Active-Active Servers and Connection Synchronisation for LVS

Simon Horman (Horms) – horms@valinux.co.jp
VA Linux Systems Japan K.K. – www.valinux.co.jp
with assistance from
NTT Commware Coporation – www.nttcom.co.jp

16th January 2004

http://www.ultramonkey.org/

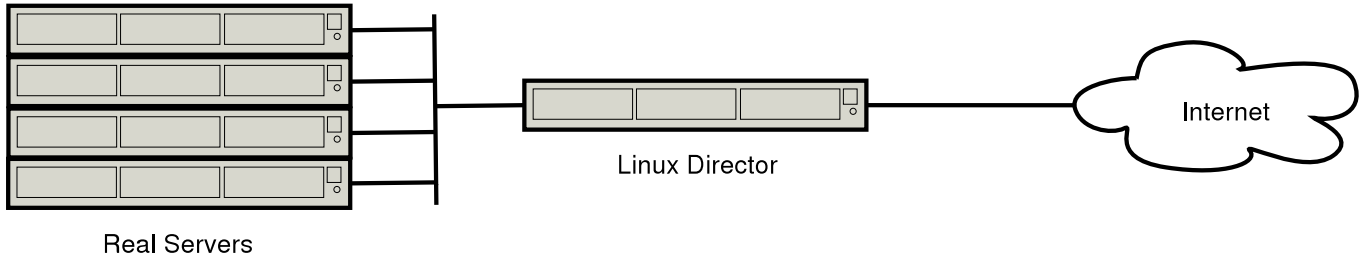
Introduction

Layer 4 Switching

- The Linux Virtual Server Project (LVS)
- Allows networked services to scale beyond a single machine
- Prevailing technology for building large web sites from commodity servers.

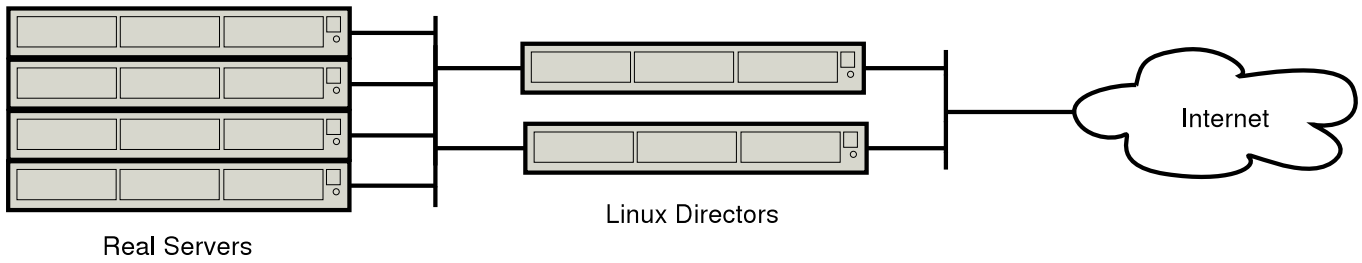
Single Point of Failure

A single Linux Director is a Single point of failure for the load-balanced service.



Active/Stand-By

This is often resolved by having two Linux Directors in an active/stand-by configuration.



Active/Stand-By Problems

When failover occurs active connections are broken.

- Part 1 of this paper will address this problem by synchronising connection information between Linux Directors.

One of the Linux-Directors is idle most of the time.

Real Servers can scale horozontally, however the load-balancing capacity is limited to that of a single Linux-Director for a given virtual service.

- Part 2 of this paper will address these problems by allowing linux directors to work in an Active-Active configuration.

LVS Connection Synchronisation Overview I

When LVS receives a new connection it selects a real server for the connection.

This is done by allocating an ip_vs_conn structure for the connection.

For each subsequent packet of a connection this structure is looked up and the packet is forwarded accordingly.

The ip_vs_conn is also used to effect NAT when LVS-NAT is being used.

Persistance is implemented by creating ip_vs_conn structures that act as a template for subsequent connections.

LVS Connection Synchronisation Overview II

When fail-over occurs, the new active linux director does not have the ip_vs_conn structures for the active connections.

Thus the connections break.

By synchronising the ip_vs_conn structures connections can continue.

Synchronisation is done using kernel daemons that send and receive synchronisation information using multicast.

