moby / moby

Feedback for macvlan/ipvlan experimental networking drivers #21735

New issue

Open mavenugo opened this issue on Apr 2, 2016 · 63 comments

mavenugo commented on Apr 2, 2016 Contributor

#21122 brought in the experimental support for built-in macvlan/ipvlan network drivers. These drivers make use of the native kernel capabilities to plumb the containers with the vlans defined in the underlay / physical network. It is also well documented here : https://github.com/docker/docker/blob/master/experimental/vlan-networks.md.

This issue is opened to get feedback from those users who are interested in this feature. This will help us to move it out of experimental (based on the feedback).

2

mavenugo added area/networking kind/question labels on Apr 2, 2016

kelseyhightower commented on Apr 4, 2016

I was able to test the new ipvlan l3 support and for the most part everything seems to be working except the ability to reach the internet when running on a Google Cloud Platform (GCP) VM.

After a bit of debugging it seems that packets coming from the container connected to the ipvlan l3 network do not hit the iptables `POSTROUTING` which prevents nat from working -- a requirement on GCE since we only route traffic from a valid VM IP address.

Env

```
$ uname -a
Linux docker-next 4.2.0-34-generic #39-Ubuntu SMP Thu Mar 10 22:13:01 UTC 2016 x86_64 x86_64
x86_64 GNU/Linux
```

Docker

```
$ docker version
Client:
 Version:      1.11.0-dev
 API version:  1.24
 Go version:   go1.5.3
 Git commit:   8eb8a1d
 Built:        Sat Apr  2 10:14:06 2016
 OS/Arch:      linux/amd64
 Experimental: true

Server:
 Version:      1.11.0-dev
 API version:  1.24
 Go version:   go1.5.3
 Git commit:   8eb8a1d
 Built:        Sat Apr  2 10:14:06 2016
 OS/Arch:      linux/amd64
 Experimental: true
```

Steps to Reproduce

```
$ sudo docker network create \
  -d ipvlan \
  -o parent=eth0 \
  --subnet=192.168.214.0/24 \
  -o ipvlan_mode=l3 ipnet210
```

```
$ sudo iptables -t nat -A POSTROUTING -s 192.168.214.0/24 !192.168.214.0/24  -j MASQUERADE
```

Assignees
mavenugo

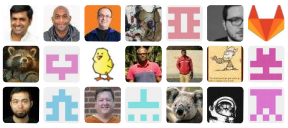
Labels
area/networking
kind/question

Projects
None yet

Milestone
No milestone

Notifications

30 participants



and others

```
$ sudo docker run --net=ipnet210 --rm -ti ubuntu /bin/bash
```

One inside the container try to ping google.com:

```
$ ping google.com
```

```
PING google.com (74.125.201.139) 56(84) bytes of data.
```

> At this point it just hangs because GCE will not route traffic from the container IP.

Running the same command using the default container network works:

```
$ sudo docker run --rm -ti ubuntu /bin/bash
```

```
ping -c 3 google.com
```

```
PING google.com (74.125.201.139) 56(84) bytes of data.
64 bytes from in-in-f139.1e100.net (74.125.201.139): icmp_seq=1 ttl=54 time=0.621 ms
64 bytes from in-in-f139.1e100.net (74.125.201.139): icmp_seq=2 ttl=54 time=0.567 ms
64 bytes from in-in-f139.1e100.net (74.125.201.139): icmp_seq=3 ttl=54 time=0.618 ms

--- google.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.567/0.602/0.621/0.024 ms
```

Troubleshooting

I used tcpdump to figure out the container connected to the ipvlan network was not being masqueraded:

```
$ sudo tcpdump -n -i eth0 icmp
```

```
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
04:34:12.008621 IP 192.168.214.3 > 74.125.69.102: ICMP echo request, id 15, seq 117, length 64
04:34:13.016610 IP 192.168.214.3 > 74.125.69.102: ICMP echo request, id 15, seq 118, length 64
04:34:14.024628 IP 192.168.214.3 > 74.125.69.102: ICMP echo request, id 15, seq 119, length 64
04:34:15.032622 IP 192.168.214.3 > 74.125.69.102: ICMP echo request, id 15, seq 120, length 64
04:34:16.040622 IP 192.168.214.3 > 74.125.69.102: ICMP echo request, id 15, seq 121, length 64
04:34:17.048664 IP 192.168.214.3 > 74.125.69.102: ICMP echo request, id 15, seq 122, length 64
```

> Notice the source IP is 192.168.214.3 which is the container IP and not the host.

The same tcpdump using the default container network:

```
$ sudo tcpdump -n -i eth0 icmp
```

```
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
04:26:19.068969 IP 10.240.0.4 > 209.85.145.113: ICMP echo request, id 17, seq 1, length 64
04:26:19.069806 IP 209.85.145.113 > 10.240.0.4: ICMP echo reply, id 17, seq 1, length 64
04:26:20.070753 IP 10.240.0.4 > 209.85.145.113: ICMP echo request, id 17, seq 2, length 64
04:26:20.071265 IP 209.85.145.113 > 10.240.0.4: ICMP echo reply, id 17, seq 2, length 64
04:26:21.070468 IP 10.240.0.4 > 209.85.145.113: ICMP echo request, id 17, seq 3, length 64
```

> Notice the source IP is 10.240.0.4 -- the host IP.

