Testing the Transport Side of Highly Available Hosts

STUART KENDRICK

Stuart Kendrick works as a third-tier tech at the Fred Hutchinson Cancer Research Center in Seattle, where he dabbles in trouble-shooting, deep infrastructure design, and developing tools to monitor and manage devices. He started earning money as a geek in 1984, writing in FORTRAN on CRAY-1s for Science Applications International Corporation; worked in desktop support, server support, and network support at Cornell University; and reached FHCRC in 1993. He has a BA in English, contributes to BRIITE (http://www.briite.org), and spends free time on yoga and CrossFit.
skendric@fhcrc.org

When I configure Highly Available (HA) systems, I intend to configure them such that if either half goes down, the service remains accessible to the user. Too often, however, I find that I have configured them to be Highly Unavailable (HU), meaning that if either half goes down, the service becomes inaccessible.

Highly Available hosts typically sport redundant Ethernet NICs. Originally, I would test these by unplugging the cable feeding each NIC, perhaps while sending a continuous ping to the host's IP address to measure how well it handled the event. As it turns out, this test covers a limited selection of possible failure scenarios, not enough to determine whether the host has been configured HA or HU. In this article I describe a test protocol which more accurately assesses the host's configuration.

HA Transport in Datacenters

There are many ways to design a datacenter [1] to deliver Highly Available Ethernet/IP [2]. While the details vary widely—and have repercussions for host configuration and the parameters described here—they all share the same concepts: elements probe one another periodically to verify that the current network path is viable and, if it isn't, switch to an alternate path. For the purposes of discussion, Figure 1 illustrates one such design. Contrast it with what I will call a Single Point of Failure (SPOF) design, in which Switch B and Router B do not exist.

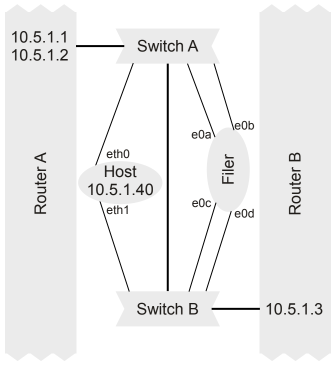


Figure 1: HA Ethernet/IP datacenter design

The Host is equipped with two NICs, one plugged into Switch A, the other into Switch B. Filer is a NetApp storage controller; we'll talk about it later. The two Switches are interconnected and also provide paths to the upstream routers (likely marketed as Layer 3 switches by their vendors, but for the purposes of this discussion, I'll use the term *router* to indicate a Layer 3 device). The Host is configured to rely on 10.5.1.1 as its default gateway. 10.5.1.1 is configured as a virtual IP address, currently owned by Router A. If Router A goes down, then Router B will acquire 10.5.1.1, thus hiding the loss of the default gateway from Host. We accomplish this magic via the use of a *gateway redundancy protocol*, e.g., VRRP, CARP, HSRP, or GLBP. Here is an example of HSRP in action on the wire; each router emits these Hellos once per second.

```
Delta T   Src         Dst        Protocol
0.55430   10.5.1.2 -> 224.0.0.2  HSRP Hello (state Active)
0.07153   10.5.1.3 -> 224.0.0.2  HSRP Hello (state Standby)
```

If the standby router (Router B) does not hear from the active router (Router A) for a configured amount of time (three seconds in our environment), then the standby router will change its state to Active, adopt both the MAC address and the IP address of the default gateway (10.5.1.1), and thus set up shop as the "go to" router on this subnet.

The Host can be configured to employ both NICs simultaneously (Active/Active) or to rely on one NIC while holding the other in reserve (Active/Standby). In both cases, though, it faces the same problem that Router A and Router B face: how does it know when its partner—in this case, one of its NICs—is defunct?