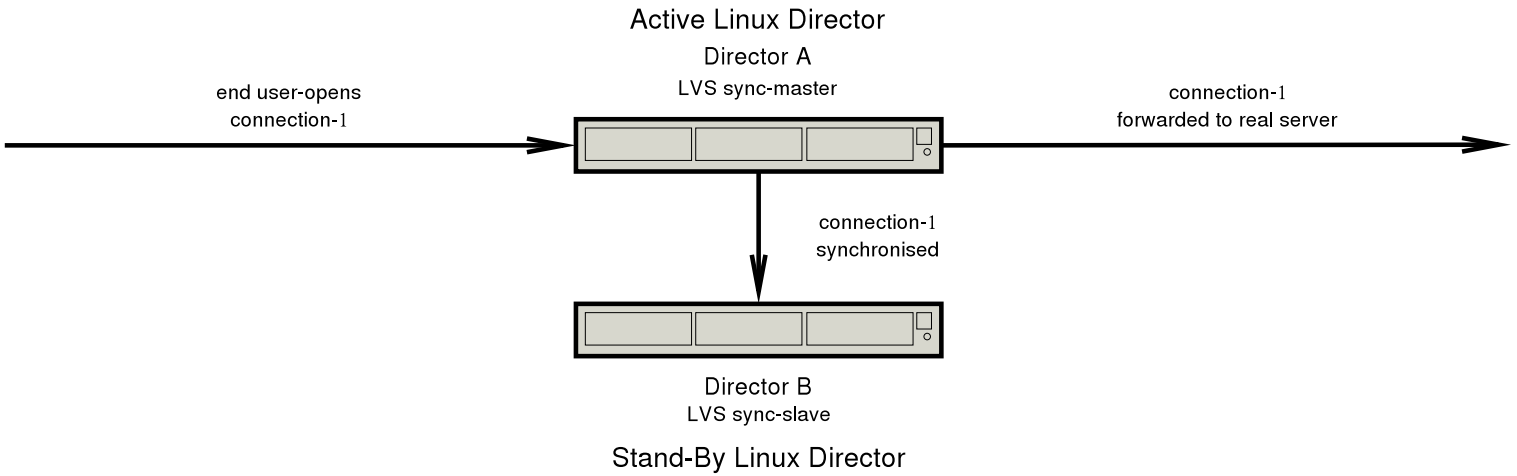
Master/Slave Problem

The existing LVS connection code relies on a sync-master/sync-slave setup

- The sync-master sends information about active connections, but does not recieve information.
- The sync-receive receives information about active connections, but does not send information.

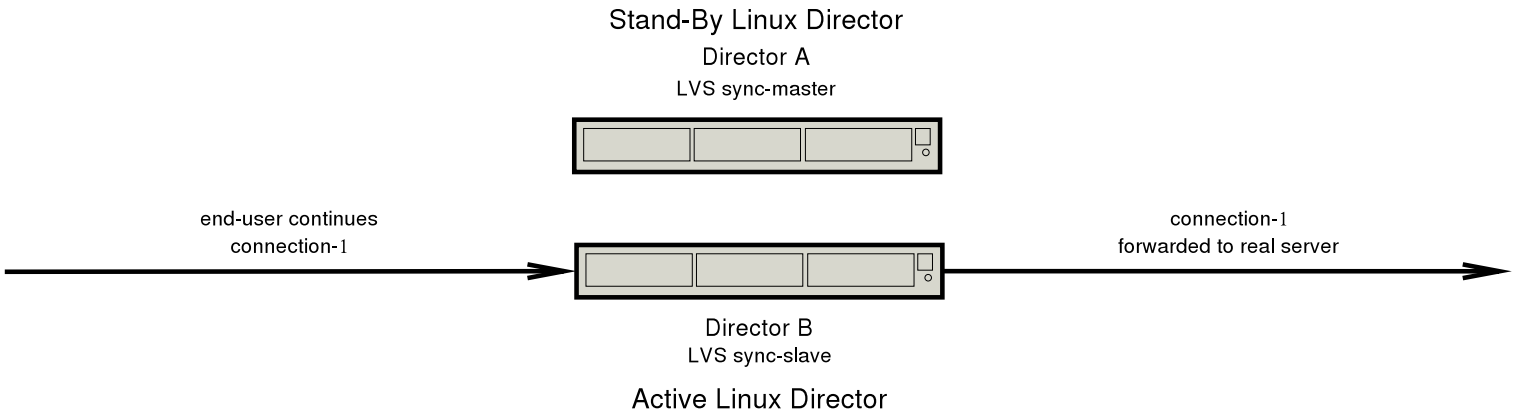
Master/Slave Problem: Example

There are two linux directors, director-a and director-b. Director-A is the LVS sync-master and the active linux director. Director-B is an LVS sync-slave and the stand-by linux director.