Additional Information

Host networking

```
docker0   Link encap:Ethernet  HWaddr 02:42:7a:8b:10:eb
          inet addr:172.17.0.1  Bcast:0.0.0.0  Mask:255.255.0.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
```

```
          RX packets:110 errors:0 dropped:0 overruns:0 frame:0
          TX packets:86 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:8189 (8.1 KB)  TX bytes:8461 (8.4 KB)

eth0      Link encap:Ethernet  HWaddr 42:01:0a:f0:00:04
          inet addr:10.240.0.4  Bcast:10.240.0.4  Mask:255.255.255.255
          UP BROADCAST RUNNING MULTICAST  MTU:1460  Metric:1
          RX packets:36914 errors:0 dropped:0 overruns:0 frame:0
          TX packets:17099 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:238455630 (238.4 MB)  TX bytes:1640488 (1.6 MB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lxcbr0    Link encap:Ethernet  HWaddr f2:0a:df:77:29:43
          inet addr:10.0.3.1  Bcast:0.0.0.0  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

2

thaJeztah commented on Apr 4, 2016 — Member

Thanks for testing, **@kelseyhightower**

kelseyhightower commented on Apr 4, 2016

@thaJeztah This is an awesome feature and I want to help. I'm ready for more testing if anyone has idea, but it seems like the limitation here is that ipvlan does get processed by iptables. I understand there are patches inflight to "fix" this, but for now it's safe to say docs will be needed in the sort term to call out what to expect when running on most cloud platforms.

I'm going to retest using my own gateway in GCE and report back. Ideally this should work and if so I'll be happy to contribute docs for the community.

mavenugo commented on Apr 4, 2016 — Contributor

@kelseyhightower thanks for your feedback. **@nerdalert** & I are tag teaming to get your questions answered. As you said, there may be possible limitations with Netfilter on ipvlan. We were primarily targeting the underlay integration by plumbing on the vlan (for L2 mode) and exchange routes (for L3 mode). But, I guess it would be reasonable to try out the GCE / Azure / AWS cases and clearly document the process.

Thanks again for your help.

nerdalert commented on Apr 4, 2016 — Contributor

For sure thanks **@kelseyhightower** Getting GCE lab up tonight (thanks to ur assist). Like **@mavenugo** said we were thinking underlay with route (re)distribution. Would be cool to get the Iptables support for cloud providers once the Linux pieces are upstreamed. Like the idea on clearly pointing out cloud compatibility.

Just for others reading the thread that aren't up to speed Im going to add a bit of context on the modes wrt to cloud scenarios. Ipvlan L3 mode is arguably so scalable and attractive because allow any flooding of broadcast/multicast traffic. That has positive stability and security implications. The notion of subnets and the given constraints, become much less relevant since any address can talk to any address if they share the same parent interface. It operates almost identically like a router on the network, if an interface is not redistributed back into the IGP/EGP then neighboring routing tables have no knowledge of the network/subnet/prefix/host/etc. But nothing is free, so the downside is it requires some basic route distribution since L3 TX processing occurs in the IP stack on the slave interface (container eth0).

Bit of context on Ipvlan L2 and Macvlan Bridge modes. They are both out of the box functional since it relies on flooding and learning just like any other L2 scenario which makes them a good place to start for general usage as the require nothing extra for multi-host/Internet communications. You can point it to tagged existing interfaces, VLAN tagged sub-interfaces that will dynamically be provisioned for the user or even bonded ethernet pairs (bond0)/LAG/mLAG/teamed_NICs/etc. Generally all brownfield deployments use VLANs, interested to see how CSPs handle them. Last note, Macvlan bridge mode uses a unique MAC per container while Ipvlan L2 mode uses the same MAC addr as the `-o parent=` interface. Ipvlan L2 mode traffic will appear no different when leaving the host from a layer2 perspective, the difference (and limitation) will be if the provider is locking down IP traffic to a particular IP/MAC pairing.

Thanks again for the helpful convo and feedback!

kelseyhightower commented on Apr 5, 2016

There are a few gotchas with ipvlan running in L3 mode. It seems I cannot ping the host, or in this case get a response end-to-end:

The Container

```
kelseyhightower@docker0:~$ sudo docker run -ti --net=ipvlan busybox /bin/ping docker0
```

```
PING docker0 (10.240.0.3): 56 data bytes
^C
--- docker0 ping statistics ---
14 packets transmitted, 0 packets received, 100% packet loss
```

The Host

```
$ sudo tcpdump -n -i eth0 icmp
```

```
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
15:28:08.946223 IP 10.240.0.3 > 192.168.0.2: ICMP echo reply, id 256, seq 4, length 64
15:28:09.946387 IP 10.240.0.3 > 192.168.0.2: ICMP echo reply, id 256, seq 5, length 64
15:28:10.946519 IP 10.240.0.3 > 192.168.0.2: ICMP echo reply, id 256, seq 6, length 64
15:28:11.946647 IP 10.240.0.3 > 192.168.0.2: ICMP echo reply, id 256, seq 7, length 64
15:28:12.946777 IP 10.240.0.3 > 192.168.0.2: ICMP echo reply, id 256, seq 8, length 64
15:28:13.946932 IP 10.240.0.3 > 192.168.0.2: ICMP echo reply, id 256, seq 9, length 64
15:28:14.947066 IP 10.240.0.3 > 192.168.0.2: ICMP echo reply, id 256, seq 10, length 64
15:28:15.947215 IP 10.240.0.3 > 192.168.0.2: ICMP echo reply, id 256, seq 11, length 64
15:28:16.947340 IP 10.240.0.3 > 192.168.0.2: ICMP echo reply, id 256, seq 12, length 64
^C
9 packets captured
9 packets received by filter
0 packets dropped by kernel
```

Notice we don't wee the request coming in, but we see a reply. Inside the container the response is also dropped.

nerdalert commented on Apr 5, 2016 Contributor

Really glad you mentioned that limitation. Its one of the gotchas that users will hit. I actually really appreciate the elegance of the design. By isolating the parent interface it closes off a potential backdoor into the underlying infra. Its similar to keep the network management addresses on a separate VLAN or VRF in general network designs. This will be required for regulatory and general best practices.

The doc is also loaded up with warnings on this but it will still be a recurring question just because it is so natural to ping localhost connectivity as a first step.

