L3DSR – Overcoming Layer 2 Limitations of Direct Server Return Load Balancing

Jan Schaumann, Systems Architect

<jschauma@yahoo-inc.com>

E2A7 437A 7AB8 6EA1 7E1D

F6DC BF09 CDC9 E157 FAB8

YAHOO!

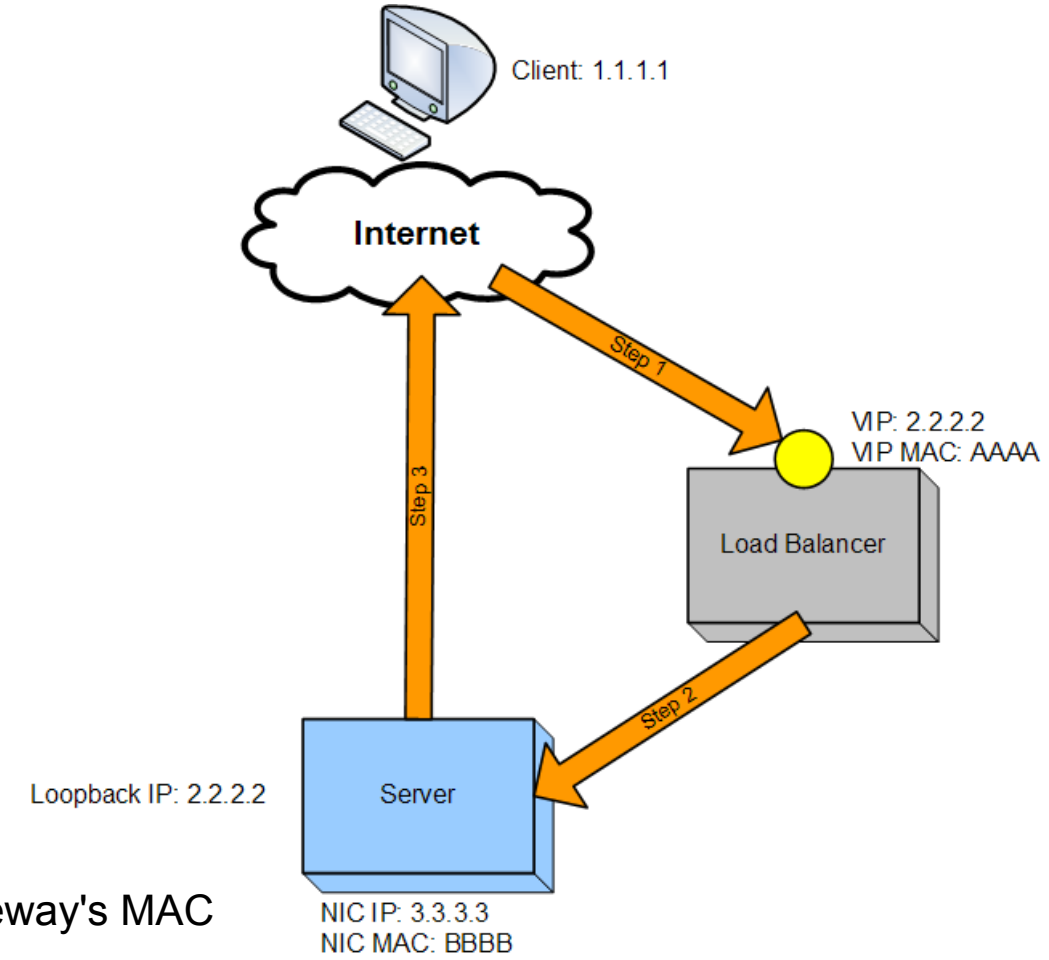
Traditional or L2 DSR

- "Direct Server Return" also known as:
  - "Direct Routing"
  - "SwitchBack" (Foundry/Brocade)
  - "nPath" (F5)

- General packet flow:
  - clients send requests to a Virtual IP (VIP) served by the LoadBalancer
  - LB determines real server to forward the request to
  - LB performs MAC Address Translation (MAT)
  - server responds *directly* to client, bypassing the LB