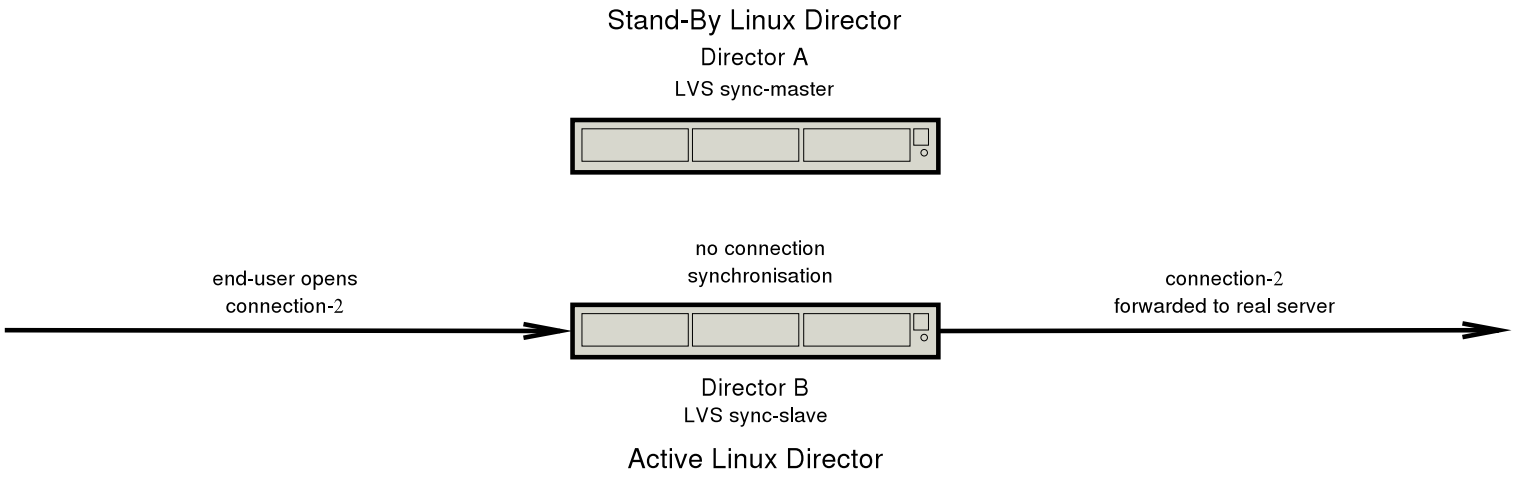
Step 1. An end user opens connection-1. Director-A receives this connection, forwards it to a real server and synchronises it to the sync-slave, director-b.

Master/Slave Problem: Example (continued)



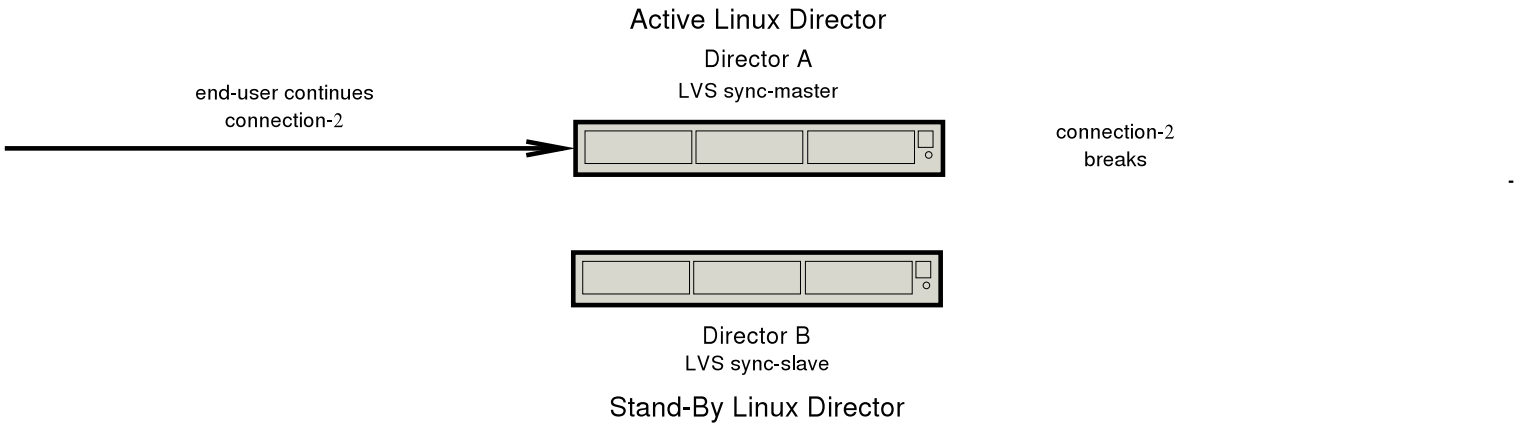
Step 2. A fail-over occurs and director-b becomes the active linux director. Connection-1 is able to continue because of the connection synchronisation that took place in step 1.

Master/Slave Problem: Example (continued)



Step 3. An end user opens connection-2. Director-B receives this connection, and forwards it to a real server. Connection synchronisation does not take place because director-b is a sync-slave.

Master/Slave Problem: Example (continued)



4. Another fail-over takes place and director-a is once again the active linux director. Connection-2 is unable to continue because it was not synchronised.

Master/Slave Problem: Fix

Patches are available for LVS which allow a linux director to run as a *sync-master* and *sync-slave* simultaneously.

- Merged into LVS in the 2.6 kernel
- Patches for LVS in the 2.4 kernel can be found on linuxvirtualserver.org

As an alternate approage, a peer to peer synchronisation system has been developed in user-space.

User-Space Synchronisation Daemon

Implements peer to peer connection synchronisation for LVS

Synchronisation code is in userspace to allow more complex synchronisation daemons to be developed more easily.