```
# NOTE: the containers can NOT ping the underlying host interfaces as
# they are intentionally filtered by Linux for additional isolation.
# In this case the containers cannot ping the -o parent=172.16.86.250

# NOTE: the containers can NOT ping the underlying host interfaces as
# they are intentionally filtered by Linux for additional isolation.

Note: In both Macvlan and Ipvlan you are not able to ping or communicate with the default
namespace IP address. For example, if you create a container and try to ping the Docker host's
```

```
eth0 it will not work. That traffic is explicitly filtered by the kernel modules themselves to
offer additional provider isolation and security.

The default namespace is not reachable per ipvlan design in order to isolate container namespaces
from the underlying host.
```

Thanks again for pointing this one out **@kelseyhightower**, its an important facet in many aspects.

1

kelseyhightower commented on Apr 5, 2016

@nerdalert Ok, this is great to know this is by design. I'm in process of standing up my own gateway in GCE and if that works I'll be in good shape for now.

fredhsu commented on Apr 22, 2016

I've been running through scenarios with both mac and ipvlan at layer 2, so far so good. Its a great way to provide traditional layer 2 networking for containers in a clean and simple fashion. It feels easier for a network person to troubleshoot and support.

ipvlan layer 3 will be a good tool for larger scale networks/applications that don't need layer 2 adjacency. I'm looking forward to having these as a networking option.

nerdalert commented on Apr 23, 2016 Contributor

Thanks for the feedback **@fredhsu**!

julianxhokaxhiu commented on May 16, 2016 • edited

Hi, I'm using the latest `1.11.1` version of Docker (mainline from Arch Linux). Is `macvlan` driver available inside Docker? Wherever I try to create a bridge network it just says `Error response from daemon: plugin not found`.

I suppose it's still on `dev` branch right? Thanks in advance

Nevermind! Sorry, I didn't saw docker#21711

dreamcat4 commented on May 16, 2016

The problem here seems to be that the experimental build automatically downloads a nightlies git (which may contain other bugs / breakages we are not interested in for testing this drivers here).

Wheras what would prefer is some experimental build version of the 1.11.1 release. No nightlies, no git. Its hard to get that from the install script. Perhaps there is the 'download standalone binary and replace the files' option. But then that might confuse apt etc when upgrading later. So some cleaner / clearer install options / install instructions for ubuntu, that doesnt hide everything and take away all control from the user in some multi-platform script... would be really appreciated here. Still havent been able to try these new drivers for that very reason.

Is there some kind of a timeline when regular users can see these drivers in release versions of docker? Reading from what was the release notes the (macvlan) L2 driver already may be considerably better than my current L2 solution + work-arounds. Which was always broke / has problematic bug in it.

mavenugo commented on May 16, 2016 Contributor

@dreamcat4 If there are other issues blocking you from trying out the feature, please open issues in this repo and it will be addressed accordingly. Since CI runs for every PR and it covers experimental builds as well, we may have to increase the coverage if you are hitting consistent problems.

> Is there some kind of a timeline when regular users can see these drivers in release versions of docker?

We are currently addressing the feedback received during testing. Especially to make these drivers work properly in a multi-host scenario. Please give it a try and share your feedback. Without user feedback it will be hard to get these drivers out of experimental.

dreamcat4 commented on Jun 1, 2016

> Please give it a try and share your feedback. Without user feedback it will be hard to get these drivers out of experimental.

@mavenugo It really sounds like you did not properly read / or fully understand my comments.

jamsix commented on Jun 7, 2016

I've been successfully running all 3 modes in my lab for weeks now, without a single problem to report. These drivers solve all connectivity issues I had with current docker networks without touching `ip link`, `ip netns` and other "behind the scene" wizardry end users are generally not interested in.

What these drivers currently lack is public awareness that they can solve network issues long plaguing docker, so they need to get out of the experimental build asap. The sooner this masterpiece sees the wider deployment, the sooner community will address the missing pieces in the general docker networking puzzle, i.e. IPAM DHCP driver, ipvlan L3 IGP/EGP/IGMP support, etc.

I know this is a kernel driver feature but seems relevant, since docker ipvlan driver could be a major driver for ipvlan L3 deployment. Since ipvlan L3 network behaves like a router from the network point of view (forwards packets between different subnets, drops multicast/broadcast) shouldn't it be reducing TTL/hop count value of the IP packets. Not so much as the endless loop prevention mechanism, I believe is not possible with current ipvlan L3 implementation, but as a troubleshooting tool that returns traceroutes analogue to physical networks.

yongtang commented on Jun 8, 2016 Member

I played with the experimental docker networking function of ipvlan recently. I understand what ipvlan and macvlan is capable of but my focus is mainly for cloud (AWS) and I am mostly concerned with the cross-host connectivity.

For ipvlan L3 mode, it is very similar to the default bridge mode (except there is no bridge interface). As a result, in order for one container in one host to communicate with another container in another host, I essentially added routes on both hosts:
Host 1 (172.x.x.100):

```
sudo docker network create -d ipvlan --subnet=10.1.100.0/24 --gateway=10.1.100.1 -o
ipvlan_mode=l3 -o parent=eth0 ipvlan100
sudo ip route add 10.1.101.0/24 via 172.x.x.200 dev eth0
sudo docker run --name=c100 --net=ipvlan100 --ip=10.1.100.10 -itd alpine /bin/sh
```

Host 2 (172.x.x.200):