Sources of Failure

By default, the average host out of the box pays attention to the Ethernet link signal transmitted by the switch in order to determine whether it should consider a NIC viable. Regrettably, a range of hardware failures and human fat-fingering can result in the switch continuing to transmit link but no longer forwarding frames. For example, when the Supervisor/switching engine (the brains card) in a switch fries, the individual ports continue to transmit link, but the brains no longer forward frames arriving from hosts. Rebooting a switch, perhaps to load a new operating system, results in similar behavior during the boot process—as the switch reboots, it performs hardware tests on its cards, toggling link up and down several times, all the while tossing incoming frames into the bit bucket. Similarly, a line card can lose its connection to the switch's backplane, or a human can fat-finger a VLAN assignment, isolating a host from its intended conversation partners. *The lights are on, but no one is home.*

Figure 2 enumerates the failure modes we have experienced at our institution. Notice how, from a purely component-based point of view, Highly Available environments will tend to experience twice as many component failures as their non-redundant equivalents, because they contain twice as many parts. When compared to a SPOF design, HA environments also mean that:

- Operating staff spend twice as much time replacing fried components.
- When systems are configured in a Highly Unavailable way, there's twice as much downtime.

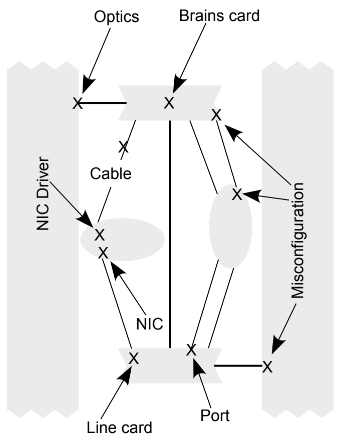


Figure 2: Sources of failure

Ethernet and IP both being *ship and pray* protocols [3], the transport infrastructure cannot recover from such errors: the burden for detecting and responding to such issues lies with the host. Hosts which rely strictly on link for determining the validity of a NIC will transmit frames into the bit bucket ad infinitum during one of these failure scenarios.

Polling

Host operating system designers and NIC driver developers commonly provide mechanisms for Ethernet NICs to exchange keep-alives with each other, with the upstream default router, or with the brains card on the nearby Ethernet switch. In this fashion, the host OS determines whether or not a given NIC has a viable path to the rest of the world (network) and then decides whether to continue forwarding frames across that NIC. However, operating systems do not ship with these features enabled; we system administrators must identify which configuration fits our environment and turn it on.

Linux

The Linux folks wrap their Ethernet HA options into their *bonding* driver [4]. In this example, I poll the IP addresses of the local routers to determine the viability of a NIC and of its path through the local network.

```
Host> cat /etc/sysconfig/network-scripts/ifcfg-bond0
DEVICE=bond0
IPADDR=10.5.1.40
NETMASK=255.255.255.0
NETWORK=10.5.1.0
BROADCAST=10.5.1.255
GATEWAY=10.5.1.1
[…]
BONDING_OPTS='mode=active-backup arp_interval=1000 arp_validate=all \
arp_ip_target=10.5.1.2 arp_ip_target=10.5.1.3 primary=eth0'

Host> cat /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE=eth0
```

```
[…]
MASTER=bond0
SLAVE=yes

Host> cat /etc/sysconfig/network-scripts/ifcfg-eth1
DEVICE=eth0
[…]
MASTER=bond0
SLAVE=yes
```

Be aware that the `ifcfg-xxxx` format was different in older versions of the bonding driver; the format illustrated here became accurate with v3.2.0 or thereabouts. Type `cat /sys/module/bonding/version` to view your version and see bonding.txt for syntax variants appropriate for your version.

Restart networking via `/etc/init.d/network restart` or similar and verify that the bonding driver correctly reports key parameters; see bolded text below.

```
Host> cat /proc/net/bonding/bond0
Ethernet Channel Bonding Driver: v3.2.4 (January 28, 2008)

Bonding Mode: fault-tolerance (active-backup)
Primary Slave: eth0
Currently Active Slave: eth0
MII Status: up
MII Polling Interval (ms): 0
Up Delay (ms): 0
Down Delay (ms): 0
ARP Polling Interval (ms): 1000
ARP IP target/s (n.n.n.n form): 10.5.1.2, 10.5.1.3

Slave Interface: eth0
MII Status: up
Link Failure Count: 2036
Permanent HW addr: 00:19:b9:32:76:25

Slave Interface: eth1
MII Status: up
Link Failure Count: 1
Permanent HW addr: 00:19:b9:32:76:27
```