Addresses serveral deficiencies in the existing LVS synchronisation protocol.

Modifies the existing synchronisation code to allow arbitary synchronisation mechanisms to be registered in the kernel.

Protocol Deficiencies

Current protocol is very simple

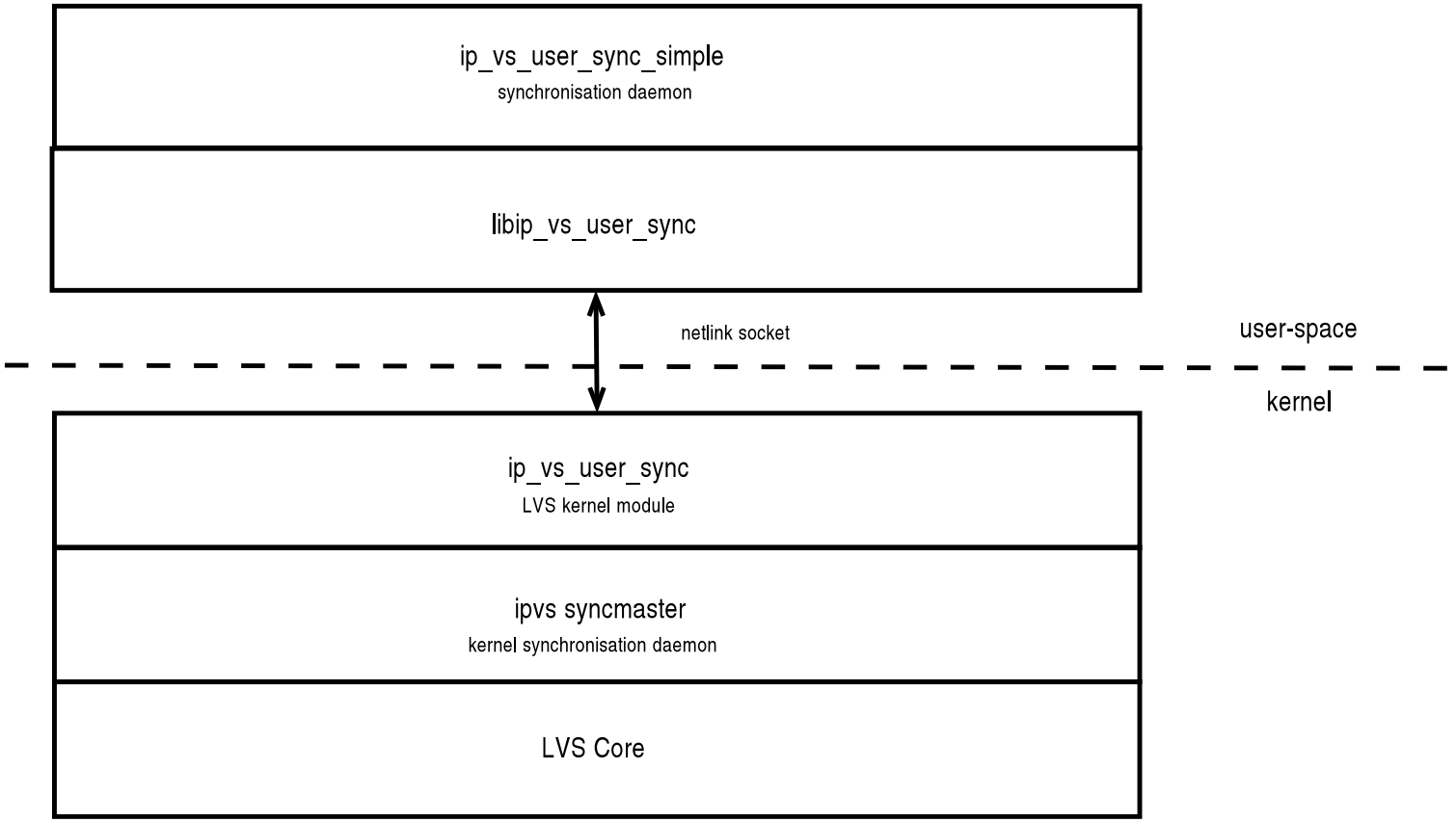
- 4 byte header
- Up to 50 24 or 48 byte connection entries
- Synchroniation packets are up to 1248 bytes long
- Connections are synchronised once they pass a *threshold* of 3 packets. This is configurable at run-time `/proc/sys/net/vs/sync_threshold`
- Connections are synchronised with a *frequency* of every 50 packets

Protocol Improvements

The following improvments have been made without impacting on the simplicity of the design

- Addition of a checksum
- Addition of a version number
- Allow the maximum packet size to be configured at run-time
  `/proc/sys/net/vs/sync_msg_max_size`
- Allow frequency to be configured at run-time
  `/proc/sys/net/vs/sync_frequency`

User-Space Synchronisation: Implementation I



User-Space Synchronisation: Implementation II

A User-Space Daemon communicates with LVS in the kernel using a netlink socket.

