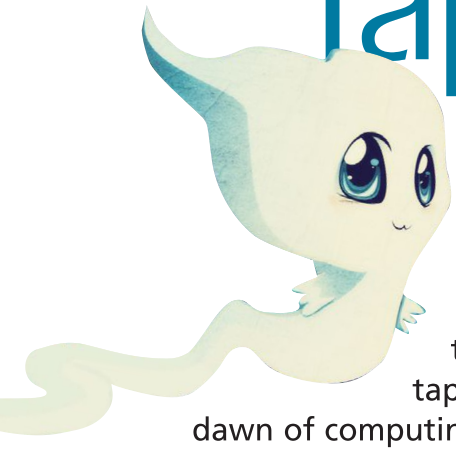
SEE TEXT ONLY

Tape's Not Dead



BY KEN MERRY

For many people evaluating storage alternatives and surveying the landscape of NAS, SAN, Object, Cloud, disk and flash storage, tape is perhaps an anachronism. Tape has been around since the dawn of computing, but does it have a place now? I believe it does. I spent more than 10 years of my career working for two start-ups that explicitly aimed to replace tape with (mostly FreeBSD-based in those cases) disk. I now work for a company that sells a lot of tape products as well as (FreeBSD-based) products that connect to disk and tape. So, I have seen both sides of the debate.

Where Does Tape Fit?

There are places where tape makes a lot of sense, and places where it doesn't. In the 1980s and 1990s, QIC and DAT technologies were aimed at home and smaller business users, and worked well enough for end-user backups. There is no equivalent small, end-user tape technology today. If you're a home user with a couple of gigabytes or maybe even a couple of terabytes to back up, tape isn't generally what you should be looking for.

Before we talk about where tape makes sense, though, it would be helpful to review some recent tape drives and cartridges, and their capabilities:

Drive	Media Type	Raw Capacity	Compressed	Raw Speed	Comp. Speed
IBM TS1155	JD	15TB	37.5TB	360MiB/sec	750MiB/sec
IBM TS1150	JD	10TB	25TB	360MiB/sec	750MiB/sec
LTO-8	LTO-8	12TB	30TB	360MiB/sec	750MiB/sec
LTO-8	LTO-M8	9TB	22.5TB	300MiB/sec	700MiB/sec
LTO-7	LTO-7	6TB	15TB	300MiB/sec	700MiB/sec
Oracle T10000D	T2	8.5TB	21.25	252MiB/sec	800MiB/sec

• The costs of the above drives and media vary widely, especially when you consider that most drives will be in a tape library, and the tape libraries will usually have multiple drives. The more data you have, the more tape makes sense. At large scale, the cost of the media is generally larger than the library and drives.
• Tape provides a number of advantages over disk and flash for storage:
• The cost per GB of tape is very low, generally lower than the cheapest disk-based backup.
• Tape makes a great archive storage media. If you have data that isn't used very often, you can store it on tape and save your expensive disk or flash arrays for frequently used data.
• The throughput is higher than most spinning disk drives. (Although at any scale, you have to consider the total number of tape or disk drives and the interconnect characteristics to evaluate the overall system throughput.)
• Tapes are designed to be moved around and stored. Disks aren't generally designed with that in mind. (Although they can be used that way.) This makes them great for off-site backup.
• Tapes generally have a 30-year shelf life. You won't find disk manufacturers making that claim.
• Tapes are good for air-gapped storage. Are you worried that a software bug, a malicious person, or ransomware will destroy your data? Back it up on tape and put it in off-site storage. Those threats can only destroy data that they can access.
Tapes provide a certain amount of genetic

diversity in storage. What would happen if you had a disk or flash drive firmware bug that corrupted data on all of your drives? What would happen if you had a filesystem bug that corrupted all of your data? With tape, you have different firmware, different media, and different software storing the data. As long as you take the time to back up, you can recover from a drive firmware bug or a filesystem or OS bug.

Tape is not a good fit for data that needs to be accessed very quickly (e.g., less than 10 seconds) or randomly. For those types of data, disk and flash are a better choice.