On the wire, the result is the following, exchanged every 1000 ms:

```
Delta T   Src                 Dst             Protocol
0.50123   Dell_32:76:25  ->   Broadcast       ARP Who has 10.5.1.2?  Tell 10.5.1.40
0.50124   Dell_32:76:25  ->   Broadcast       ARP Who has 10.5.1.3?  Tell 10.5.1.40
0.50158   Cisco_64:3a:de ->   Dell_32:76:25   ARP 10.5.1.2 is at 00:03:6c:64:3a:de
0.50162   Cisco_43:bc:00 ->   Dell_32:76:25   ARP 10.5.1.3 is at 00:03:6c:43:bc:00
```

In this *active-backup* configuration, the *active slave* emits ARP Requests every 1000 ms. If it doesn't hear an ARP Reply from either 10.5.1.2 or 10.5.1.3 within 2000 ms (2 * arp_interval: see drivers/net/bonding/bond_main.c in the Linux kernel source), then the driver marks the *active slave* (eth0 in this case) as down and initiates a failover to the *backup slave* (eth1) [5].

In this design, failure to hear from either 10.5.1.2 or 10.5.1.3 covers all the failure scenarios illustrated above.

Windows

Under Windows, NIC manufacturers provide driver features which implement similar polling. Intel calls their approach *Teaming* [6]; the admin uses a GUI [7] to configure the parameters [8]. Let's talk about the choices.

I prefer *adapter fault tolerance* (e.g., Active/Standby) over active load-balancing (e.g., Active/Active), because I find packet capture easier. With Active/Active configurations, I need two sniffers, plus the headache of merging the two trace files together (rarely a precise process)—that is a lot of overhead, particularly when I am in a hurry to fix something that is broken. Furthermore, if the host actually needs to employ both NICs in order to deliver sufficient service (needs to transmit and/or receive across both NICs), then it is no longer Highly Available—the loss of either NIC will lead to service degradation.

I set *Activation Delay* to 100, which instructs the driver to leave a NIC disabled for 100 seconds after it has determined that the NIC is ready to return to production, this because cable and switch failures can be erratic, working for a few seconds, failing for a few seconds. By instructing the driver to wait a while before re-enabling a previously disabled NIC, I harden the host against this sort of flapping experience.

And of course I enable *Probes*.

Here is what Intel Teaming looks like on the wire, with the Active and Standby NICs each sending probes to one another.

```
Delta T    Src                Dst         Protocol
0.51795    Intel_e4:ea:72 ->Multicast     Intel ANS probe Sequence: 3098765056,
                                          Sender ID 256
                                          Team ID 00:11:43:e4:ea:72
0.51796    Intel_e4:ea:73 ->Multicast     Intel ANS probe Sequence: 3098765056,
                                          Sender ID 512
                                          Team ID 00:11:43:e4:ea:72
```

Broadcom calls their approach LiveLink, which uses the same ARP polling approach that Linux *bonding* uses, although LiveLink requires that each NIC have its own IP address, in addition to the shared virtual address. For details, poke around Dell's site [9] or consult yours truly [10]. I recommend updating to the latest drivers in order to dodge a series of nasty bugs—we're using v4.1.4.0 successfully.

NetApp

The ONTAP designers chose to layer their Ethernet High Availability scheme on top of IEEE 802.3ad, aka Link Aggregation Channel Protocol (LACP). Linux bonding, Intel Teaming, and Broadcom LiveLink all support this approach as well; ONTAP requires it. LACP was intended as a protocol to permit bundling multiple Ethernet links into a single pipe or channel in order to increase the throughput available between two switches or between a host and a switch. However, as the ONTAP designers realized, LACP ships with a built-in polling protocol—the host and the switch exchange periodic Hellos to ensure that their understanding of the channel specifications are synced.

```
Delta T  Src                 Dst            Protocol
1.05400  NetApp_00:45:44 -> Slow-Protocols  LACP Actor Port = 1 Partner Port = 368
1.07000  Cisco_06:b4:7f  ->  Slow-Protocols  LACP Actor Port = 368 Partner Port = 1
```

The brains cards in Ethernet switches are responsible for emitting these Hellos. Thus, the host configured for LACP can determine whether or not anyone is home in the switch by listening for silence.