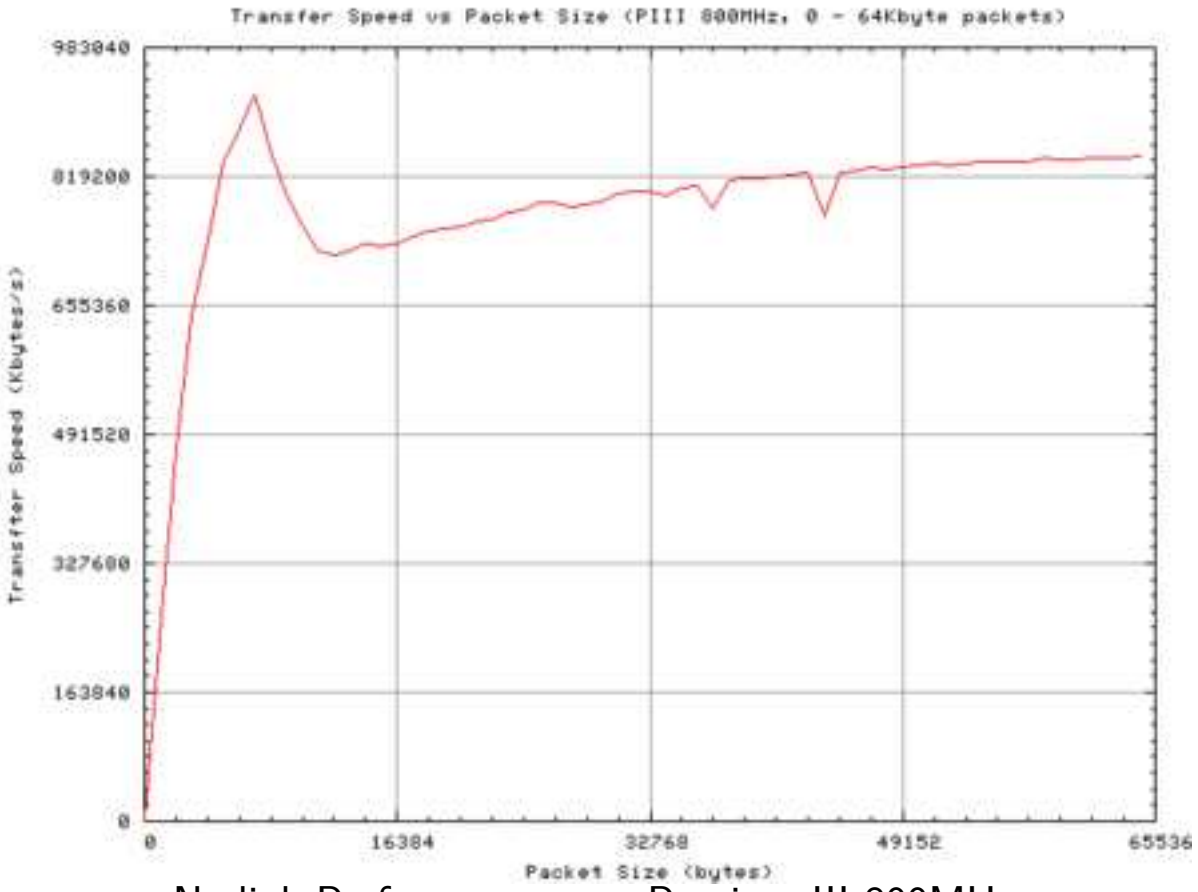
- Synchronisation information is sent and recieved using this socket
- Synchronisation information is sent to other linux directors using multicast

Allows more sophisticated daemons to be developed more easily

The main disadvantage is a possible performance impact.

- Synchronisation data is passed to user space only to have it sent over multicast which goes via the kernel
- However, testing showed that the netlink socket is very fast

Netlink Socket Performance



Netlink Performance on a Pentium III 800MHz

LVS Core: Synchronisation Hooks I

The following hooks were implemented in the LVS Core to allow arbiatry synchronisaion mechanisms to be implemented.

send_mesg	Used by *ipvs syncmaster* to send a packet
open_send	Used by *ipvs syncmaster* to open the socket used to send packets
close_send	Used by *ipvs syncmaster* to close the socket used to send packets
recv_loop	Event loop used by *ipvs syncslave* to receive packets
open_recv	Used by *ipvs syncslave* to open the socket that is used to receive packets
close_recv	Used by *ipvs syncslave* to close the socket that is used to receive packets

LVS Core: Synchronisation Hooks II

The following functions are provided to register functions for these hooks

ip_vs_sync_table_register	Register the hooks. If NULL is supplied for any of the hooks, then the hook will have no effect
ip_vs_sync_table_register_default	Register the default hooks

The default hooks implement the existing sync master/sync slave behaviour

These are registered when LVS is initialised

It is envisaged that the hook functions would be implemented in a separate kernel module

- When the module initialises itself *ip_vs_sync_table_register* should be called
- When the module is unregistering itself, *ip_vs_sync_table_register_default* should be called

ip_vs_user_space

Uses the LVS Synchronisation Hooks to pass synchronisation information to and from user-space

