

SCSI Multipathing in OpenBSD

David Gwynne <dlg@openbsd.org>

Introduction

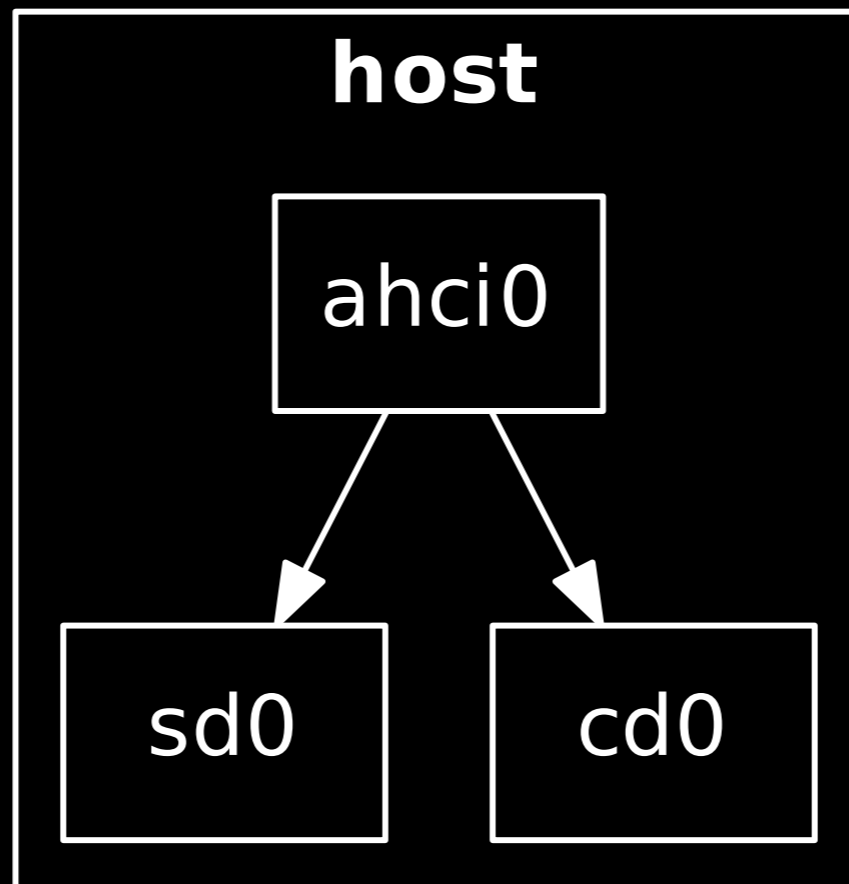
- who am I?
- what's the problem?
- what was the problem in OpenBSD?
- who has already solved it and how?
- how did I solve it?

Who am I?

- Infrastructure Architect in the Faculty of Engineering, Architecture and Information Technology at The University of Queensland in Australia
- tl;dr: I'm `dlg@uq.edu.au` in EAIT at UQ
- a core developer^{^W^W}recent slacker in OpenBSD

