Subject: Re: Network virtualization/isolation
Posted by ebiederm on Sat, 25 Nov 2006 09:09:20 GMT
View Forum Message <> Reply to Message

Daniel Lezcano <dlezcano@fr.ibm.com> writes:

>> Then a matrix of how each requires what modifications in the network
>> code. Of course all players need to agree that the description is
>> accurate.
>> Is there such a document?
>> cheers,
>> jamal
>
> Hi,
>
> the attached document describes the network isolation at the layer 2 and at the
> layer 3, it presents the pros and cons of the different approaches, their common
> points and the impacted network code.
> I hope it will be helpful :)

Roughly it is correctly but I the tradeoffs you describe are incorrect.

> Isolating and virtualizing the network
> ---
>
> Some definitions:
> ---
>
> isolation : This is a restrictive technique which divides a set of the
> available system objects to smaller subsets assigned to a group of
> processes. This technique ensures an application will use only a
> subset of the system resources and will never access other
> resources.
>
> virtualization : This technique gives the illusion to an application
> that its owns all the system resources instead of a subset of them
> provided by the isolation.
>
> container: it is the name of the base element which brings the
> isolation and the virtualization where applications are running into.
>
> system container: operating system running inside a container.
>
> application container : application running inside a container.
>
> checkpoint/restart: take a snapshot of a container at a given time
> and recreate the container from this snapshot.

>
> mobility: checkpoint/restart used to move a container to one host to
> another host.
>
> ---------------------------
>
> Actually, containers are being developed in the kernel with the
> following functions :
>
> * separate the system resources between containers in order
> to avoid an application, running into a container, to
> access the resources outside the container. That
> facilitates the resources management, ensures the
> application is jailed and increases the security.
>
> * virtualize the resources, that avoids resources conflict
> between containers, that allows to run several instance of
> the same servers without modifying its network
> configuration.
>
> * the combination of the isolation and the virtualization is
> the base for the checkpoint/restart. The checkpoint is
> easier because the resources are identified by container
> and the restart is possible because the applications can
> be recreated with the same resources identifier without
> conflicts. For example, the application has the pid 1000,
> it is checkpointed and when it is restarted the same pid
> is assigned to it and it will not conflict because pids are
> isolated and virtualized.
>
> In all the system resources, the network is one of the biggest part
> to isolate and virtualize. Some solutions were proposed, with
> different approaches and different implementations.
>
> Layer 2 isolation and virtualization
> ----------------------------------

Guys this is probably where we need to focus and not look at the
other options until we find insurmountable challenges with this.
We can make it do everything everyone needs.

> The virtualization acts at the network device level. The routes and
> the sockets are isolated. Each container has its own network device
> and its own routes. The network must be configured in each container.
>
> This approach brings a very strong isolation and a perfect
> virtualization for the system containers.
>

>
> - Ingress traffic
>
> The packets arrive to the real network device, outside of the
> container. Depending on the destination, the packets are forwarded to
> the network device assigned to the container. From this point, the
> path is the same and the packets go through the routes and the sockets
> layer because they are isolated into the container.

You don't need the extra hop. The extra hop is only there because there
are not enough physical interfaces on a machine. Plus I think with a little
work this one particular case can be optimized to the point where it
is not significant.

> - Outgoing traffic
>
> The packets go through the sockets, the routes, the network device
> assigned to the container and finally to the real device.
>
>
> Implementation:
> ---------------
>
> Andrey Savochkin, from OpenVZ team, patchset of this approach uses the
> namespace concept. All the network devices are no longer stored into
> the "dev_base_list" but into a list stored into the network namespace
> structure. Each container has its own network namespace. The network
> device access has been changed to access the network device list
> relative to the current namespace's context instead of the global
> network device list. The same has been made for the routing tables,
> they are all relatives to the namespace and are no longer global
> static. The creation of a new network namespace implies the creation
> of a new set of routing table.
>
> After the creation of a container, no network device exists. It is
> created from outside by the container's parent. The communication
> between the new container and the outside is done via a special pair
> device which have each extremities into each namespace. The MAC
> addresses must be specified and these addresses should be handled by
> the containers developers in order to ensure MAC unicity.
>
> After this network device creation step into each namespace, the
> network configuration is done as usual, in other words, with a new
> operating system initialization or with the 'ifconfig' or 'ip'
> command.
>
> ----- ------ ------- ------ ----