Traditional or L2 DSR

- Client to LB:
  - Source IP: 1.1.1.1
  - Destination IP: 2.2.2.2
  - Destination MAC: AAAA
- LB to Server:
  - Source IP: 1.1.1.1
  - Destination IP: 2.2.2.2
  - Destination MAC: BBBB
- Server to Client:
  - Source IP: 2.2.2.2
  - Destination IP: 1.1.1.1
  - Destination MAC: default gateway's MAC



Traditional or L2 DSR

- Layers are independent (L2 does not care about L3 information)

- LB and server *both* have VIP configured
- server must not answer ARP for VIP
- service must bind to VIP
  - service may also have to bind to real IP for health checking
- LB and server need to be on the same L2 network segment

Advantages of L2 DSR

- Clients' source addresses are preserved
  - Yahoo! is very interested in knowing the source address of any incoming requests
  - *SNAT loses the clients' addresses*

- Performance
  - LB only handles inbound packets
  - typical inbound traffic is much smaller than outbound traffic (about 1:8 for typical Yahoo! traffic)
  - normal DSR LBs at Y! can have hundreds of VIPs across hundreds of real servers, handling multiple Gbps of inbound traffic (tens of Gbps of outbound traffic)
  - *SNAT forces all traffic through the LB*

Limitations of L2 DSR

- server cannot directly respond to ARP requests for VIP
  - configuration of loopback aliases (or alias on interface marked as "down")
- health checking requires additional configuration
  - service on VIP on loopback alias
  - service on VIP for health check (some models)
- Port translation not possible
  - port selection is protocol dependent, so does not happen on Layer 2
- LB and all servers need to be on the same L2 network segment
  - physical location of servers behind VIP restricted
  - flexibility within datacenter limited

What's "wrong" with L2 anyway?

- instability on very large (10K+ hosts) networks
- STP designs require an active/backup design for rack uplinks – we wanted active/active for any single rack switch
- if we require multiple L2 domains (for stability), hosts in different L2 domains cannot participate in the same VIP
  - physical location of servers behind VIP restricted
  - flexibility within datacenter limited

How do we get from L2 to L3?

- Server needs to know
  - client source address
  - VIP address for which the request was made

- LB needs to
  - tell the server behind the VIP the source address of the client
  - send request to an IP different from the VIP
  - *tell the server the original destination address (ie the VIP to serve)*

A simple matter of communication...

- Where can we put extra information?
- Options considered:
  - Generic Routing Encapsulation (GRE)
  - IP-in-IP tunnelling (RFC1853)

- Main Issues: Path MTU Discovery
  - IP/IP and GRE add 24 bytes overhead
  - largest packet in tunnel would be 1476 bytes => PMTUD issues from client to LB
  - changing server side MTU to 1524 impacts all traffic on that host, may cause PMTUD issues from server to client

A simple matter of communication...

- Where can we put extra information?
- Options considered:
  - Generic Routing Encapsulation (GRE)
  - IP-in-IP tunnelling (RFC1853)

bit offset	0–3	4–7	8–13	14-15	16–18	19–31
0	Version	Header Length	Differentiated Services Code Point	Explicit Congestion Notification	Total Length	
32	Identification				Flags	Fragment Offset
64	Time to Live		Protocol		Header Checksum	
96	Source IP Address					
128	Destination IP Address					
160	Options (if Header Length > 5)					
160 or 192+	Data					

Something-something of Service

- TOS, QoS, COS – none used in Yahoo!'s environment

  => we have 6 completely unused bits

- we don't have to *send* the full address
  - server just needs to derive the full address from the information we relay
- map these bits to VIPs:
  - "DSCP 010001 means original destination address was 1.1.1.2"
  - 010001 => 1.1.1.2
  - ...
- let's reserve a few just in case...
- use of bits 010XXX – 101XXX yields 40 values (per LB)