```
sudo docker network create -d ipvlan --subnet=10.1.101.0/24 --gateway=10.1.101.1 -o
ipvlan_mode=l3 -o parent=eth0 ipvlan101
sudo ip route add 10.1.100.0/24 via 172.x.x.100 dev eth0
sudo docker run --name=c101 --net=ipvlan101 --ip=10.1.101.10 -itd alpine /bin/sh
```

After that, container c100 and container c101 could communicate with each other.

Compared with bridge mode, ipvlan L3 mode potentially could reduce the latency. Though from maintenance point of view, route still needs to be maintained.

So I would say ipvlan L3 mode is kind of only an `enhancement` on cloud (AWS) environment.

yongtang commented on Jun 8, 2016 Member

For ipvlan L2 mode, I actually encountered quite a few issues on EC2. The biggest issue is that AWS seems to disable the broadcast and use a proxy ARP, which will not return the mac address of the IP if the IP is not managed by AWS.

I am still trying to see if there is a way to get around this in AWS.

nerdalert commented on Jun 12, 2016 Contributor

Thanks **@yongtang**, the drivers are definitely targeted at integration into existing data centers where Ops has control of the underlying network. That said, I am optimistic the abilities L3 Ipvlan open will change networking over time in departing from L2 views of the world. Thanks again for the feedback!

dreamcat4 commented on Jun 22, 2016

Hi there. Have tried the MACVLAN driver today. It looks pretty good. However the host machine cannot communicate with its containers. I would like to know how to do this please. Its an important feature. Cannot understand how we are supposed to do it since the documentation does not explain how.

All ping / port scan on host machine --> container fails.

ghost commented on Jun 22, 2016

> - **Note:** In both Macvlan and Ipvlan you are not able to ping or communicate with the default namespace IP address. For example, if you create a container and try to ping the Docker host's `eth0` it will **not** work. That traffic is explicitly filtered by the kernel modules themselves to offer additional provider isolation and security.

it's expected behavior.
don't remember why, but have seen macvlan/macvtap limitations explained *somewhere* before.

dreamcat4 commented on Jun 22, 2016 • edited

> don't remember why, but have seen macvlan/macvtap limitations explained somewhere before.

OK. Then I need to know a way to work around the issue.

[EDIT] if my docker host machines has 2 network adapter cards, then can manually add a route going through the other NIC. (to a few of the container ips).

But what if it only has 1 NIC on the machine? (as its a laptop, the only other NIC is wifi, which is not very fast / sometimes the signal is not available. Or if its sat on the end of a VPN link... leads to single NIC situation once again.

inoc603 commented on Jun 29, 2016 • edited

I've tried macvlan an ipvlan l2 mode with sucess, ~~but I got `Error response from daemon: plugin not found` when trying to create an ipvlan l3 network.~~

Update

Thanks to **@sanimej**, I found the stupid typo in my command and fix the issue. Ipvlan l3 mode is working as expected. So I deleted all the details to save some space. Again sorry for the simple mistake.

Macvlan and ipvlan l2 mode are pretty straight forward, though using ipvlan l3 mode takes a little more effort to configure routes for cross-host connectivity. With the method suggested by **@yongtang** , I'm able to connect containers on two hosts in the same network. But containers on both hosts have no internet access, unlike with macvlan and ipvlan l2. I'm very new to network stuff. Do I have to configure the router or is there any workaround to give internet access to ipvlan l3 containers?

Also I found it very difficult to run those examples in some cloud environment where there is strict mac and ip limitation. Running a container in ipvlan l2 mode even breaks my ssh connection until I manually remove the network.

I'll continue to do some benchmarks on these drivers, and see how they improve network performance. Are there any existing ones to show the difference?

sanimej commented on Jun 29, 2016

@inoc603 If the command you gave is exactly what you used there is a typo in the driver name, `-d ipvaln`

1

surajssd commented on Jul 4, 2016 • edited

Containers on different host getting same MAC and same IP address?

Host 1

I did this to create network like this

```
$ docker network  create  -d macvlan --subnet=192.168.121.0/24 --gateway=192.168.121.1 -o macvlan_mc
6b530f55bd4494ba93b023115c18b35e7d906836d74c3aff9030ac0dc0eae42d
```

whose network looks like this

```
$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:54:00:e4:47:a8 brd ff:ff:ff:ff:ff:ff
    inet 192.168.121.224/24 brd 192.168.121.255 scope global dynamic eth0
       valid_lft 2624sec preferred_lft 2624sec
    inet6 fe80::5054:ff:fee4:47a8/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:9d:61:5f:ae brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:9dff:fe61:5fae/64 scope link
       valid_lft forever preferred_lft forever
```

started a container

```
$ docker run --net=macvlan1 -it --rm  --name fedora1 fedora:my bash
[root@60752c949607 /]# ip a sh eth0
25: eth0@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default
    link/ether 02:42:c0:a8:79:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.121.2/24 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:c0ff:fea8:7902/64 scope link
       valid_lft forever preferred_lft forever
```

Host 2

Created network similarly

```
$ docker network  create  -d macvlan --subnet=192.168.121.0/24 --gateway=192.168.121.1 -o macvlan_mc
c1d02b472519af23cbcdefbaa1a6469a21703f2422d3d64f0aa23a4964f9069c
```

whose network looks like this

```
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:54:00:e3:11:22 brd ff:ff:ff:ff:ff:ff
    inet 192.168.121.8/24 brd 192.168.121.255 scope global dynamic eth0
       valid_lft 2853sec preferred_lft 2853sec
    inet6 fe80::5054:ff:fee3:1122/64 scope link
```