In ONTAP-speak, I create dynamic, multimode Virtual Interfaces (VIFs) [11] using the LACP protocol and then combine pairs into single-mode VIFs, where e0a and e0c are plugged into Switch A and e0b and e0d are plugged into Switch B, per Figure 1 [12].

```
Filer> rdfile /etc/rc
hostname Filer
vif create lacp dmmvif1 -b ip e0a
vif create lacp dmmvif2 -b ip e0b
vif create lacp dmmvif3 -b ip e0c
vif create lacp dmmvif4 -b ip e0d
vif create single svif1 dmmvif1 dmmvif2
vif create single svif2 dmmvif3 dmmvif4
```

Myself, I don't like entangling host and switch configurations—adds an additional dependency and yet another way for the switch admin to break the host. Furthermore, while the host can detect both cable failure and switch failure using this scheme, it cannot detect either a switch admin fat-fingering a VLAN assignment or the loss of the path between Switch A and Router A. Finally, LACP misconfigurations and bugs are hard to troubleshoot, because capturing the LACP traffic requires an in-line sniffer (LACP is a link-local protocol, invisible to Wireshark running on the host or port mirroring on the switch).

On the other hand, the LACP approach dodges the polling and configuration subtleties inherent in the competing techniques, relying as it does on the protocol's built-in Hello function. And for the ONTAP developers and testers, it eliminates an entire chunk of functionality (an ARP-based polling mechanism, for example) which they would otherwise have to implement and maintain—all steps in the right direction, as far as uptime goes. In the end, we like our NetApps for a range of reasons, and this is the only NIC HA approach ONTAP supports, so we do it.

Application-Layer Protocols

As an aside, if our application-layer protocol contains its own polling techniques, then we can dispense with all these kernel-level and driver-level shenanigans. For example, SCSI Initiators and Targets exchange frames, if only NOPs, every five seconds. When configured with *multipathing*, SCSI running over IP (iSCSI) and SCSI running over Fibre Channel (FC) have no need for these lower-layer fault detection protocols—SCSI itself detects the failure of a path and initiates failover to a backup path—in our experience, a robust technique.

Here we see the iSCSI Initiator (10.5.1.50) emitting NOPs to its two iSCSI Targets (10.5.1.61 and 10.5.1.62).

```
Delta T    Src          Dst          Protocol
0.04404    10.5.1.50 -> 10.5.1.61    iSCSI NOP Out
0.04450    10.5.1.61 -> 10.5.1.50    iSCSI NOP In
0.05009    10.5.1.50 -> 10.5.1.62    iSCSI NOP Out
0.05043    10.5.1.62 -> 10.5.1.50    iSCSI NOP In
5.04423    10.5.1.50 -> 10.5.1.61    iSCSI NOP Out
5.04451    10.5.1.61 -> 10.5.1.50    iSCSI NOP In
5.05118    10.5.1.50 -> 10.5.1.62    iSCSI NOP Out
5.05230    10.5.1.62 -> 10.5.1.50    iSCSI NOP In
```

Test Procedure

So how do we verify that all this stuff actually works? I have experimented with pulling cables, assigning ports to the wrong VLANs, and even inserting mini-switches between host and switch (in order to sustain link but still produce a bit bucket—I yank the cable marked with an ‘X’ in Figure 3).

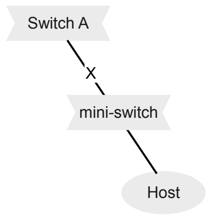


Figure 3: Manual testing

Each of these tests provides varying levels of validation, with the mini-switch test being particularly effective. However, the test I prize above all is rebooting each of the switches and routers in turn, because a reboot exercises a whole range of failure scenarios, from loss/restoration of link to the more brutal *the lights are on, but no one is home* condition, incurred while the switch ports are transmitting the link signal but the brains card is still performing hardware checks.