> | LAN |<->| eth0 |<->| veth0 |<-|ns(1)|->| eth0 |<->| IP |
> -----   ------   -------   ------   ----

Note. veth is only necessary because there are enough physical
network interfaces to go around.

> (1) : ns = namespace (aka. Virtual Environment).
>
> The advantages of this implementation is the algorithms used by the
> network stack are not touched, only the network data access is
> modified. That's facilitate the maintenance and the evolution of the
> network code. The drawback is in the case of application container,
> the number of containers can be much more important, (hundred of
> them), that implies a number of network devices more important, a
> longer path to go through the virtualization layer and a more
> resources consumption.

>
> Layer 3 isolation and virtualization
> -----------------------------------
>
> The virtualization acts at the IP level. The routes can be isolated
> and the sockets are isolated.
>
> This approach does not bring isolation at the network device
> layer. The isolation and the virtualization is less stronger than the
> layer 2 but it presents a negligible overhead and resource
> consumption near from the non virtualized environment. Furthermore,
> the isolation at the IP level makes the administration very easy.

All administration issues I have seen can be fixed with good tools
and doing things the way the rest of linux does them. Currently
you do things differently.

> - Ingress traffic
>
> The packets arrive to the real device and go through the routes
> engine. From this point, the used route is enough to know to which
> container the traffic can go and the sockets subset assigned to the
> container.

Note this has potentially the highest overhead of them all because
this is the only approach in which it is mandatory to inspect the
network packets to see which container they are in.

My real problem with this approach besides seriously complicating
the administration by not delegating it is that you loose enormous

amounts of power.

> - Outgoing traffic:
>
> The packets go through the sockets, the assigned routes and finally to
> the real device.
>
> The socket are isolated for each container, the current container
> context is used to retrieve the IP address owned by the
> container. When the source address is not specified, the owned IP is
> used to fill the source address of the packet. This is done when doing
> raw, icmp, multicast, broadcast, tcp connection and udp send
> message. If the bind is done on the interface instead of a ip address,
> the source address should be checked to be owned by the container too.
>
>
> Implementation:
> ---------------
>
> Concerning the implementation, several solutions exist. All of them
> rely to the namespace concept but instead of having all the network
> resources relative to the namespace, the namespace pointer is used as
> an identifier.
>
> One of these solutions is the bind filtering. This implementation is
> the simplest to realize but it brings little isolation. If a mobility
> solution must be implemented on the top of that isolation, the bind
> filtering should be coupled with the socket isolation. The bind
> filtering consists in placing several hooks at some strategic points
> into function calls (bind, connect, send datagram, etc ...) in order
> to fill source address and avoid the bind to an IP address outside of
> the container. The container destination should be determined from the
> ingress traffic.
>
> The second solution consists in relying on the route engine to ensure
> the isolation. The routes are all accessible from all the namespaces
> but they contain the information of what namespace they belong. By
> this way, when the traffic is outgoing, only the routes belonging to
> the namespace are used. When the traffic is incoming, it goes through
> a route, because this one has the namespace owner information, the
> traffic can go to the right namespace. The advantage of this approach
> is to have an isolation near of what can provide the layer 2 isolation
> for the IP layer without loss of performances. The drawback is the
> complexity of the code which is strongly linked with the routing
> algorithms and that's do not facilitate the maintenance.
>
>
> ----- ------ -----------