When inserted into the kernel it registers itself

Subsequent synchronisation packets recieved from the LVS Core are sent to user-space via a netlink socket

Synchronisation packets recieved from the netlink socket are passed to the LVS Core

Thus the functionality of both the sync master and sync slave are implemented

Peer to Peer connection synchronisation may be established in place of the old master/slave relationship

libip_vs_user_sync

Convieneicne library for user-space programes to use a netlink socket to communicate with the ip_vs_user_space kernel code

Provides calls analogous to send(), and recv()

Also provides a simple packet structure which is used for kernel/user-space communication.

ip_vs_user_sync_simple

Bare-Bones synchronisation daemon

Illustrates how libip_vs_user_sync and ip_vs_user_space can be used to create a user-space synchronisation daemon.

Suitable for use in both active/stand-by with two linux directors and active-active configurations with any number of linux directors

Sends synchronisation information information recieved via the netlink socket to other linux directors using multicast and vice versa

Active-Active

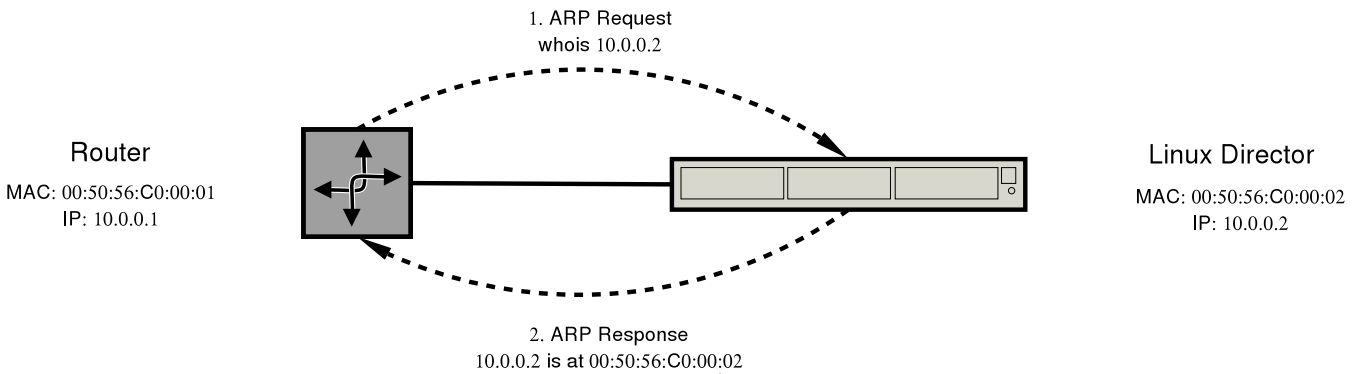
The proposed method of providing Active-Active is to have all linux directors configured with the same Ethernet Hardware (MAC) Address and IP Address.

Designed for use with linux directors

Applicable for any type of host

MAC Addresses

- 48bit integer values
- Used by Ethernet hardware to address each other
- ARP is used to resolve the MAC address for an IP address
- ARP results are typically cached for a few minutes