L3DSR Packet Flow

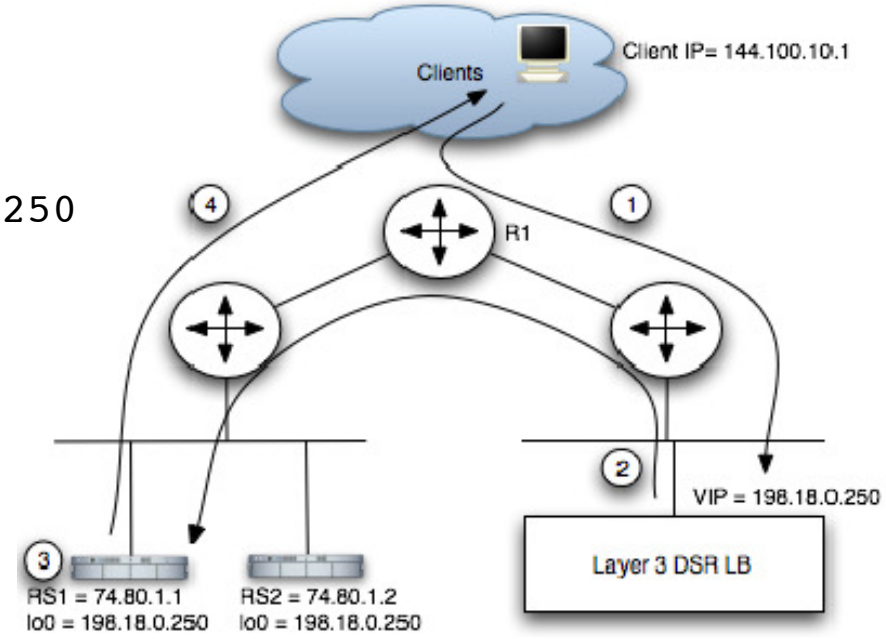
- LB and servers need to agree on DSCP <=> VIP mapping

- LB sets DSCP bit according to known mapping
- LB changes destination address to the server's (real) IP, keeps client's source address

- Server checks DSCP bit
- Server rewrites destination address according to known mapping to appropriate VIP
- VIP configured on loopback alias as with L2DSR
- Server responds to client's source address from VIP

L3DSR Packet Flow

- Incoming:
  - Client Source IP: `144.100.10.1`
  - Client Destination IP: `198.18.0.250`
  - DSCP: `0x0` (explicitly cleared)
- LB to Server:
  - Source IP: `144.100.10.1`
  - Destination IP: `74.80.1.1`
  - DSCP: `0x4`
- Server to Client:
  - Source IP: `198.18.0.250`
  - Destination IP: `144.100.10.1`



	ToS Bits	Source IP	Dest IP
1	0x0	144.100.10.1	198.18.0.250
2	0x4	144.100.10.1	74.80.1.1
3	0x4	144.100.10.1	74.80.1.1
4	0x0	198.18.0.250	144.100.10.1

[Server iptables rewrite rule] → 198.18.0.250

Hardware Vendor Support

- A10 AX3200 >= 2.2.5
- Brocade ADX Series >= 12.1d
- Brocade/Foundry ServerIron 450
  - M7 and JetCore blades
  - >= 10.2.01p
- Citrix Netscaler running 8.x, 9.x

Hardware Vendor Support – A10

```
slb template port dscp_port_7
   dscp 7
!
slb server hostname.yahoo.com 10.128.9.31
   port 80 tcp
      template port dscp_port_7
!
slb service-group vip1.yahoo.com:80 tcp
   health-check status.html
   member hostname.yahoo.com:80
!
slb virtual-server vip1.yahoo.com 10.128.8.45
   port 80 tcp
   service-group vip1.yahoo.com:80
   no-dest-nat
```

Hardware Vendor Support – Brocade

```
server remote-name hostname.yahoo.com 10.128.9.35
 port default disable
 port http
 port http url "GET /index.html"
!
server virtual vip1.yahoo.com 10.1.128.49
 tos-marking 7 hc-l3-dsr
 port default disable
 port http
 bind http hostname.yahoo.com http
!
```