I like to measure the behavior of hosts using mass-ping [13], possibly because it is a brilliant and precisely honed tool for tracking IP connectivity or possibly just because I wrote it. mass-ping emits a stream of ICMP Echos, one per second, to a list of IP addresses or to entire subnet ranges, giving the operator real-time feed-back on behavior and saving the test result to a CSV file, which a supporting tool can convert into a graphic.

```
Host> sudo mass-ping -s yes -c "Reboot Switch A" -n switch-a -w 900 -q 10.5.1.0/24
[sudo] password for skendric:
Sanity check...
Identifying live hosts...

Beginning with 144 live addresses
Pinging targets every 1 seconds with timeout 0.2 seconds, running for 15 minutes,
hit Ctrl-C to cancel...
144 144 144 144 144 144 144 144 144 144 144 144 144 144 144 144 144 144 144 144
3   3   3   130   130 130 130 130 130 130 130 130 130 130 130 130 130 130 130 130
130 130 130 […] 144 144 144 144 144 144 144 144 144 144 144 144 144 144 144 144
144 144 144 144 144 144 144 144 144 144 144 144
```

The report at the end summarizes results:

```
# target        hits    misses
# -------------- -----   ------
aurora          897      3
jennite         897      3
madison         417      483
osiris          415      485
[…]
```

The graph-mass-ping script takes the CSV data file as input and produces a graphic illustrating when each host missed pings. The exclamation point represents ICMP Replies while the periods represent silence.

```
Title           Mass-Ping:  Reboot Switch A
Invocation      mass-ping -i 1 -w 900 -t 0.2 -q 10.5.1.0/24
Details         Run from Host on 2010-01-30 at 05:49 by skendric
Errors
Node count      144
Time count      900

Nodes                             Time
            05:49:05  05:49:15  05:49:20  05:49:25  05:49:30
aurora      !!!!!!!!!!!!!!!!!!!!...!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
jennite     !!!!!!!!!!!!!!!!!!!!...!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
madison     !!!!!!!!!!!!!!!!!!!!.............................................
osiris      !!!!!!!!!!!!!!!!!!!!.............................................
[…]
```

In this example, the datacenter contained 144 active IP addresses, 130 of which didn't blink when Switch A went down. Of those that noticed, aurora and jennite each needed three seconds to flip to their standby NICs, while madison and osiris stumbled and didn't recover.

Ideally, I run multiple simultaneous mass-ping invocations, from hosts located within the datacenter (one on each VLAN) and from at least one host located outside the datacenter, while performing each test. Large-format color printers (tabloid sized paper), tape, and a lot of wall space allows me to view mass-ping output for an entire datacenter. Periods during which all hosts miss pings suggest a systemic issue—unplugged switch-to-router uplink cable or misconfigured routing protocol. Periods during which many hosts miss pings but many do not suggest a misconfigured VLAN. And periods during which a few hosts miss pings but most do not suggest host-specific issues. In our environment, the most common causes of systemic failure have been fried optics and misconfigured router interfaces, while the most common causes of host failure have been bad NICs, unplugged cables, and buggy NIC drivers.

Some hosts throttle the rate at which they will respond to pings—get enough mass-ping sessions going and you can bump into that rate-limiter. For example, under ONTAP, check your settings for option `ip.ping_throttle.drop_level`.

Rubber Hits the Road

We started implementing redundant Ethernet/IP transport in 1998, intermittently testing it using manual techniques. In 2004 I automated the test process, writing code [14] which steps through the 15 or so redundant pairs of routers and switches at our institution, rebooting each one in turn, and watching, via pings, hosts on the other side. It runs via cron once per month. When it detects trouble, it halts and pages me. Each year, this automated process uncovers a handful of flaws in the switched/routed infrastructure plus numerous host-specific issues [15].

The most spectacular issue we have uncovered to date revolved around our NetApps. We had been skipping the automated testing of the larger datacenters, based on conflicts with projects and general anxiety. In fact, the previous test had occurred just prior to turning up our new centralized VMware cluster—seven Sun 4450s mounting a clustered FAS3020 back-end via NFS. Turns out that we had misinterpreted NetApp documentation, believing that ONTAP implemented an ARP-based probing scheme, similar to Linux bonding, and that this came for free, without specific configuration on our part. *Note to self: There is no free lunch.* We rebooted the first Ethernet switch to load the new OS, and VMs started crashing. More precisely, Windows crashes when it cannot write to disk; Linux enters read-only mode.

```
Apr 24 05:45:02 cairo kernel: Remounting filesystem read-only
Apr 24 05:45:02 cairo syslog-ng[2312]: io.c: do_write: write() failed (errno 30),
Read-only file system
```

In theory, detecting the frozen Windows guests was easy—our network management station reported them down. In practice, we had been forgetting to add VMs to the management station's lists. Worse, many of the Linux guests continued to respond to management station polls just fine . . . but, of course, at some point, being unable to write to disk impacted their behavior. *Note to self: Upgrade monitoring strategy to include polls which incur disk writes.* Finally, the VMware hosts themselves lost touch with the VMware console after this event, making them unmanageable via the GUI and requiring reboots of each host to restore this function. Crawling through the lists of VMs and rebooting the ones which were hung or read-only took hours, followed by a multi-day process of migrating guests off a host, rebooting the host, and migrating guests back onto it.