```
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:82:47:18:02 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:82ff:fe47:1802/64 scope link
       valid_lft forever preferred_lft forever
```

started a container

```
$ docker run --net=macvlan2 -it --rm --name fedora2 fedora:my bash
[root@a28a73e939d3 /]# ip a sh eth0
14: eth0@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default
    link/ether 02:42:c0:a8:79:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.121.2/24 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:c0ff:fea8:7902/64 scope link
       valid_lft forever preferred_lft forever
```

Both containers on different host are getting same MAC address and therefore same IP address, can someone tell me what is that I am doing wrong?

Both machine has following docker version

```
$ docker version
Client:
 Version:      1.12.0-rc3
 API version:  1.24
 Go version:   go1.6.2
 Git commit:   91e29e8
 Built:
 OS/Arch:      linux/amd64
 Experimental: true

Server:
 Version:      1.12.0-rc3
 API version:  1.24
 Go version:   go1.6.2
 Git commit:   91e29e8
 Built:
 OS/Arch:      linux/amd64
 Experimental: true
```

And machine info

```
$ uname -a
Linux fedora1 4.2.3-300.fc23.x86_64 #1 SMP Mon Oct 5 15:42:54 UTC 2015 x86_64 x86_64 x86_64 GNU/Linu
```

inoc603 commented on Jul 4, 2016

Hi **@surajssd** , how are you running your 2 hosts? From the info you provided, your 2 hosts have the same MAC `52:54:00:e4:47:a8` and IP `192.168.121.224/24`. Are you sure the host information is right?

surajssd commented on Jul 4, 2016

OTOH if I specify `--ip=192.168.121.x` manually then this assigns the IP address also generates new MAC address

on machine 1

```
[vagrant@fedora1 ~]$ docker run --net=macvlan1 -it --rm --ip=192.168.121.80 --name fedora1 fedora:my
[root@7d7f7b63fd5d /]# ip a sh eth0
27: eth0@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default
    link/ether 02:42:c0:a8:79:50 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.121.80/24 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:c0ff:fea8:7950/64 scope link
       valid_lft forever preferred_lft forever
```

on second machine

```
[vagrant@fedora2 ~]$ docker run --net=macvlan2 -it --rm  --ip=192.168.121.81 --name fedora2 fedora:m
[root@5c110bf0d19e /]# ip a sh eth0
13: eth0@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default
    link/ether 02:42:c0:a8:79:51 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.121.81/24 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:c0ff:fea8:7951/64 scope link
       valid_lft forever preferred_lft forever
```

surajssd commented on Jul 4, 2016 • edited

@inoc603 mistakenly i sent wrong info for the first time it seems :(

I have updated the comment!

KalleDK commented on Jul 9, 2016

Hi. I'm trying to test a setup where the host has two nics, and I want one to be the hosts, and the other to be dedicated for the docker containers. I feel like I've tried everything, but nothing seems to work. This is via a centos minimal install, where test.docker is installed. I got eth0 as the hosts, everything works fine there. Eth1 is the dedicated for the clients.

docker network create -d macvlan --subnet=172.20.20.0/24 --gateway=172.20.20.1 -o parent=eth1 macnet

docker run --net=macnet -it --rm --ip=172.20.20.214 debian bash

Nut i can't even ping the gateway from inside the container - Am I doing something wrong ?

fredhsu commented on Jul 15, 2016

@KalleDK Just to sanity check, are you using any 802.1q tags between eth1 on your host and your switch? Also, I'm assuming 172.20.20.1 is a switch/router upstream of your server and not on the host itself?

1

KalleDK commented on Jul 17, 2016

I finally figures it out - and it was my problem. I had forgot to allow VM's interfaces, to be in promiscious mode.

nickmaleao commented on Aug 10, 2016

Hello

I'm having a default gateway problem when i try to connect two networks simultaneously to the same container.

Example:

docker network create -d macvlan
--subnet=87.x.x.248/29
--gateway=87.x.x.249
-o parent=public0 public_docker

docker network create -d macvlan
--subnet=10.0.0.0/24 \
--gateway=10.0.0.1
-o parent=priv0 priv-docker

Test1:

1 - docker run --name net1 --net=public_docker --cap-add NET_ADMIN --cap-add=NET_RAW --cap-add NET_BROADCAST --ip=87.x.x.253 -itd alpine /bin/sh - default gateway=87.x.x.249
2 - docker network connect --ip 10.0.0.250 priv-docker net1 -> default gateway is overwritten

Test2:

1 - docker run --name net1 --net=priv-docker --cap-add NET_ADMIN --cap-add=NET_RAW --cap-add NET_BROADCAST --ip=10.0.0.250 -itd alpine /bin/sh - default gateway = 10.0.0.1
2 - docker network connect --ip 87.x.x.249 public_docker net1 -> default gateway remains the same 10.0.0.1.

No matter what the order of the configuration, is seems that the driver gives preference to the private ip address, is there a way to configure only a network but without any default gateway.

The only way i have to resolve this problem, is to manually change the gateway in the container.

nerdalert commented on Aug 10, 2016 Contributor

Hi **@nickmaleao**. The use case was designed to have an external router to route between networks. The driver uses the Libnetwork default IPAM functionality. I have seen a request similar to this that **@aboch** and I looked at but it was using Ipvlan which simply sets the default gateway to the interface but that is an experimental driver. Short of adding a flag to the driver that is saying ignore the default IPAM I can't think of any way to fit that use case.

I'm curious, when you manually change the container GW can the two containers ping one another? Without something proxy arping between the two subnets it shouldn't work in my mind. Ipvlan L3 mode can route between anything. I've never tried `docker connect` w/ the plan drivers since the use case was for underlay integration. Adding **@aboch** in case he has some insight. Thanks!

thaJeztah referenced this issue on Aug 11, 2016

cannot connect to gateway using macvlan network #25607 Open

nickmaleao commented on Aug 11, 2016 • edited by thaJeztah

Hi **@nerdalert** and **@aboch** , thanks for the information.

Regarding your question,apart from the host, both containers can ping themselves and any host that belongs to the lan 10.0.0.0/24,

Container Net1
Kernel IP routing table