Hardware Vendor Support – Citrix

```
add lb monitor index.html HTTP -respCode 200      \
  -httpRequest "GET /index.html" -interval 7      \
  -retries 2
add server hostname.yahoo.com 10.128.9.35
add service hostname.yahoo.com-80                 \
  hostname.yahoo.com ANY 80                       \
  -usip YES -cip DISABLED
bind lb monitor index.html hostname.yahoo.com-80
add lb vserver vip1.yahoo.com-80 ANY              \
  10.128.8.44 80 -lbmethod ROUNDROBIN             \
  -m TOS -tosId 7
bind lb vserver vip1.yahoo.com-80                 \
  hostname.yahoo.com-80
```

Server OS Support – FreeBSD

- Yahoo! developed kernel module providing a simple single-purpose packet filter
- enable destination-address rewriting via `sysctl(8)`
- originally written by John Baldwin, now maintained by Jan Schaumann
- a whooping 256 lines of code (including comments etc.)
- support version 6 and above (i386/amd64)

Server OS Support – FreeBSD

```
$ sudo /sbin/kldload dscp_rewrite
$ /sbin/kldstat | grep dscp
 5    1 0xffffff8000be9000 106000   dscp_rewrite.ko
$ sudo sysctl -w net.inet.ip.dscp_rewrite.enabled=1
net.inet.ip.dscp_rewrite.enabled: 0 -> 1
$ sudo sysctl -w net.inet.ip.dscp_rewrite.7=10.1.128.49
net.inet.ip.dscp_rewrite.7: 0.0.0.0 -> 10.1.128.49
$ sudo /sbin/ifconfig lo0 10.1.128.49 netmask 0xffffffff alias
$ ifconfig -a
bce0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
        options=1bb<RXCSUM,TXCSUM,VLAN_MTU,VLAN_HWTAGGING,JUMBO_MTU,VLAN_HWCSUM,TSO4>
        ether 00:15:c5:e5:4c:69
        inet 10.128.9.35 netmask 0xffffff00 broadcast 10.128.9.255
        media: Ethernet 1000baseTX <full-duplex>
        status: active
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> metric 0 mtu 16384
        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x5
        inet6 ::1 prefixlen 128
        inet 127.0.0.1 netmask 0xff000000
        inet 10.1.128.49 netmask 0xffffffff
```

Server OS Support – FreeBSD

```
$ ping -c 3 -z 28 10.128.9.35
PING 10.128.9.35 (10.128.9.35): 56 data bytes
64 bytes from 10.1.128.49: icmp_seq=0 ttl=64 time=0.035 ms
64 bytes from 10.1.128.49: icmp_seq=1 ttl=64 time=0.046 ms
64 bytes from 10.1.128.49: icmp_seq=2 ttl=64 time=0.045 ms

--- 10.128.9.35 ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.035/0.042/0.046/0.005 ms


15:25:59.822743 IP (tos 0x1c, ttl 64, id 11612, offset 0, flags [none], proto ICMP (1), length
   84) 10.128.9.35 > 10.128.9.35: ICMP echo request, id 22133, seq 0, length 64
15:25:59.822759 IP (tos 0x1c, ttl 64, id 11613, offset 0, flags [none], proto ICMP (1), length
   84) 10.1.128.49 > 10.128.9.35: ICMP echo reply, id 22133, seq 0, length 64
15:26:00.823815 IP (tos 0x1c, ttl 64, id 11617, offset 0, flags [none], proto ICMP (1), length
   84) 10.128.9.35 > 10.128.9.35: ICMP echo request, id 22133, seq 1, length 64
15:26:00.823831 IP (tos 0x1c, ttl 64, id 11618, offset 0, flags [none], proto ICMP (1), length
   84) 10.1.128.49 > 10.128.9.35: ICMP echo reply, id 22133, seq 1, length 64
15:26:01.835959 IP (tos 0x1c, ttl 64, id 11621, offset 0, flags [none], proto ICMP (1), length
   84) 10.128.9.35 > 10.128.9.35: ICMP echo request, id 22133, seq 2, length 64
```

Server OS Support – RHEL

- Yahoo! developed kernel module and `iptables(8)` plugin to allow mangling of the DSCP field
- enable destination-address rewriting via `iptables(8)`
- originally written and now maintained by Quentin Barnes
- another whooping ~200 lines of code or so
- supports RHEL 4, 5 (i386/amd64; xenU)