After that, we started learning about dynamic multimode VIFs and LACP. During our most recent test of a datacenter, the NetApps and VMware cluster rode through without blinking—we rebooted switches repeatedly, and they didn't break a sweat (dropping only the occasional ping). At this point, our filers literally don't care about the loss of an Ethernet switch.

Subtleties

Even without the fancy LACP configuration, most of our NetApps have ridden through the loss of a switch without a problem, because our switches tend to toggle link, if only briefly, when they reboot. So when Switch A reboots, for example, it drops link on all ports for a few seconds. ONTAP notices the loss of link and flips to a backup NIC attached to Switch B. The first switch does its stuff, returns to life, and begins to service traffic—though the Filer ignores it, happily using its "b" side NIC.

In the NetApps backing our VMware cluster, however, we had started to use the *favor* command, which instructs ONTAP to fail back to the primary NIC, once link is restored. That, of course, got us into trouble—line cards in a rebooting switch will transmit the link signal early in the reboot process, long before the brains card will forward frames. *The lights are on, but no one is home.* As a result, ONTAP would re-enable NICs attached to the rebooting switch too soon, sending traffic into oblivion. In my experience, this example illustrates several themes in the behavior of HA designs under stress:

- In general, HA systems handle losing a component more gracefully than returning a component to service.
- Failed components will sometimes oscillate on their way to returning to life or on their way to a permanent death.

Configuring HA systems to dawdle before returning a previously failed component to service helps protect against these issues.

Conclusion

Like everything else in this business, developing these techniques has taken me years of trial and error, and I expect to advance them further as I continue to better understand how to migrate transport infrastructure from Highly Unavailable to Highly Available configurations.

Note that I have focused on one particular Ethernet/IP datacenter design; the configuration choices I sketch here apply to that particular design. Different transport designs call for different configuration choices in bonding, Teaming, LiveLink, and LACP.

In the future, I hope to expand my validation toolkit to include read/write tests across storage devices and pings across Fibre Channel networks, to tackle the challenge of verifying the failure behavior of application-layer clusters, and to develop tools that proactively identify flaws in configurations.

References

[1] Cisco Datacenter Infrastructure Design Guide: http://www.cisco.com/application/pdf/en/us/guest/netsol/ns107/c649/ccmigration_09186a008073377d.pdf.

[2] Trey Layton, Virtual Port Channels, Cross-Stack EtherChannels, MultiChassis EtherChannels: http://communities.netapp.com/blogs/ethernetstorageguy/2009/09/23/virtual-port-channels-vpc-cross-stack-etherchannels-multichassis-etherchannels-mec--what-does-it-all-mean-and-can-my-netapp-controller-use-them.

[3] Howard Goldstein, Storage Network Design, Performance, and Troubleshooting: http://www.hgai.com/index_files/Page488.htm.

[4] Linux Ethernet Bonding Driver HOWTO: http://www.kernel.org/doc/Documentation/networking/bonding.txt.

[5] Discussion of subtleties on the Bonding Development list: http://sourceforge.net/mailarchive/forum.php?thread_name=AANLkTimhWimxhQZ__i-eNU%3DYFJaXgQU_5J%3DLDUmO%3DFY0%40mail.gmail.com&forum_name=bonding-devel.

[6] Intel Advanced Network Services Software Increases Network Reliability, Resilience, and Bandwidth: http://www.intel.com/network/connectivity/resources/doc_library/white_papers/254031.pdf.

[7] How to Configure Teaming Modes on Intel Network Adapters: http://www.youtube.com/watch?v=GQ4WGGdlqpc.