```
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         87.103.120.249  0.0.0.0         UG    0      0        0 eth1
10.0.0.0        0.0.0.0         255.255.255.0   U     0      0        0 eth0
87.x.x.248      0.0.0.0         255.255.255.248 U     0      0        0 eth1
```

Container Net2
Kernel IP routing table

```
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.0.1        0.0.0.0         UG    0      0        0 eth0
10.0.0.0        0.0.0.0         255.255.255.0   U     0      0        0 eth0
```

coderplay commented on Oct 17, 2016 • edited

All the tests are on AWS EC2 instances.

MacVlan

Amazon Linux

```
$ uname -a
Linux ip-172-31-24-86 4.4.19-29.55.amzn1.x86_64 #1 SMP Mon Aug 29 23:29:40 UTC 2016 x86_64 x86_64
x86_64 GNU/Linux
```

```
$ docker version
Client:
 Version:      1.12.2
 API version:  1.24
 Go version:   go1.6.3
 Git commit:   bb80604
 Built:        Tue Oct 11 05:27:08 2016
 OS/Arch:      linux/amd64
 Experimental: true

Server:
 Version:      1.12.2
 API version:  1.24
 Go version:   go1.6.3
 Git commit:   bb80604
 Built:        Tue Oct 11 05:27:08 2016
 OS/Arch:      linux/amd64
 Experimental: true

$ ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc mq state UP group default qlen 1000
    link/ether 0e:7c:03:67:8b:6a brd ff:ff:ff:ff:ff:ff
    inet 172.31.24.86/20 brd 172.31.31.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::c7c:3ff:fe67:8b6a/64 scope link
       valid_lft forever preferred_lft forever
$ docker network  create  -d macvlan --subnet=192.168.0.0/16 --ip-range=192.168.2.0/24 -o
macvlan_mode=bridge -o parent=eth0 macvlan
fa31b4f75b6985545dd86c049e1033813da68bc624d3fc27e6e83de60c4477df

$ docker run --net=macvlan -it --name macvlan_1 --rm alpine /bin/sh
docker: Error response from daemon: failed to create the macvlan port: operation not supported.
```

Ubuntu LTS 16.04

```
$ uname -a
Linux ip-172-31-25-150 4.4.0-36-generic #55-Ubuntu SMP Thu Aug 11 18:01:55 UTC 2016 x86_64 x86_64
x86_64 GNU/Linux

$ docker version
Client:
 Version:      1.12.2
 API version:  1.24
 Go version:   go1.6.3
 Git commit:   bb80604
 Built:        Tue Oct 11 17:39:06 2016
 OS/Arch:      linux/amd64
 Experimental: true

Server:
 Version:      1.12.2
 API version:  1.24
 Go version:   go1.6.3
 Git commit:   bb80604
 Built:        Tue Oct 11 17:39:06 2016
 OS/Arch:      linux/amd64
 Experimental: true

$ ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc mq state UP group default qlen 1000
    link/ether 0e:75:20:b3:d4:92 brd ff:ff:ff:ff:ff:ff
    inet 172.31.25.150/20 brd 172.31.31.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::c75:20ff:feb3:d492/64 scope link
       valid_lft forever preferred_lft forever

$ docker network  create  -d macvlan --subnet=192.168.0.0/16 --ip-range=192.168.2.0/24 -o
macvlan_mode=bridge -o parent=eth0 macvlan
8da6c27a6b57d5398cb7b6c76c9e5c9610188ae8d6bb117e46588d44499e20ff

$ docker run --net=macvlan -it --name macvlan_1 --rm alpine /bin/sh
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
c0cb142e4345: Pull complete
Digest: sha256:ca7b185775966003d38ccbd9bba822fb570766e4bb69292ac23490f36f8a742e
Status: Downloaded newer image for alpine:latest
/ # ping www.google.com
ping: bad address 'www.google.com'
/ #
```

On another host

```
$ ping 172.31.25.150
PING 172.31.25.150 (172.31.25.150) 56(84) bytes of data.
64 bytes from 172.31.25.150: icmp_seq=1 ttl=64 time=0.495 ms
64 bytes from 172.31.25.150: icmp_seq=2 ttl=64 time=0.545 ms
64 bytes from 172.31.25.150: icmp_seq=3 ttl=64 time=0.608 ms
^C
--- 172.31.25.150 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.495/0.549/0.608/0.050 ms

$ docker run --net=macvlan -it --name macvlan_2 --rm alpine /bin/sh
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
c0cb142e4345: Pull complete
Digest: sha256:ca7b18577596003d38ccbd9bba822fb570766e4bb69292ac23490f36f8a742e
Status: Downloaded newer image for alpine:latest
/ # hostname -i
192.168.3.1
/ # ping www.google.com
ping: bad address 'www.google.com'
/ # ping 192.168.2.1
PING 192.168.2.1 (192.168.2.1): 56 data bytes
^C
--- 192.168.2.1 ping statistics ---
35 packets transmitted, 0 packets received, 100% packet loss
```

IpVlan

Amazon Linux

The same hosts as MacVlan

L2