Server OS Support – RHEL

```
$ sudo iptables -t mangle -A PREROUTING -m dscp –dscp 7 -j DADDR --set-daddr=10.128.8.46
$ lsmod | grep ipt
iptable_mangle          4545  1
ipt_DADDR               4352  5
ipt_dscp                3137  5
ip_tables              21825  3 iptable_mangle,ipt_DADDR,ipt_dscp
$ sudo iptables -t mangle -L
Chain PREROUTING (policy ACCEPT)
target     prot opt source      destination
DADDR      all  --  anywhere    anywhere      DSCP match 0x1c DADDR set 10.128.8.46
[...]
$ sudo ifconfig lo:1 10.128.8.46 netmask 255.255.255.255
$ ifconfig -a | more
eth0      Link encap:Ethernet  HWaddr 00:11:43:E1:DA:F0
          inet addr:10.128.9.31  Bcast:10.128.9.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
lo:1      Link encap:Local Loopback
          inet addr:10.128.8.46  Mask:255.255.255.255
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
```

Server OS Support – RHEL

```
$ ping -c 3 –Q 28 10.128.9.31
PING 10.128.9.31 (10.128.9.31) 56(84) bytes of data.
64 bytes from 10.128.8.46: icmp_seq=0 ttl=64 time=0.084 ms
64 bytes from 10.128.8.46: icmp_seq=1 ttl=64 time=0.106 ms
64 bytes from 10.128.8.46: icmp_seq=2 ttl=64 time=0.112 ms

--- 10.128.9.31 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.084/0.100/0.112/0.016 ms, pipe 2

07:57:27.121040 IP (tos 0x1c, ttl  64, id 0, offset 0, flags [DF], proto 1, length: 84)
   10.128.9.32 > 10.128.9.31: icmp 64: echo request seq 0
07:57:27.121049 IP (tos 0x1c, ttl  64, id 7562, offset 0, flags [none], proto 1, length: 84)
   10.128.8.46 > 10.128.9.32: icmp 64: echo reply seq 0
07:57:28.120800 IP (tos 0x1c, ttl  64, id 0, offset 0, flags [DF], proto 1, length: 84)
   10.128.9.32 > 10.128.9.31: icmp 64: echo request seq 1
07:57:28.120807 IP (tos 0x1c, ttl  64, id 7563, offset 0, flags [none], proto 1, length: 84)
   10.128.8.46 > 10.128.9.32: icmp 64: echo reply seq 1
07:57:29.120608 IP (tos 0x1c, ttl  64, id 0, offset 0, flags [DF], proto 1, length: 84)
   10.128.9.32 > 10.128.9.31: icmp 64: echo request seq 2
07:57:29.120616 IP (tos 0x1c, ttl  64, id 7564, offset 0, flags [none], proto 1, length: 84)
   10.128.8.46 > 10.128.9.32: icmp 64: echo reply seq 2
```

L3DSR Health Checking

- need to be able to check to see if the iptables plugin/kernel module is in place and working correctly – otherwise traffic may get blackholed

- LB sends:
  - Source IP: <LB IP> Dest IP: <Server IP>
  - Healthcheck URL "GET /index.html"
  - DSCP: 7
- Server replies:
  - Source IP: <VIP IP> Dest IP: <LB IP>
  - Status code "200 OK"

- due to destination address rewriting, source/destination on the LB do *not* match

L3DSR Benefits

- Location Independence of servers within the datacenter
  - we can physically move hosts across different Layer 2 segments
  - we can rebuild a host on a different Layer 2 segment, then use it to replace broken hosts
  - we can deploy new hosts to add to existing VIPs without concern for the physical location / IP allocation within same Layer 2 segment
- Preservation of clients' source address
- DSR Performance
  - up to 8x the number of VIPs per LB compared to other L3 techniques

L3DSR Drawbacks

- use of DSCP bits reserved for L3DSR cannot simultaneously be used for QoS

- need to keep track of the DSCP <=> VIP mappings
  - extend the database storing VIP settings
  - update any tools used to turn centrally stored values into host configuration