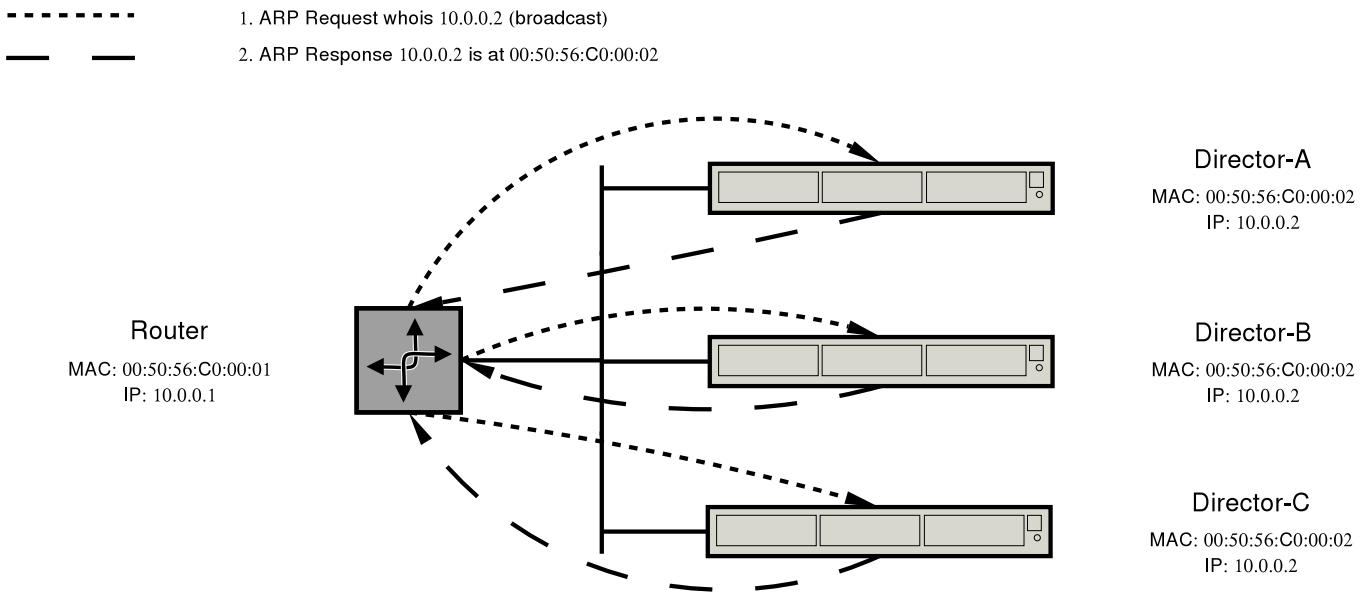


ARP Request/Response

- MAC addresses are preassigned to ethernet equipment by their manufacturer
- Thus the MAC addresses for NICs present on a network should be unique
- Non-broadcast or multicast ethernet frames sent to a given MAC address are received by a single NIC.

A Common MAC and IP Addresses

- Most ethernet hardware allows the MAC address to be changed
- It is possible to configure the NICs of multiple hosts to have the same MAC address
- If multiple hosts have the same MAC and IP then an ARP request for the common IP address will result in multiple ARP responses all with the common MAC address
- Subsequent packets sent to the common IP and thus the common MAC will be recieved by all of the hosts all with the common MAC address



Switch Port Problem

- Switches send packets to for a given MAC address only to the port that MAC address is attached to
- In this case multiple ports have the same MAC address attached
- But the switch may still want so send the packets down only one of these ports
- This means that only one Linux Director will received packets for the common IP and MAC address

Switch Port Solution

- Switches know which MAC address is connected to which port by observing packets which are received from the port
- If the Switch doesn't know which port a MAC address belongs to it will sent packets for that MAC address to *all* ports
- If Packets are never sent with the common MAC address then the switch is not able to associate the MAC address with a port
- Thus packets for the common MAC address will be sent to all ports

Switch Port Solution: Implementation

- The MAC address of the NIC is used to receive packets
  - This is the normal behaviour
- Patch the Kernel so an alternate MAC address can be used to send packets
- Alternate MAC is set using `/proc/sys/net/ipv4/conf/*/outgoing_mac`
- Simple change to ethernet.c to check this proc value and use it if set
- Otherwise the address of the NIC is used