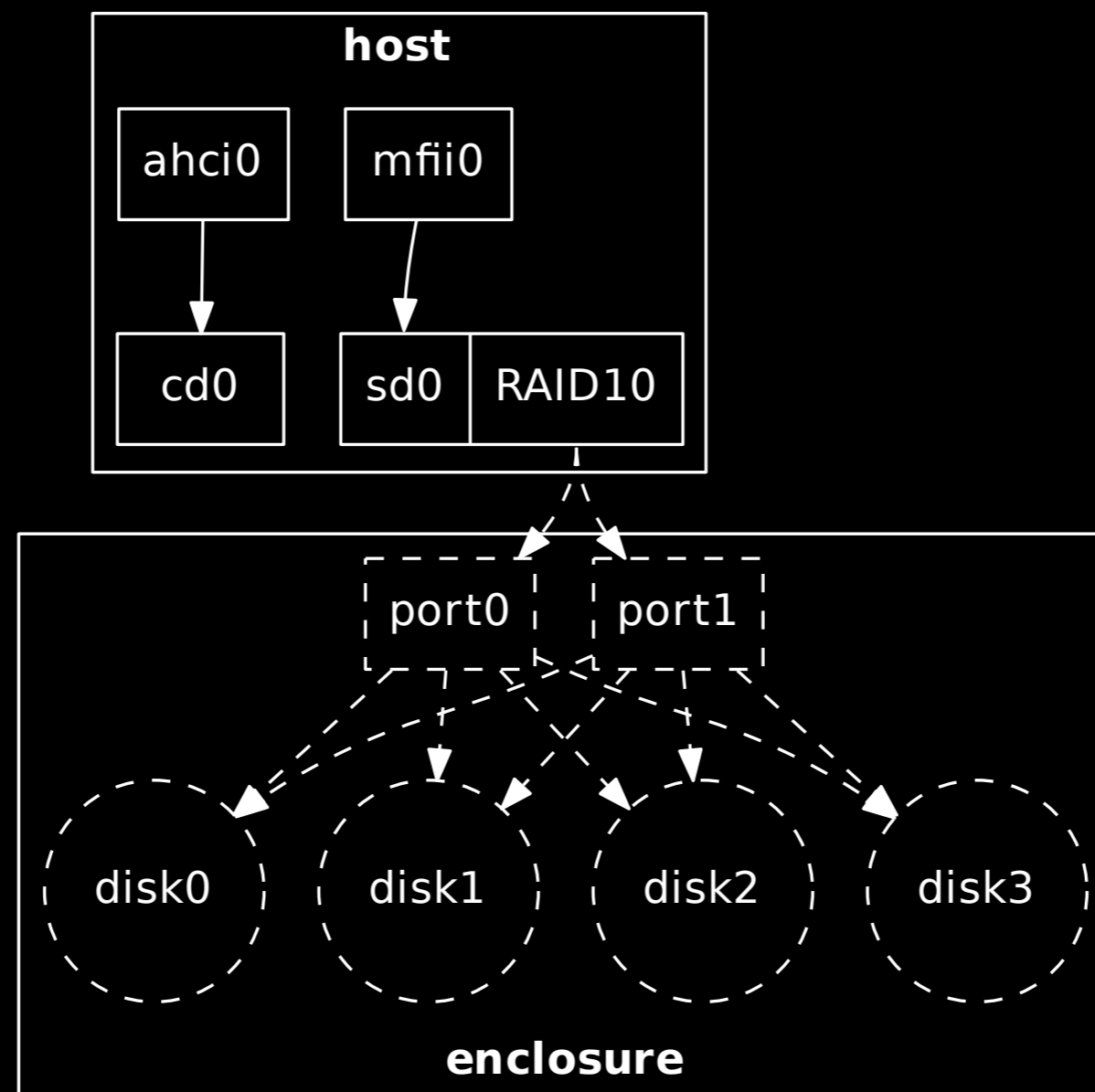
What's the problem?

- we all have something like this:



What's the problem?