Tape in FreeBSD

FreeBSD has had tape support since it was the 386BSD patchkit. The original FreeBSD SCSI layer was written by Julian Elischer, who also wrote the st(4) tape driver. Justin Gibbs and I wrote FreeBSD's CAM (Common Access Method) SCSI layer, which went into the FreeBSD tree in 1998 and included the sa(4) (Sequential Access) tape driver. (Justin wrote most of the sa(4) driver.) Matt Jacob rewrote the sa(4) driver substantially starting in late 1998, and much of what it is now is due to his work. Matt was the maintainer of the driver for many years. I am the de-facto maintainer now.

The sa(4) driver supports tape drives that range from old SCSI-1 9-track tape drives to the latest Fibre Channel attached IBM TS1155 tape drives. It includes modern density support, using the mt(1) getdensity subcommand. You can use that command to ask a modern tape drive what kinds of cartridges it supports and in which formats. That is very helpful for IBM TS (and now LTO) tape drives, where one cartridge can have multiple available formats with different capacities.

The sa(4) driver also supports unmapped I/O. Unmapped I/O buffers are userland buffers that are not mapped into the kernel's virtual address space. Instead, they are translated directly to physical pages, which are then transmitted to the FC/SAS/SCSI controllers firmware for DMA. Mapping user buffers into kernel virtual address space produces slowdowns on modern machines with lots of cores due to the TLB (Translation Lookaside Buffer) shootdowns necessary to update the cache in each core. Unmapped I/O allows data that the kernel doesn't need to touch to pass through the kernel without being mapped, avoiding the TLB shootdown. With lots of I/O activity, avoiding TLB shootdowns can boost performance noticeably.

Handling Tape Libraries

FreeBSD has had built-in support for tape libraries via the ch(4) driver and chio(1) utility since the transition from the original SCSI layer to CAM in 1998. I ported the ch(4) driver and chio(1), which were written by Jason Thorpe, from NetBSD to FreeBSD/CAM. The ch(4) driver supports everything from very old libraries to the latest tape libraries.

The mtx(1) utility (in ports/misc) can also control a tape library and operates via SCSI passthrough.

While chio(1) and mtx(1) are good for command line control of a library, a backup application with tape library-level support (like Bacula or Amanda) will use chio(1) or mtx(1) to move tapes between slots and tape drives in a tape library.

Data Integrity

A backup isn't much good if the data gets corrupted; data integrity is very important. You want to be sure that your backups work and that all the bits make it to tape in the right order.

The tape API presents unique challenges for storage protocols like SCSI. When the OS is talking to a hard drive via SCSI, hardware, software, or firmware failures can sometimes cause commands or data to get dropped. If that happens, the driver times out the outstanding command, aborts it, and sends status up to the CAM midlayer so that the command can be retried if the user requested retries. In the case of a timeout on a write, the command and data may or may not have reached the drive originally, but resending the command does no harm. The hard drive will simply rewrite the data again and move on.

Tape is a sequential medium, though. Writes are sent to tape without an explicit logical block address. The address is implicit. Each write (or read) advances the block number down the tape. If a write sent to a tape drive times out, the OS has a dilemma: what made it down to the drive? Did the entire block make it down, and we simply didn't get status back? Did the drive get part of the data? Did the drive get none of the data? Retrying without knowing what made it to the drive could lead to data corruption. The alternative would seem to be aborting the entire backup job with an error.

The ANSI T10 (t10.org) and T11 (t11.org) committees, who write the SCSI and Fibre Channel specifications, respectively, came up with a solution to the problem. It was originally called FC-

Tape, and is now included in the FCP-4 (Fibre Channel Protocol) specification from T10. Sections 4.4 through 4.7 of the spec outline Fibre Channel features needed to assure data integrity for tape drives. They are:

Precise Delivery of Commands

This feature provides a way for an initiator (server) to specify a CRN (Command Reference Number) with each command. The CRN is a number from 1 through 255 that wraps around. If the target (tape drive in this case) receives a command that is not in sequence with the previous command and CRNs are enabled, it will reject the command. This insures that tape I/O requests are executed in order.