> | LAN |<->| eth0 |<->| ns(1)| IP |
> -----   ------   -----------
>
> (1) : ns = namespace (aka. Virtual Environmenent).
>
> Common points between layer 2 and layer 3 implementations
> ----------
>
> Because the need of the sockets isolation is the same for the layer 2
> and the layer 3, the socket isolation is the same for the two
> approaches.
>
> The t-uple key, source address, source port, destination address,
> destination port is extended with the network namespace. At the bind
> time, the port usage verification is extended with the network
> namespace pointer too. A port is already in use only if the port and
> network namespace match. If the port match but namespace does not
> match, that means the port is in use but in another namespace.
>
> There can be several listening point on the same port with source
> address set to inaddr_any. When an incoming connection arrives, the
> namespace destination is already resolved and the right connection is
> found without ambiguity.
>
>
> Network resources
> ----------
>
> L2 : Layer 2
> L3 : Layer 3
> BF : Bind Filtering
>
>   ----------
> | L2 | L3 | BF |
> ----------
> | Sockets | Isolated | Isolated | Isolated(1) |
> ----------
> | Routes | Isolated | Isolated(2) | X |
> ----------
> | Inetdev | Virtualized | Virtualized | X |
> ----------
> | Network devices | Virtualized | X | X |
> ----------
>
> (1) : The socket should be isolated, in the case of mobility
> (2) : The routes can be isolated or not
>
>

> Network code modifications
> ------------------------
>
>
> | L2 | L3 | BF |
> | struct sock | | |
> | Sockets | hash tables | idem | idem(1) |
> | async sock event | | |
> | Routes | routes table | route cache | X |
> | | route resolver | | |
> | | struct ifaddr | | |
> | Inetdev | X | add addr | X |
> | | del addr | | |
> | | gifconf | | |
> | specific net dev | | | |
> | Netdevice | loopback | X | X |
> | | dev list | | |
>
> (1) if mobility is needed
>
> Solution pros/cons
> ------------------
>
> | L2 | L3 | BF |
> | Isolation | Excellent | Good | Weak |
> | Virtualization | Total | Partial | None |
> | Network setup | Complicated | Trivial | Simple |
> | Overhead | High | Negligible | Negligible |

So you have two columns that you rate these things that I disagree
with, and you left out what the implications are for code maintenance.

1) Network setup.
   Past a certainly point both bind filtering and Daniel's L3 use a new
   paradigm for managing the network code and become nearly impossible for
   system administrators to understand. The classic one is routing packets
   between machines over the loopback interface by accident. Huh?

The L2. Network setup iss simply the cost of setting up a multiple machine network. This is more complicated but it is well understood and well documented today. Plus for the common cases it is easy to get a tool to automate this for you. When you get a complicated network this wins hands down because the existing tools work and you don't have to retrain your sysadmins to understand what is happening.

2) Runtime Overhead.

Your analysis is confused. Bind/Accept filter is much cheaper than doing a per packet evaluation in the route cache of which container it belongs to. Among other things Bind/Accept filtering allows all of the global variables in the network stack to remain global and only touches a slow path. So it is both very simple and very cheap.

Next in line comes L2 using real network devices, and Daniel's L3 thing. Because there are multiple instances of the networking data structures we have an extra pointer indirection.

Finally we get L2 with an extra network stack traversal, because we either need the full power of netfilter and traffic shaping gating access to what a node is doing or we simply don't have enough real network interfaces. I assert that we can optimize the lack of network interfaces away by optimizing the drivers once this becomes an interesting case.

3) Long Term Code Maintenance Overhead.

- A pure L2 implementation. There is a big one time cost of changing all of the variable accesses. Once that transition is complete things just work. All code is shared so there is no real overhead.

- Bind/Connect/Accept filtering. There are so few places in the code this is easy to maintain without sharing code with everyone else.

- Daniel's L3. A big mass of special purpose code with peculiar semantics that no one else in the network stack cares about but is right in the middle of the code.

Eric