```
$ docker network  create  -d ipvlan --subnet=192.168.0.0/16 --ip-range=192.168.2.0/24 -o
ipvlan_mode=l2 -o parent=eth0 ipvlan
90d2d7b50ae9a79a9b872e6423babdda54331608a9c02991158edacf8d8febe9

$ docker run --net=ipvlan -it --name ipvlan_1 --rm alpine /bin/sh
/ # ping www.google.com
ping: bad address 'www.google.com'
```

On another host

```
$ ping 172.31.24.86
PING 172.31.24.86 (172.31.24.86) 56(84) bytes of data.
64 bytes from 172.31.24.86: icmp_seq=1 ttl=255 time=0.280 ms
64 bytes from 172.31.24.86: icmp_seq=2 ttl=255 time=0.238 ms
^C
--- 172.31.24.86 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.238/0.259/0.280/0.021 ms


$ docker network  create  -d ipvlan --subnet=192.168.0.0/16 --ip-range=192.168.3.0/24 -o
ipvlan_mode=l2 -o parent=eth0 ipvlan
372df04ea762fe021ae477457eb3aa4f012d132da298cc3afb415c4d3e7842f1

$ docker run --net=ipvlan -it --name ipvlan_2 --rm alpine /bin/sh
/ # ping 192.168.2.1
PING 192.168.2.1 (192.168.2.1): 56 data bytes
^C
--- 192.168.2.1 ping statistics ---
23 packets transmitted, 0 packets received, 100% packet loss
```

L3

replay **@yongtang** 's way, connectivity failed.

Ubuntu LTS 16.04

The same hosts as MacVlan

L2

```
$ docker network  create  -d ipvlan --subnet=192.168.0.0/16 --ip-range=192.168.2.0/24 -o
ipvlan_mode=l2 -o parent=eth0 ipvlan
03accc2292a04fd03fdccf01a5af2d8bd98896297e3c32b04a0dfb01937b1edd
```

```
$ docker run --net=ipvlan -it --name ipvlan_1 --rm alpine /bin/sh
/ # ping www.google.com
ping: bad address 'www.google.com'
/ #
```

Another host

```
$  ping 172.31.25.150
PING 172.31.25.150 (172.31.25.150) 56(84) bytes of data.
64 bytes from 172.31.25.150: icmp_seq=1 ttl=64 time=0.598 ms
64 bytes from 172.31.25.150: icmp_seq=2 ttl=64 time=0.732 ms
^C
--- 172.31.25.150 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.598/0.665/0.732/0.067 ms

$ docker network  create  -d ipvlan --subnet=192.168.0.0/16 --ip-range=192.168.3.0/24 -o
ipvlan_mode=l2 -o parent=eth0 ipvlan
49f1d7a9cfeb339d3974eb1294c92e48f8a0f1d6f2ffc6ebba5ffaaae4a943f2

