

trunk(4) - link aggregation and link failover interface

I found an article about [OpenBSD 3.8: Hackers of the lost RAID](#) which had some fun information about a trunk interface in OpenBSD that sounded like a nice thing.

This page is about some experiments done using this interface on a small firewall based on a Portwell PPAP-100 which is a small CPU with 6 fxp network interfaces.

Equipment used:

- Portwell firewall named Azumi
- IBM Thinkpad model X31 named Timon
- Greycom GC-1424F 24-port switch, used for other things during experiments!
- Cat 5E 1 meter cables

Software used:

- OpenBSD 3.9-current - see [dmesg](#)
- NetStrain 3.0 (c) 2002 Christoph Pfisterer

Method used - not very scientific, but fun nonetheless :-)

1. connect two systems with 1 meter cables to switch - measure maximum throughput one interface
2. connect another ethernet cable to switch and reconfigure firewall to use trunk interface for failover
3. redo experiments, mess with cables

Measurements done

All measurements are done using Netstrain which is a nice and simple tool -

Default Publication

some may find it too simple.

When you use netstrain you start a netstrai nd daemon at one end and then run the netstrain client at the other. The client is then run in either send, recv or both mode.

Sample run from Azumi with netstrai nd started at Timon on IP address 192.168.18.133:

```
root@azumi:hllk# netstrain timon 1234 send
NetStrain 3.0 (c) 2002 Christoph Pfisterer <cp@chrisp.de>
Looking up hostname timon...
Connecting to 192.168.18.133 port 1234 using IPv4...
Connected
sent: 1069M, 11419.7K/s total, 11414.2K/s current
recv'd: 0B, 0B/s total, 0B/s current
^C
```

Since this method is very dynamic - you can see the numbers being updated. It seemed the number were about 11400K/s for both send and recv. When specifying both - resulting in sending and receiving data at the same time, the number were like this:

```
root@azumi:hllk# netstrain timon 1234 both
NetStrain 3.0 (c) 2002 Christoph Pfisterer <cp@chrisp.de>
Looking up hostname timon...
Connecting to 192.168.18.133 port 1234 using IPv4...
Connected
sent: 485M, 8939.0K/s total, 8931.5K/s current
recv'd: 505M, 9303.7K/s total, 9292.4K/s current
^C
```

This is fine - if connection is lost I see the result immediately and the two machines are working well.

Configure trunk

Since I had this network with DHCP I decided to just use another interface and try getting DHCP address on the trunk interface.

```
root@azumi:# cat hostname.fxp0
up
root@azumi:# cat hostname.fxp1
up
root@azumi:# cat /etc/hostname.trunk0
trunkproto failover trunkport fxp0 trunkport fxp1
dhcp
```

During boot it gets configured and console has this information:

```
starting network
DHCPREQUEST on trunk0 to 255.255.255.255 port 67
DHCPCACK from 192.168.18.1
bound to 192.168.18.153 -- renewal in 30000 seconds.
```

So it seems the trunk is configured and it even gets an IP-address :-)
Ifconfig reveals the two fxp interfaces being in UP state and the trunk0 configured as expected:

Default Publication

```
fxp0: flags=8943 mtu 1500
lladdr 00:90:fb:a1:3d:c6
trunk: trunkdev trunk0
media: Ethernet autoselect (100baseTX full-duplex)
status: active
inet6 fe80::290:fbff:feal:3dc6%fxp0 prefixlen 64 scopeid 0x1
# ifconfig fxp1
fxp1: flags=8943 mtu 1500
lladdr 00:90:fb:a1:3d:c6
trunk: trunkdev trunk0
media: Ethernet autoselect (100baseTX full-duplex)
status: active
inet6 fe80::290:fbff:feal:3dc5%fxp1 prefixlen 64 scopeid 0x2
# ifconfig trunk0
trunk0: flags=8843 mtu 1500
lladdr 00:90:fb:a1:3d:c6
trunk: trunkproto failover
trunkport fxp1 active
trunkport fxp0 master,active
groups: trunk egress
media: Ethernet autoselect
status: active
inet6 fe80::290:fbff:feal:3dc6%trunk0 prefixlen 64 scopeid 0xb
inet 192.168.18.153 netmask 0xfffff00 broadcast 192.168.18.255
```

I tried removing the cable from fxp0 while running netstrain and after a short while it switched over and continued transferring data. Another test using ICMP gave this information:

```
# ping timon
PING timon (192.168.18.133): 56 data bytes
64 bytes from 192.168.18.133: icmp_seq=0 ttl=255 time=0.962 ms
64 bytes from 192.168.18.133: icmp_seq=1 ttl=255 time=1.129 ms
64 bytes from 192.168.18.133: icmp_seq=2 ttl=255 time=1.265 ms
64 bytes from 192.168.18.133: icmp_seq=3 ttl=255 time=1.408 ms
64 bytes from 192.168.18.133: icmp_seq=4 ttl=255 time=1.161 ms
64 bytes from 192.168.18.133: icmp_seq=5 ttl=255 time=1.156 ms
64 bytes from 192.168.18.133: icmp_seq=8 ttl=255 time=0.329 ms
64 bytes from 192.168.18.133: icmp_seq=9 ttl=255 time=1.381 ms
64 bytes from 192.168.18.133: icmp_seq=10 ttl=255 time=1.145 ms
64 bytes from 192.168.18.133: icmp_seq=11 ttl=255 time=1.609 ms
```

So it lost a few packets around sequence number 6 and 7 - I don't know if it lost the request or the response - but anyhow it worked and it seems pretty useful.

Results

The test was fun and it actually was a bit short - everything worked as designed and knowing OpenBSD network configuration in advance it was really simple to configure.

I can imagine an important server being configured with two network cards and then connected to the infrastructure through two switches using this method.

Thank you OpenBSD - it works perfectly.