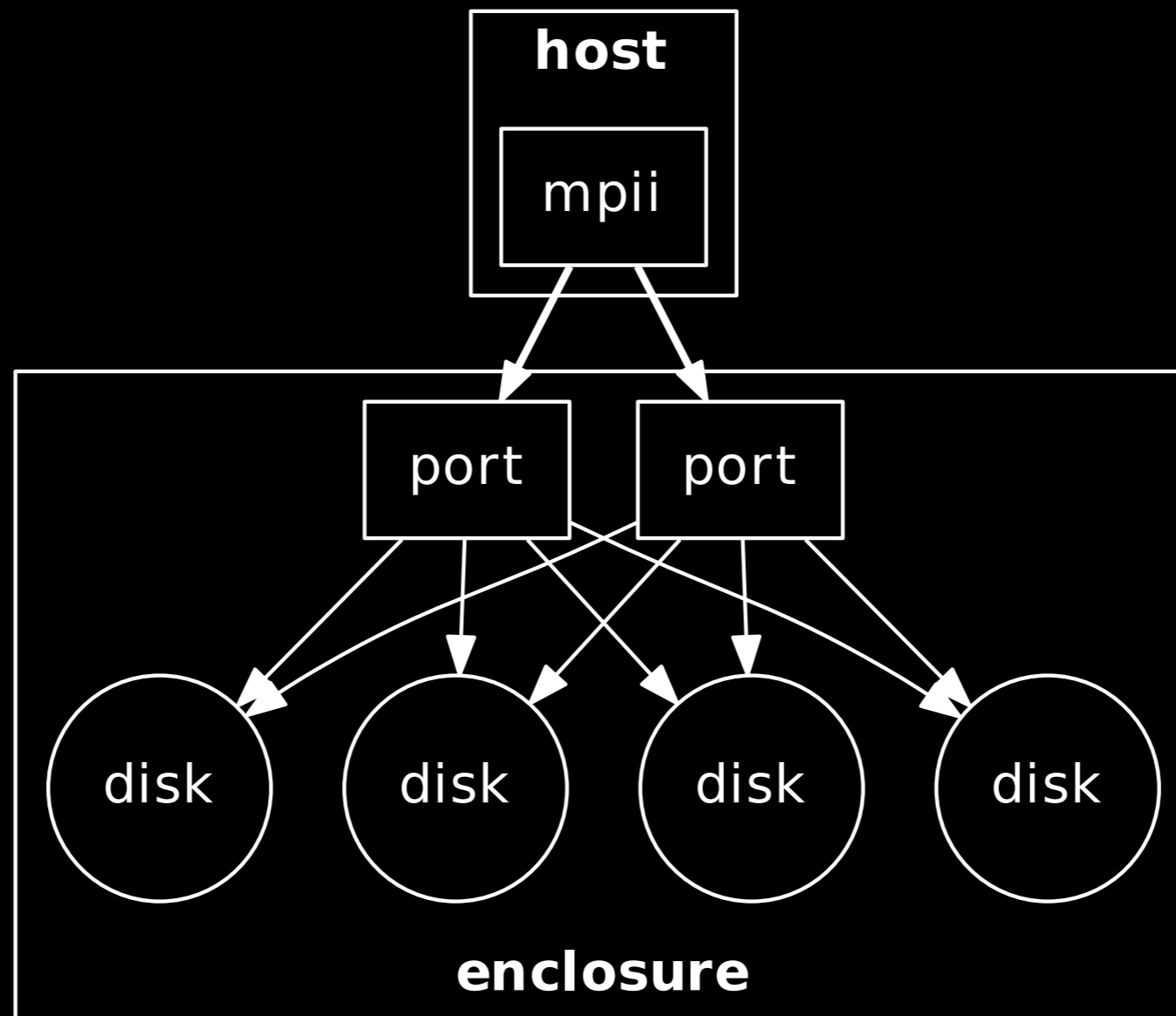
- ... or if we're lucky something like this:



What's the problem?

- in both those cases the hosts have adapters that wired straight to to SCSI devices
 - the RAID controller hides the physical complexity
 - the SCSI you see is only visible on one adapter
- but what if we replaced that mfii (hardware RAID) with an mpfi (vanilla SAS adapter)?

What's the problem?



What's the problem?

- now we see each disk twice
 - a single physical disk will attach as two separate sd(4) instances in OpenBSD
 - once per path
 - that is the problem
- so what does that mean?
- and what can we do with or about it?

SCSI Basics

- SCSI is a family of standards defined by T10
- roughly split between commands and transports
- commands are the messages between initiators (hosts) and logical units (eg, disks, tape drives)
- transports are the wires and stuff you run the commands over

SCSI Commands

- surprisingly sane
- separate command and data payload
- only the initiator (client or host) sends commands to LUs (server or device)
- commands are stateless
- targets can complete commands with extra info called sense data

SCSI Transports

- commands sit on top of transport protocols like SPI, FC, SAS, iSCSI, USB, FireWire, ATAPI, etc
 - or a RAID implementation
 - or an emulation/translation layer (eg, SATL)
- surprisingly transparent to the command layer
 - except for USB and ATAPI

SCSI Topologies

- the simplest is a target plugged into an initiator
- but...
- a computer can have multiple SCSI adapters
- a adapters and disks can have multiple ports
- you can put switches in between them
- you can put RAID in with a box full of disks
- that RAID can be "highly available"

SCSI Topologies

- yet SCSI as a protocol only cares about links between initiators and a logical units
 - SBC/RBC/SES etc don't really care
 - T10 didn't really address this until recently
- so multiple paths are a software problem
- multipath is historically expensive, complicated, and very proprietary
 - OpenBSD just isn't in that space

SCSI in OpenBSD

- *"Originally written by Julian Elischer for TRW Financial Systems for use under the MACH(2.5) operating system"*
- from 1992 or earlier - this comment still exists
- largely unchanged in OpenBSD for 15 or 17 years
- assumed LUs were only visible to a single adapter
- the SCSI stack was cut into three layers: adapters, midlayer, and targets