[8] Network Connectivity Advanced Networking Services: http://www.intel.com/support/network/sb/cs-009744.htm.

[9] Configuring LiveLink for a Smart Load Balancing and Failover Team: http://support.dell.com/support/edocs/network/P29352/English/bacs.htm#configuring_livelink.

[10] Stuart Kendrick, Configure HA Servers in Datacenters: http://www.skendric.com/philosophy/Configure-HA-Servers-in-Data-Centers.pdf.

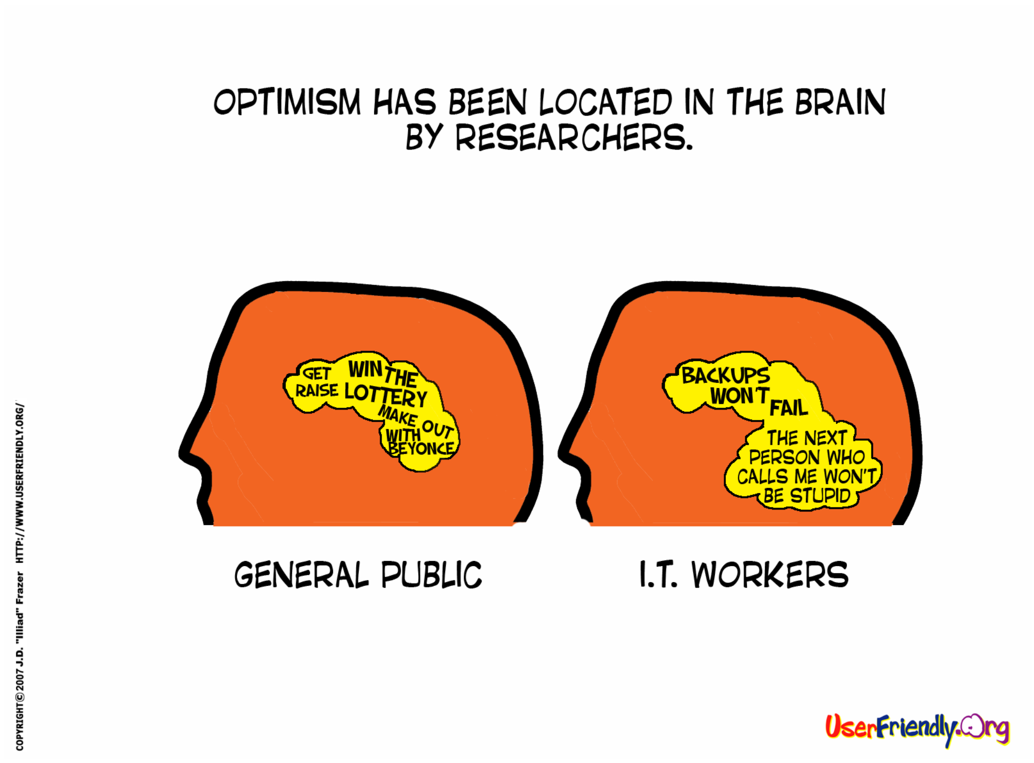
[11] Trey Layton, Multimode VIF Survival Guide: http://communities.netapp.com/blogs/ethernetstorageguy/2009/04/04/multimode-vif-survival-guide.

[12] Stuart Kendrick, Focus on NetApp: http://www.skendric.com/philosophy/Toast-Ethernet-IP.pdf.

[13] Mass-Ping: http://www.skendric.com/nmgmt/polling/mass-ping/.

[14] Red-Reboot: http://www.skendric.com/nmgmt/device/Cisco/red-reboot.

[15] Stuart Kendrick, "A Few Thoughts on Uptime," pp. 64-65: http://www.skendric.com/philosophy/A-Few-Thoughts-on-Uptime.pdf.


OPTIMISM HAS BEEN LOCATED IN THE BRAIN BY RESEARCHERS.
GET RAISE
WIN THE LOTTERY
MAKE OUT WITH BEYONCE
BACKUPS WON'T FAIL
THE NEXT PERSON WHO CALLS ME WON'T BE STUPID
GENERAL PUBLIC
I.T. WORKERS
COPYRIGHT© 2007 J.D. "Illiad" Frazer HTTP://WWW.USERFRIENDLY.ORG/
UserFriendly.Org