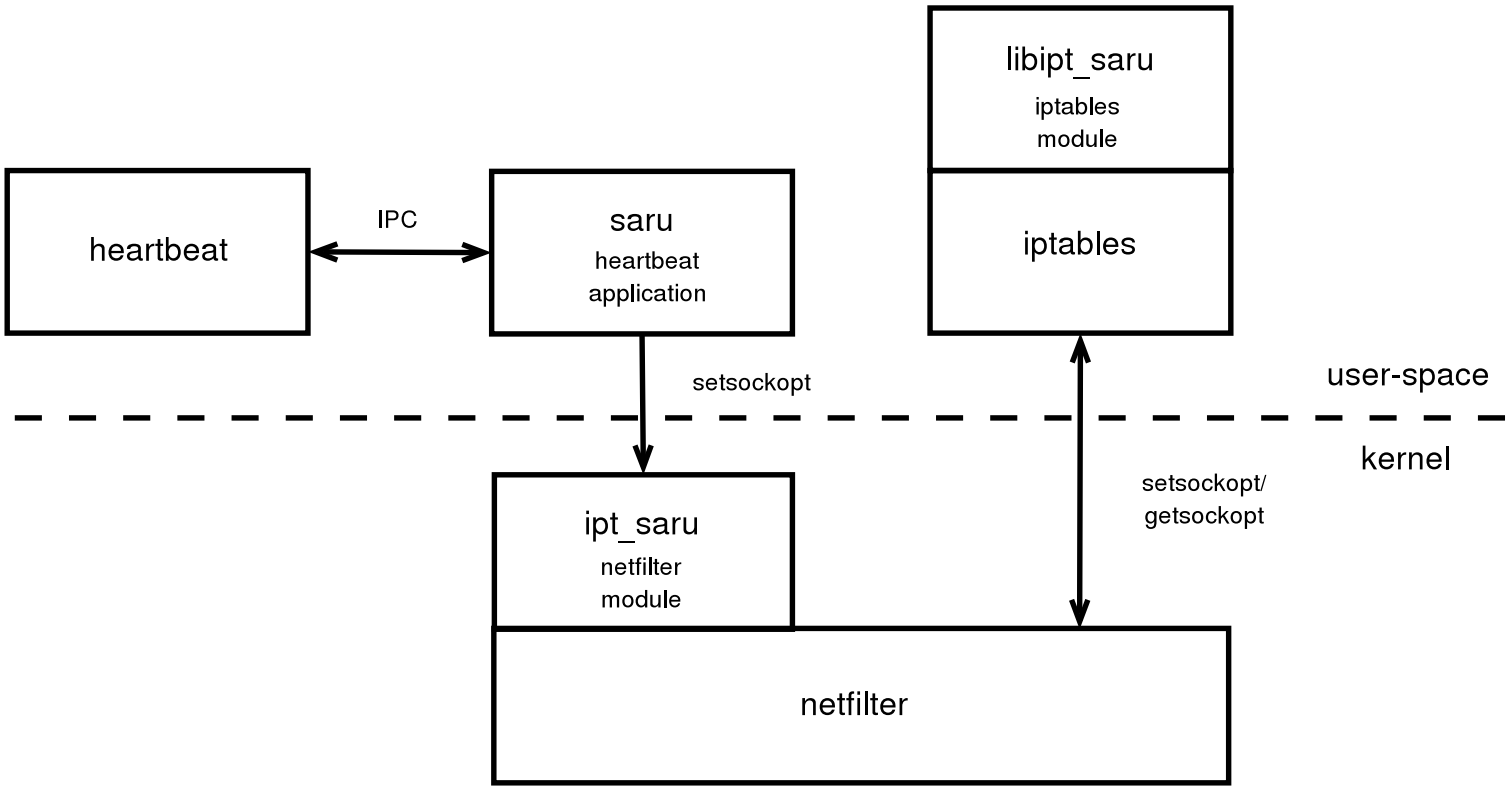
Filtering

- All the Linux Directors receive all packets for the common IP address
- Filtering ensures that only one Linux Director load balances each connection
- This can be done using netfilter/ipchains
- Static filters are easy to establish
- But do not adapt as Linux Directors are added and removed

Dynamic Filtering

- Netfilter module that can have its rules changed on the fly
- User-Space daemon that updates the module's filter rules

Dynamic Filtering Block Diagram



Dynamic Filtering: User Space-Daemon

- Monitors the other Linux Directors in the cluster
- Master is elected to simplify resource allocations
- Master allocates blocks of the available connection-space to the available Linux Directors
- Blocks are reallocated by the master as Linux Directors join and leave the cluster
- If the master leaves the cluster a new one is elected

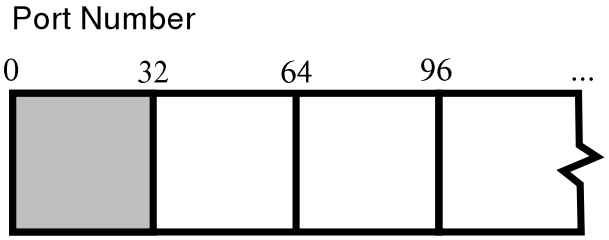
Blocks I

- It is important that the decision about which Linux Director should load balance which connection is made in advance
- Making this decision as connections arrive would be too slow
- This is done by dividing the available connections into blocks

Blocks II

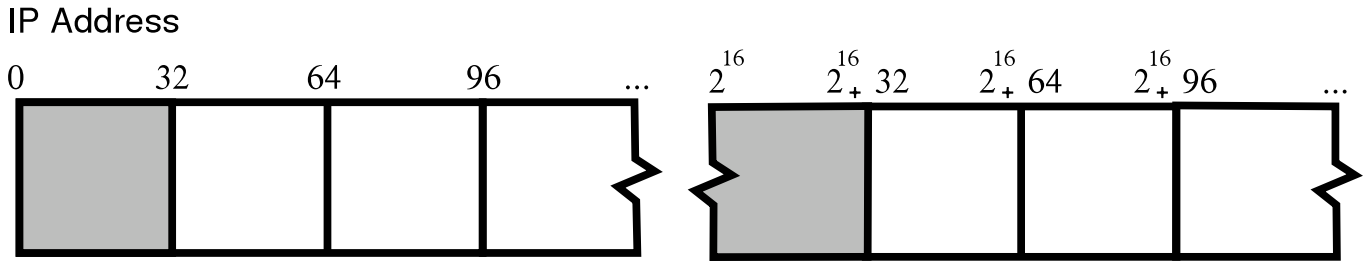
- Currently this can be done by dividing the source or destination ports or IP addresses into into 512 blocks

- Ports are divided up using $port/512$, thus there are $2^{16}/512 = 32$ contiguous ports per block.



Blocks III

- IP addreses are divided up using $(addr \bmod 2^{16})/512$, thus there are 2^{16} sub-blocks of $2^{32-16}/512 = 32$ contiguous addresses per block.



Blocks IV

- When a connection is recieved its block is looked up
- If this block has been allocated to the node by saru then the packet is accepted
- Otherwise it is dropped – another node will accept it

Active-Active Applications

- Designed to run on Linux Directors running LVS
- But the design is generic - may handle any type of network packets
- Thus, may be used on any type of server

Conclusion

LVS is a fast and reliable way to load balance internet services

Connection tracking prevents active/stand-by linux directors from dropping connections when a failover occurs

Active-Active allows linux directors to scale horozontally in the same way that LVS allow real servers to scale