SCSI Adapters

- attaches the midlayer to hardware controllers
 - or SCSI command "sinks"
- takes SCSI commands from the midlayer and moves them on and off the hardware
 - isn't required to maintain target state
 - really just command transport

The SCSI Midlayer

- appears as scsibus(4) in the device tree
- target probing/attachment
- basic management of openings per LU
- API for command submission
 - biased to block I/O
 - completely synchronous unless you were doing block I/O
- grew hotplug and different sized busses

SCSI targets

- target drivers (sd/cd/st/etc) got support for modern commands (eg, big read/writes, read capacity 16, etc)
- ses was replaced with new ses/safte drivers
- had to grow hotplug like the midlayer
- ...but everything worked pretty well so didn't need fixing and nothing significant changed

Multipath gear

- at my job I get to replace expensive, complicated, and very proprietary equipment, and keep the old bits as toys
- I get to play with a lot of "big" hardware and a handful of different operating systems
- multipath gear is relatively cheap now
 - iSCSI, SAS, and 2nd hand FC gear is easy to get, but you have to want it

Multipath gear

- I decommissioned an EMC CX-500 and then a StorageTek STK6140 (both FC)
- sun servers generally don't need RAID because Solaris is good
- I have spare v880s with two FC loops
- we've bought a Dell MD3000i and MD3200i
- lots of JBODs with redundant SAS backplanes
- accepted old FC kit and using bits as JBODs now

What do others do?

- "serious" operating systems have proper stacks
- wikipedia has a list
- however, the ones I'm going to talk about
 - GEOM_MULTIPATH and GEOM_FOX in FreeBSD
 - dm-mpio in Linux (and NetBSD?)
 - mpxio and scsi_vhci in Solaris

GEOM Multipathing

- works at the block level - no SCSI knowledge
- the same LU will appear multiple times out of the SCSI layer as separate GEOM instances
- GEOM will recognise this and consume all the paths, and will make a single producer appear
- Only one path is used until it goes away or fails IO

GEOM Multipathing

- I can't see a significant difference between MULTIPATH and FOX from the documentation
- GEOM requires a label on disk to recognise multiple paths, doesn't use hardware/LU IDs
- the hardware can tell you this stuff, requiring a human to press buttons makes it error prone and harder than it should be to use
- some hardware will accept I/O on secondary controllers without errors, but go slower

dm-mpio

- works at the block layer again
- but issues SCSI commands to paths (via ioctl)
 - has vendor specific code to make decisions
 - checkers ask if the path can accept I/O
 - prioritizers ask how good a path is
- will use all available paths for I/O and do failover and failback etc

GEOM and dm

- working at the block layer means only block devices get multipathing
 - enclosures, tapes, libraries, etc miss out
 - relies on I/O failure or path removal for failover
 - I/O failure can mean more than a bad path
- however, doesn't require more from a stack
 - if block I/O, hotplug, and failure handling are there, then GEOM and dm-mpio should be fine

mpxio and scsi_vhci

- Multiplexed I/O and SCSI Virtual Host Controller Interface
- multipathing done at the SCSI layer, not block
- scsi_vhci acts like a normal SCSI adapter driver in the Solaris kernel
- SCSI requests against scsi_vhci end up in mpxio which routes them to paths on real adapters

mpxio

- real adapter drivers have to be modified to support mpxio
- mpxio requires alternate hotplug and command handling to normal SCSI stack
- new devices get presented to mpxio first which can decline taking them
- alternate SCSI command paths

mpxio

- mpxio builds groups of paths and handles scheduling of I/O within those groups
- vendor specific handling is implemented in modules which a third party can supply
- the modules handle claiming of paths, testing of paths, interpretation of sense data, path prioritisation, and activation of paths

Multipath in OpenBSD

- so I had equipment and interest
 - but not a lot of time :(
- I started in 2008 and "finished" it a month ago
- but effectively rewrote most of the SCSI midlayer along the way and touched most SCSI adapters
- mostly inspired by mpxio with help from dm-mpio

SCSI Device IDs - devids

- modern SCSI devices provide identifiers via VPD page 83 (or we fake them with serials or bus IDs)
- made up of a type and a buffer, eg
 - naa.500000e010902de0
 - t10.ATA_ST3320620AS_5QF075CF
 - serial.0781556b420F0AC34077
- a LU will present the same devid no matter where or how many times it gets attached

mpath(4)