Confirmed Completion of FCP I/O Operations

Confirmed completion is a Fibre Channel option that lets the Target (in this case, a tape drive) request that the Initiator (the server) tell the target that it has received a status response. This lets the target know that it does not need to retransmit status to the initiator.

A normal SCSI write command sequence goes like this:

```
Initiator -> Target: I want to write 512KiB.
Target -> Initiator: Go ahead and Transfer the data.
Initiator -> Target: Transferring 512KiB of data.
Target -> Initiator: Data successfully accepted into cache.
```

At this point, the Initiator knows that the Target has completed the write. The Target knows that it has completed the write, but does not know whether the Initiator received the message. Confirmed completion adds another message:

```
Initiator -> Target: I got your completion message.
```

Retransmission of Unsuccessfully Transmitted IUs (Information Units)

If a read or a write command to a Fibre Channel device is taking too long, this feature gives the Fibre Channel driver and/or firmware a mechanism to do link level error recovery without the big hammer of timeouts, aborts, and retries (or rather failures in the case of tape).

If a target (tape drive) takes more than 3 or 4 seconds to respond to a command or a data transmission, the initiator (server) can send REC (Read Exchange Concise) ELS (Extended Link Services) Fibre Channel command to the target to find out the status of the command.

If the target never got the command, the initiator now knows that (as opposed to not knowing what got dropped) and can retransmit it. If the target only got half the data, the initiator now knows that, and can retransmit the data using the SRR (Sequence Retransmission Request) ELS command if the target supports SRR.

This allows link level error recovery in a time frame that is much smaller than typical tape timeouts. The sa(4) driver uses a 32-minute timeout for read and write commands. That is the recommended timeout value from an IBM LTO-5. Other drives are similar. To see the timeout values for your tape drive, try this:

```
camcontrol opcodes sa0 –T
```

So if a problem happens on a link, without FC-Tape, the FC driver will wait 32 minutes, abort the I/O, and then send a failure notification back up the stack to the sa(4) driver. There are no retries, because you can't safely retry tape I/O operations for reasons we've outlined.

With FC-Tape, though, the FC driver or firmware will send a REC to find out the status of the command, and assuming the drive is still responsive, the initiator and target can get the transfer moving again in seconds as opposed to minutes.

This capability can also help ensure reliable tape access in a Fibre Channel environment with reliability issues. An analogue from the network world is using TCP (Transmission Control Protocol) on top of IP (Internet Protocol) to ensure reliable transmissions on a network that can sometimes drop packets.

Task Retry Identification

Task Retry Identification is a mechanism for identifying a retry sequence. This is a necessary addition because of the way commands are identified in Fibre Channel.

In Fibre Channel, the initiator (server) gives an OX_ID (Originator Exchange ID) to each SCSI (or other) command it sends to the target (tape drive). When the target responds to the initial command, it replies with the same OX_ID and adds its own RX_ID (Receiver Exchange ID). The OX_ID and

RX_ID are how a particular command is identified from then on. When data is transmitted to the target, the initiator specifies the OX_ID and RX_ID so that the target knows which command the data belongs to.

The OX_ID and RX_ID are 16 bite values can quickly wrap around. For this reason, the Task Retry Identifier is specified along with Fibre Channel commands (FCP_CMND), REC, and SRR to tie the error recovery actions together and make it clear which command they're referring to.

FreeBSD FC-Tape Support

FreeBSD has three Fibre Channel drivers, two of which are in the tree and one that will be in the tree soon.

The isp(4) driver was written by Matt Jacob and covers Qlogic controllers from Parallel SCSI all the way to 16Gb Fibre Channel. Matt added FC-Tape support in 2012 thanks to a contract from Spectra Logic Corporation.

The mpt(4) driver includes support for older LSI FC controllers up to 4Gb, but does not support FC-Tape.

The ocs_fc(4) driver, which was written by Broadcom (formerly Emulex), should be committed to FreeBSD soon. It supports Broadcom's 16Gb and 32Gb Fibre Channel cards and does support FC-Tape.

SAS

Fibre Channel includes a number of link level error recovery features for tape drives, but what about SAS?