- increased complexity
  - host configuration now requires additional kernel modules and configuration steps
  - new code and new configuration options on load balancers
  - more places where things can go wrong

L3DSR Caveats

```
22:43:43.855888 IP (tos 0x1c, ttl 64, id 35851, offset 0, flags [none], proto ICMP (1),
  length 84, bad cksum 0 (->251c)!) 10.163.162.14 > 10.163.163.17: ICMP echo request, id
  6179, seq 0, length 64
22:43:43.856092 IP (tos 0x0, ttl 64, id 16332, offset 0, flags [none], proto ICMP (1),
  length 84) 10.163.163.17 > 10.163.162.14: ICMP echo reply, id 6179, seq 0, length 64
22:43:44.897346 IP (tos 0x1c, ttl 64, id 35856, offset 0, flags [none], proto ICMP (1),
  length 84, bad cksum 0 (->2517)!) 10.163.161.14 > 10.163.163.17: ICMP echo request, id
  6179, seq 1, length 64
```

```
22:43:43.855888 IP (tos 0x0, ttl  64, id 35851, offset 0, flags [none], proto: ICMP (1),
  length: 84) 10.163.162.14 > 10.163.163.17: ICMP echo request, id 6179, seq 0, length
  64
22:43:43.856092 IP (tos 0x0, ttl  64, id 16332, offset 0, flags [none], proto: ICMP (1),
  length: 84, bad cksum 0 (->715b)!) 10.163.163.17 > 10.163.162.14: ICMP echo reply, id
  6179, seq 0, length 64
22:43:44.897346 IP (tos 0x0, ttl  64, id 35856, offset 0, flags [none], proto: ICMP (1),
  length: 84) 10.163.162.14 > 10.163.163.17: ICMP echo request, id 6179, seq 1, length
  64
```

L3DSR Caveats

A10:

```
slb template port dscp_port_7
   dscp 7
```

Brocade:

```
server virtual vip1.yahoo.com 10.1.128.49
 tos-marking 7 hc-l3-dsr
```

Citrix:

```
add lb vserver vip1.yahoo.com-80 ANY                \
  10.128.8.44 80 -lbmethod ROUNDROBIN               \
  -m TOS -tosId 7
```

L3DSR Caveats

```
$ ping -c 3 -z 28 10.128.9.35
PING 10.128.9.35 (10.128.9.35): 56 data bytes
64 bytes from 10.1.128.49: icmp_seq=0 ttl=64 time=0.035 ms
64 bytes from 10.1.128.49: icmp_seq=1 ttl=64 time=0.046 ms
64 bytes from 10.1.128.49: icmp_seq=2 ttl=64 time=0.045 ms

--- 10.128.9.35 ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.035/0.042/0.046/0.005 ms
$
```

DSCP = 7 (configure)

TOS = 28 (check)

Hex = 0x1c (troubleshoot)

```
15:25:59.822743 IP (tos 0x1c, ttl 64, id 11612, offset 0, flags [none], proto ICMP (1), length
   84) 10.128.9.35 > 10.128.9.35: ICMP echo request, id 22133, seq 0, length 64
15:25:59.822759 IP (tos 0x1c, ttl 64, id 11613, offset 0, flags [none], proto ICMP (1), length
   84) 10.1.128.49 > 10.128.9.35: ICMP echo reply, id 22133, seq 0, length 64
15:26:00.823815 IP (tos 0x1c, ttl 64, id 11617, offset 0, flags [none], proto ICMP (1), length
   84) 10.128.9.35 > 10.128.9.35: ICMP echo request, id 22133, seq 1, length 64
15:26:00.823831 IP (tos 0x1c, ttl 64, id 11618, offset 0, flags [none], proto ICMP (1), length
   84) 10.1.128.49 > 10.128.9.35: ICMP echo reply, id 22133, seq 1, length 64
15:26:01.835959 IP (tos 0x1c, ttl 64, id 11621, offset 0, flags [none], proto ICMP (1), length
   84) 10.128.9.35 > 10.128.9.35: ICMP echo request, id 22133, seq 2, length 64
```