- a virtual SCSI adapter (like `scsi_vhci`)
 - presents a single SCSI device to the kernel
 - sends commands down paths
- an API for paths to register with (along with a bunch of callbacks)
 - path registration is voluntary (ie, `mpath` doesn't go and look for paths itself)
 - relies on devids to identify and collect paths

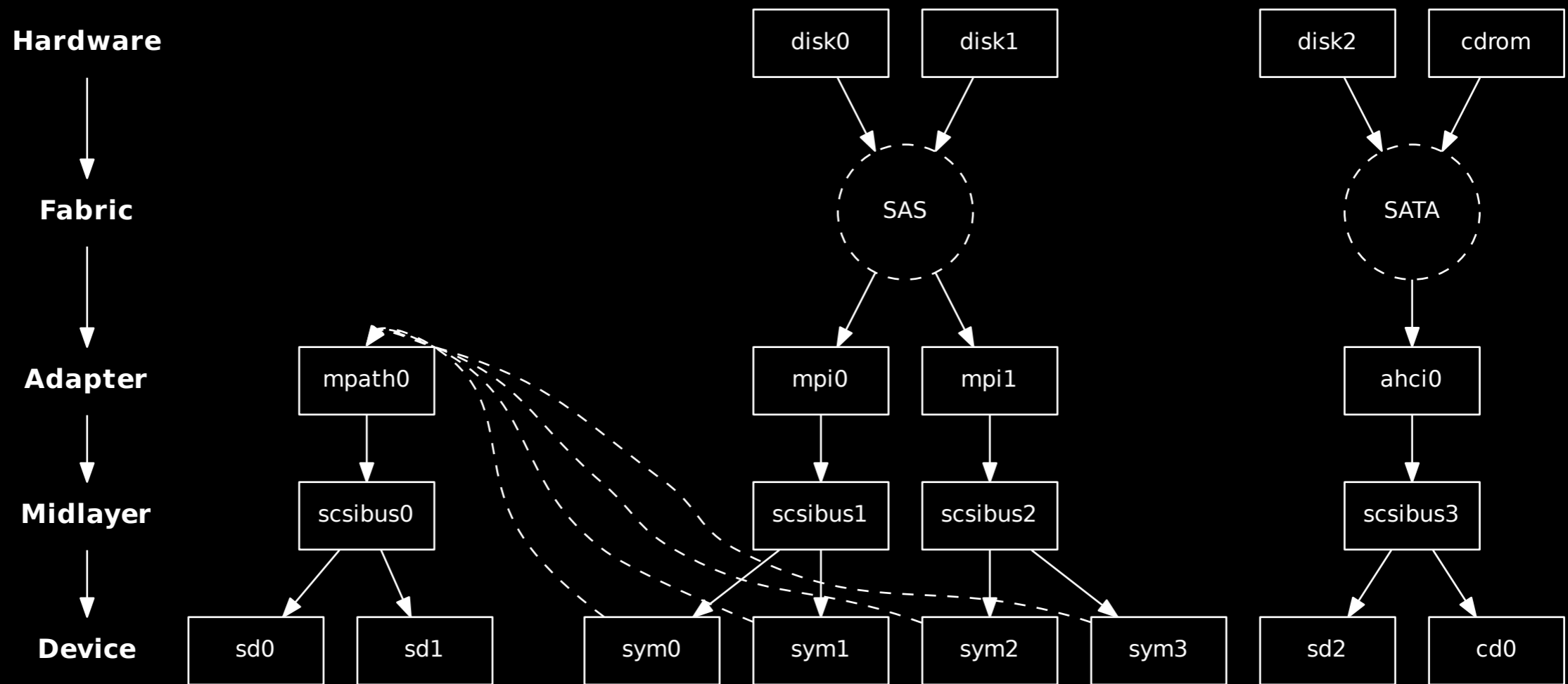
mpath(4)

- I did the original mpath at f2k9
- had the midlayer intercept paths and give them to mpath before normal autoconf
 - easy support for symmetric access devices
 - had to special case a bunch of stuff in attach/detach paths, but it worked and was reliable
 - started on LSI/Engenio support then realised I was rewriting autoconf inside scsibus(4)

mpath path drivers

- vendor/device specific knowledge is now implemented as a traditional device driver
- eg `sym(4)`, `rdac(4)`, `hds(4)`, `emc(4)`
- uses `autoconf foo_match()` to claim a path
- path drivers implement the callbacks `mpath` needs to work with too
- `foo_attach()` registers the device as a path with `mpath_attach_path()`

mpath stack



mpath vs the midlayer

- at f2k9 I discovered that the synchronous behaviour of the API the SCSI midlayer provided was really annoying, almost soul destroying
- mpath(4) acts as a proxy
 - requests to it are done by issuing new requests over paths with lots of copied values
 - to get async or concurrent commands I would have had to reuse or fake a struct buf