SAS includes a feature called TLR (Transport Layer Retries) that is similar to the FC-Tape features. It can retransmit commands and data as needed. For more details about how this works, get the SPL-4r12 (SAS Protocol Layer 4, revision 12) specification from t10.org and look in section 8.2.1.

The mps(4) (LSI/Broadcom 6Gb SAS) and mpr(4) (LSI/Broadcom 12Gb SAS) drivers in FreeBSD support TLR and turn it on for tape drives.

Bit Error Rate and Checksums

Another topic that comes up when talking about hard drives and tape drives is Bit Error Rate. The Bit Error Rate is the likelihood of a tape or disk drive returning a byte to the host that is incorrect. You can also think of it as the likelihood of silent data corruption.

If your data is corrupt on the disk or tape, you want the drive to tell you about it, and not pass back bad data.

Disk and tape drives use various algorithms to reduce the likelihood of silent data corruption, and, by default, the tape algorithms are better. But, you can add checksums to both disk and tape storage to significantly reduce the likelihood of silent data corruption. (Using ZFS is an excellent way to reduce the incidence of silent data corruption with disk.)

The Bit Error Rate topic alone could fill a fairly lengthy article, so instead of doing that here, this article covers disk and disk channel bit error rate pretty well:

http://www.enterprisestorageforum.com/storage-technology/sas-vs.-sata-1.html

And this paper from the LTO consortium briefly explains that tape drives have better error detection and correction algorithms and so are much less prone than disks to silent data corruption:

https://www.lto.org/wp-content/uploads/2014/06/LTO16_0026_ValueProp_Reliability_01_2016_FINAL.pdf

Modern tape drives also support Protection Information, which is an extra CRC (CRC32 or Reed-Solomon CRC) that can be written with each tape block, and checked by the tape drive on read and write. The FreeBSD sa(4) driver supports Protection Information. The only application I know of that supports adding the CRC to each block on FreeBSD is IBM's LTFS. (See below.) See the mt(1) man page section on the 'protect' subcommand and the sa(4) man page for more details on how it works.

Users and applications are also free to write their own checksums to tape and read them back. Even if tape has a Bit Error Rate that is much better than disk, that doesn't help if the data gets corrupted before it makes it to the tape, or after it is read off of the tape. (Protection Information, above, can help in that situation, especially if it is generated in the application.)

Application Support

There are many applications that talk to tape, but there are two major open-source backup applications that run on FreeBSD and talk to tape: Amanda and Bacula. There are also other built-in tools you can use to talk to tape. And finally, there is LTFS.

Amanda

Amanda stands for Advanced Maryland Automatic Network Disk Archiver. The Amanda home page is at http://amanda.org. The Amanda server is in the ports tree at misc/amanda-server.

Amanda offers a myriad of features. One of the things I like about Amanda is that it works with ZFS snapshots. When you do a full backup, it creates a new snapshot of the ZFS filesystem and uses 'zfs send' to send the snapshot to tape. If you do an incremental backup, though, it creates another snapshot and does an incremental 'zfs send' to capture the changes from the last Amanda snapshot.

The drawback to backing up ZFS snapshots is that if you have to restore a single file, you will have to pull the entire snapshot for the filesystem in question off tape in order to restore the file. If the file is on an incremental backup, you'll have to restore the incremental and the full snapshot from tape to restore the file.

You can reduce the chances of needing to restore an individual file by establishing periodic ZFS snapshot creation. If you have hourly, daily, weekly, and monthly snapshots (pruned accordingly so you don't keep too much history), users can restore accidentally deleted files themselves. This is much faster than pulling the file off tape and hopefully requires no administrator support other than pointing the user in the right direction.

Bacula

Bacula is the other major open-source backup package available on FreeBSD. It offers a myriad of features as well. It is available in the ports tree at sysutils/bacula-server.

One of the major features of Bacula is that it does do file-based backup. If you do need to restore individual files from tape, and you don't want to pull the entire filesystem off tape, Bacula is an excellent solution. It stores the list of files in a database like Postgres or MySQL.

Built-in Tools

FreeBSD ships with several tools that natively talk to tape.