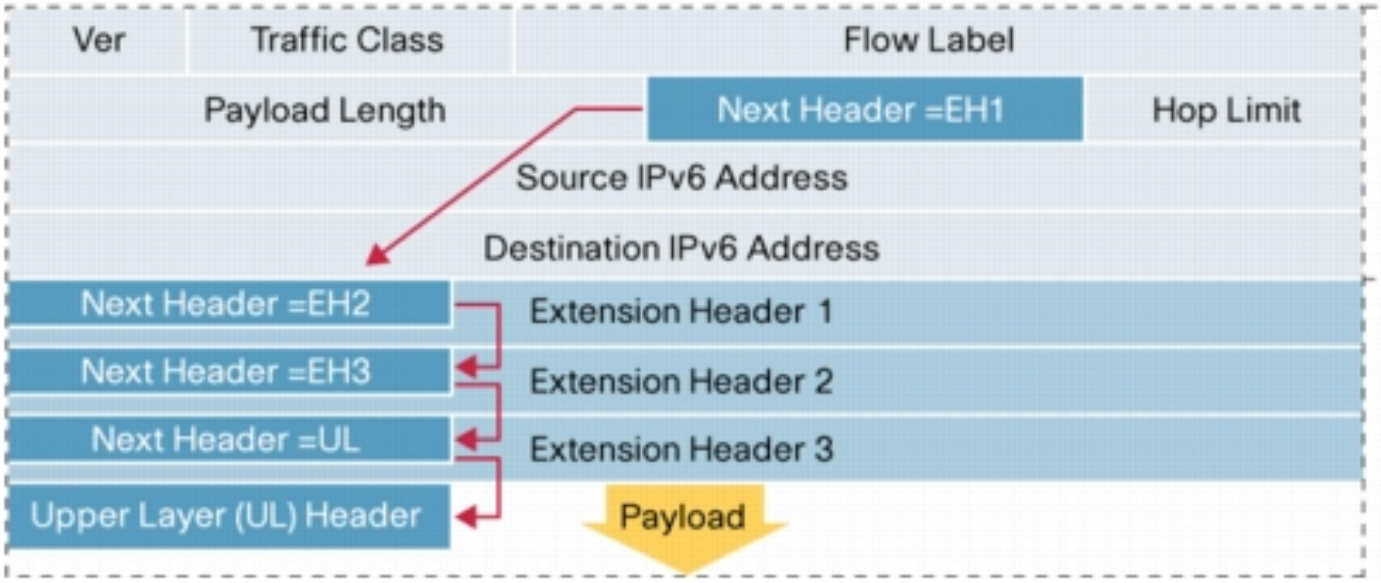
Future Enhancements

- Port Translation
  - now possible – we're on Layer 3
  - may require the server-side kernel module to also rewrite the port of the incoming packets

- Revisit the Loopback Alias
  - *if* hosts are guaranteed to be on separate l2 segments, VIP can be aliased on the real interface
    - Pro: packets don't get pushed up and down the TCP stack on entry
    - Con: hosts now *must* be on different L2 segments; host configuration deviates (more) from other VIP configurations; LB configuration increases in complexity

Future Enhancements: IPv6

- use *Traffic Class* field?
- use *Destination Options* EH?
- define new EH?



Open Source L3DSR

- LB vendor support is already there
- internal process started to open source FreeBSD and Linux kernel modules and `iptables(8)` extension as well as helper tools and glue scripts
- preferred license: BSD
  - linux bits may require GPL
- most likely to be hosted on GitHub

L3DSR at a glance

- allows for Direct Server Return Load Balancing across Layer 3
  - location independence of servers in a VIP
  - high performance
  - preservation of clients' source address
- uses the DSCP field in the IPv4 header to relay information to the servers in a VIP
  - use of DSCP bits reserved for L3DSR cannot be used for QoS
  - debugging/configuration at times confusing
- supported by A10, Brocade/Foundry, Citrix
- server support on FreeBSD and RHEL
  - servers rewrite destination address of incoming packets based on DSCP values
- future enhancements include: IPv6 support, port translation, open sourcing of kernel modules