Old SCSI API

- the midlayer tried to be helpful by calling `biodone()` on behalf of `sd(4)`, `cd(4)`, etc
 - so couldn't reuse the buf cos it would get completed twice
- async completions only got block I/O state
 - so couldn't copy SCSI state from path xfer into the mpath xfer to provide total transparency

Old SCSI API

- async path completion would have caused mpath I/O to restart from interrupt context, and we have to try and do it because of the contract between layers
 - we would end up doing synchronous and polled SCSI xfers for multipath targets
- so I rewrote the guts of the midlayer
 - krw@ has talked this a couple of years ago

New SCSI API

- totally event driven (except for autoconf)
 - even resource allocation is via callbacks
 - command submission from any context
 - completion via callbacks with complete `scsi_xfer` state intact until explicitly freed
- no midlayer knowledge of block I/O anymore
 - all contained in `sd(4)`, `cd(4)`, `st(4)` now

New SCSI API

- improved command scheduling
 - round-robin access to contended resources
- simplified midlayer and adapter interactions
- introduced a lot of fine grained locking
- provided compat via wrappers, but that's all gone
 - new sync API on top of the callback ones

mpath(4) and new SCSI

- mpath(4) I/O path got rewritten against a midlayer that was rewritten largely cos mpath(4) needed it
- now supports asynchronous and concurrent command submission and can intercept sense data for failure detection
- has had solid support for symmetric and asymmetric devices in sym(4) for a year or three

"finishing" mpath(4)

- this talk got accepted so I had to finish mpath(4)
- it lacked support for controller failover
 - it only attached paths on active controllers
 - if LUN ownership in an array changed mpath(4) wouldn't notice and I/O would fail or stop
- only supported paths on a device

"finishing" mpath(4)

- implemented groups between paths and devices
 - groups identified by path driver at attach time usually by querying which controller they're on
 - groups are either active or inactive
 - all paths in the active group get I/O
 - round-robin in the active group
- asym devices add each path to different groups so mpath(4) doesn't have to do MRU anymore

"finishing" mpath(4)

- implemented detection of LUN ownership change by fleshing out path drivers sense callback
 - intercepted on the way back from normal I/O
- groups are queried for their active state until one is found, then I/O resumes as normal
- querying is implemented as a state machine
 - originally as a task in a thread but OMG PAIN
 - event based with callbacks now

mpath(4) in practice

- multipathing exists to increase availability of SCSI devices and hide transient failures or topology changes from things using them
- ie, mpath(4) is there to tell lies
 - eg, don't panic if you unplug the disk / is on
- increased bandwidth is nice, but meh
- to lie well it needs to trust what it sits on top of

mpath(4) in practice

- you need adapter drivers that handle failures and timeouts, and do hotplug really well
 - that means mpi(4), mpfi(4), and vscsi(4)
 - isp(4) resists us at the moment
- I've been extremely conservative about which devices path drivers claim
 - might make it conditional adapter driver too

mpath(4) in practice

- you need very intimate detail about your devices
 - sym(4) is free
 - dm-mpio is the best reference we have :(
 - rdac(4) is working great cos I have it plugged in
 - I will update hds(4) and emc(4)
 - path drivers are easy if you can get the hardware knowledge, so please help

mpath(4) in reality

- mpath(4) is not enabled in GENERIC yet
- very solid on top of mpi(4), mpfi(4),
vscsi(4)+iscsid(8)
 - but iscsid(8) is not enabled or finished really
- sym(4) support likely to be the most popular
 - the hardware is cheap
 - but people will expect a lot from software RAID

mpath(4) in the future

- enable mpath(4) and sym(4) at least
- logical-block I/O scheduling is necessary
 - difference between 10MB/s and 120MB/s writes
 - may improve path utilisation too
- add path drivers as hardware and time permits
- mpath ahci(4)?
- mpath on clustered RAID?

Questions?