• tar(1) is not just a utility for distributing source. It stands for Tape ARchive, and tar variants have been around since UNIX Version 7. If you need to quickly send a group of files to tape, tar(1) will do that for you, and the tape should be readable on most any Unix system.

• dump(8) is the system utility for backing up UFS filesystems, and it also talks to tape. Dump will preserve the full metadata of the filesystem, and is a great tool for full backups. You can also do incremental backups with dump(8).

• dd(1) also knows how to talk to tape. If you just want to copy blocks off a tape, you can use dd(1) to do it. For example: "dd if=/dev/nsa0 of=my_tape_image bs=128k". dd will read until it encounters a file mark. You can run dd again to pull blocks off tape after you encounter a filemark.

• camdd(8) knows how to talk to tape. I wrote it as a faster, multithreaded version of dd and as an example of how to use to the asynchronous pass(4) driver interface. One of its features, though, is that it can talk to tape. For example: "camdd –i pass=da5,bs=128k,depth=8 –o file=/dev/nsa0,bs=128k". That will issue 8 128KiB reads at a time via the pass(4) driver to the disk da(5) and write those blocks in order (which is very important!) to tape one at a time.

LTFS

LTFS stands for the Linear Tape File System. This is what it seems to be—a filesystem on a tape. It is intended to be both a filesystem that you can use to read and write individual files and a standard interchange format.

LTFS uses two partitions on tape, an Index partition and a Data partition. All of the filesystem metadata, including the location of each file's blocks on tape, is stored in an XML file that lives on the index partition and is included in the data partition. When you add files to the filesystem, a new version of the index is generated and written to tape.

IBM originally wrote LTFS, and transferred the standard to SNIA (https://www.snia.org/tech_activities/standards/curr_standards/ltfs). The LTO Consortium (http://lto.org) offers compliance testing and certification for LTFS implementations, so that different vendors can insure their implementations will work with others.

IBM has offered the source for its version of LTFS (also called Spectrum Archive Single Drive Edition) that talks to IBM tape drives for a number of years. IBM BSD licensed their version of LTFS in October 2017 (it was previously licensed under the LGPL), and it is available here:

https://github.com/LinearTapeFileSystem/ltfs

I ported IBM's LTFS to FreeBSD in 2013 (sponsored by Spectra Logic), and I am in the process of preparing that work for a pull request so that it can go into IBM's tree.

LTFS works with LTO-5 drives and higher. It requires tape partition support and a big enough flash chip on the tape (called the MAM—Media Auxiliary Memory).

IBM's LTFS uses FUSE (Filesystem in Userspace) to present a file interface. LTFS itself runs as a userland process, and FUSE forwards the VFS requests to the LTFS process.

LTFS isn't a replacement for a backup program like Amanda or Bacula. That is because it only talks to individual tapes. It's left to the user (or vendor) to wrap a larger application around it to manage tapes in the tape library and move data to and from the tape.

The LTFS filesystem on FreeBSD would mostly be useful as a tape interchange format. (Many media companies use it.) If you have a lot of data, you would really want a bigger application that could use LTFS as its on-tape format.

Conclusion

Tape is still alive and is actively developed and used for storing large volumes of data. Even if your data lives in the cloud, it is likely backed up on tape.

If you have large volumes of data to back up, you need air-gapped or offsite storage, or you just want some genetic diversity in your storage, tape may be a good solution. ●

KEN MERRY has been a FreeBSD committer since 1998, and a FreeBSD user since 1994. He is the coauthor of the *FreeBSD CAM I/O subsystem* and the author of *CTL, the CAM Target Layer*. He lives near Atlanta with his wife and two sons.

Thank you!

The FreesBSD Foundation would like to acknowledge the following companies for their continued support of the Project. Because of generous donations such as these we are able to continue moving the Project forward.

FreeBSD™ FOUNDATION

Are you a fan of FreeBSD? Help us give back to the Project and donate today! **freebsdfoundation.org/donate/**

Please check out the full list of generous community investors at freebsdfoundation.org/donate/sponsors

Uranium
intel

Iridium
NetApp

Platinum
VERISIGN

Gold
iXsystems NETFLIX
facebook

Silver
Microsoft vmware Tarsnap
HUAWEI STORMSHIELD