$ docker run --net=ipvlan -it --name ipvlan_2 --rm alpine /bin/sh
/ # ping 192.168.2.1
PING 192.168.2.1 (192.168.2.1): 56 data bytes
^C
--- 192.168.2.1 ping statistics ---
2 packets transmitted, 0 packets received, 100% packet loss
```

L3

replay **@yongtang** 's way, connectivity failed.

Conclusion

None of the above tests succeed. I also tested with `nc` instead of `ping`, failed.

thaJeztah assigned **mrjana**, **thaJeztah** and **mavenugo** on Nov 8, 2016

thaJeztah removed their assignment on Nov 8, 2016

erandu commented on Dec 9, 2016 • edited

I have tested the Macvlan driver, and I am interested by using this one. But I need to use the Macvlan driver on a multi-host environment, and the problem is that the IPAM for Ipvlan/Macvlan driver is currently just local (per-host).
The issue have already been discussed there : docker/libnetwork#1150

Do you have any news about a global IPAM driver ? **@mavenugo**

erandu unassigned **mrjana** on Dec 9, 2016

johnbelamaric commented on Dec 9, 2016

@erandu What IPAM driver are you thinking of? Our (infoblox) IPAM driver can handle this situation, as you can configure the local and global address spaces to come from the same pool if you wish. But of course you need Infoblox to make it work...

erandu commented on Dec 9, 2016

I am thinking about the Docker default IPAM driver. Unfortunately, using an externa system like Infoblox DDI is not possible for me.

dreamcat4 commented on Dec 9, 2016

Quick question: by any chance does such infobox driver also let same docker host itself see the containertraffic? It would save me a lot of trouble / hassles.

johnbelamaric commented on Dec 10, 2016

@dreamcat4 this is an IPAM only driver. So, it's not really directly related to that. If you are using macvlan you I expect you could do what you want by putting a host interface on the intended VLAN. If you want to see all traffic for all VLANs I expect it is possible, but would need to understand your configuration and use case to give a better answer. For example, what do you mean by "see" - with tcpdump or do you expect to have something receiving and processing the packets normally?

dreamcat4 commented on Dec 10, 2016

So still 2nd hardware nic then? The use case is basically not having enough computers or nics lying about. For example this laptop is serving double duty as a nas server + other home server / microserver + desktop machine. I need to be able to use the nas services in my containers and this machine only has 1 hardware nic.

johnbelamaric commented on Dec 10, 2016

No, I think you could do it with a virtual interface. There is usually a way but the devil is in the details.

mavenugo commented on Dec 10, 2016 Contributor

@dreamcat4 Do you think your question is related to macvlan/ipvlan drivers ? If yes, can you pls elaborate ?

mavenugo commented on Dec 10, 2016 Contributor

@erandu could you please explain your use-case better ? We are thinking about node-local network support for swarm. Though technically it seems useful, I cannot think of a valid use-case that covers and satisfies all areas of swarm features (Multi-Host Service Discovery and Load-balancing for example). At the same time, converting MacVlan driver as a multi-host driver comes with its own limitations and we are weighing the options. Am also thinking of any non-native way of supporting this use-case that will help address the use-cases without having to depend on changing the mac-vlan driver or swarm architecture.

dreamcat4 commented on Dec 11, 2016

Well what I was asking was more along the lines of user work-arounds. I really don't wish to repeat myself (why copy-paste the same comments from yesterday?) The reasons for my initial inquiry are already clarified / explained in a follow uip comment. And in fact a kind person here has already provided a partial answer. Which was:

> No, I think you could do it with a virtual interface. There is usually a way but the devil is in the details.

Both helpful and hopeful. So thank you.

Now **@mavenugo** if you have an opinion to weigh in on this, then why not? Either you don't think this should be done as some hack, but instead done more properly / formally within the official macvlan docker driver or else higher up at the kernel level. Otherwise you think the opposite, whatever that may be. Or maybe you guys already had some form of opportunity to affect such decision already taken at an earlier time during development. But decided to go the other way for whatever reason.

Honestly I'm more interested in user workarounds at this point. Unless there is some magical change in kernel security poliicy. A comment I've read implies that it is not possible to offer such an option at the kernel level for macvlan. Its in some patch or changelog note / commit message. Sorry I can't cite the exact source where it was originally first mentioned. And in docker project its been explained that you guys just pass to the kernel.... hence user workarounds.

erandu commented on Dec 12, 2016 • edited

@mavenugo I don't use Docker swarm, I use fleet with Docker, on multiple coreOS nodes.
So what I expected is something like having the Macvlan driver with a multi-host IPAM driver.

mavenugo commented on Dec 12, 2016 Contributor

@dreamcat4 am really trying to understand the use-case since it is not clear to me. I guess you are referring to `docker host itself see the container traffic`. So the next comment on the use-case `I need to be able to use the nas services in my containers and this machine only has 1 hardware nic`. Since Macvlan driver prevents such host access, the only other option that I can think of is to use macvlan driver to get the static IP for the container and connect the container to another bridge driver so that you can access the container both from external services via the static ip (via macvlan driver) and internally from the host using the IP assigned to that container in bridge network.

Will that satisfy your use-case ?

mavenugo commented on Dec 12, 2016 Contributor

@erandu okay. I understand the use-case better. In the current multi-host docker networking implementation, we require the driver to be global-scoped. Global-scoped drivers typically requires external kv-store (prior to docker 1.12) or swarm-mode (1.12+). Also the global-scoped network driver goes hand-in-hand with global-scoped ipam driver as well. That is the reason I raised docker#27082 to support local-scoped drivers in swarm-mode to ease such deployments.

But your case is a bit different and you are looking for a global-scoped Macvlan driver. There is a lot of details & limitations to such a configuration and hence my question on the use-cases.

erandu commented on Dec 13, 2016

@mavenugo I also forgot to mention that I already use etcd as a k/v store for other applications. This why this is not a problem for me.
Could you explain some of these details and limitations ?

yongtang commented on Dec 19, 2016 Member

@coderplay Sorry I didn't notice your comment with respect to the ipvlan issue on AWS before. The Src/Dest Check has to be disabled on your instances in order for ipvlan to work on AWS, as the ipvlan will send the packet out with container's IP, not your instance's IP.

The Src/Dest can only be disabled by modifying the attribute of the instances (after the instances have been created). You could either disable Src/Dest Check on AWS Console, or with `aws ec2 modify-instance-attribute`

abhishekes commented on Dec 28, 2016 • edited

@yongtang I had earlier tried out ipvlan as you had recommended in your comment earlier (tested on AWS) and found that they worked as you described - I was able to ping remote containers by adding routes. However, when I am trying it now, I see that connectivity to the host from outside is breaking as soon as I start the container. Stopping the container restores connectivity to host.

I don't find anything wrong with the routing when the container is running.

Only difference is probably that I was using Ubuntu 14.04 earlier and now I am using 16.04. It seems like after starting the container, all the traffic is getting bridged directly to the container and hence the host is not seeing any traffic.

Is there something I could be missing here. I hope specifying --parent=eth0 does not make the eth0 network exclusive the containers attached to that network.

Update: OK, this might be a possible bug here. I created an additional container on the same network and deleted it and this restored connectivity to the host.

This is currently getting reproducible consistently.

1. Create a ipvlan network with parent as eth0
2. Start a new container on the above network -> Connectivity to host from outside is broken.
3. Start another container on the same network -> Connectivity to host from outside is still broken.
4. Stop container created in either Step 2 or Step 3 -> Connectivity to host is restored.
   The remaining container on the ipvlan network also works.

chinakevinguo commented on Jan 6, 2017 • edited

container on macvlan can not connect to gateway ?

Docker Host

```
[root@node01 ~]# uname -a
Linux node01 3.10.0-327.el7.x86_64 #1 SMP Thu Nov 19 22:10:57 UTC 2015 x86_64 x86_64 x86_64
GNU/Linux

[root@node01 ~]# ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 08:00:27:5a:e9:e7 brd ff:ff:ff:ff:ff:ff
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 08:00:27:5f:be:76 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.1/24 brd 192.168.1.255 scope global enp0s8
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe5f:be76/64 scope link
       valid_lft forever preferred_lft forever
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN
    link/ether 02:42:39:72:23:80 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:39ff:fe72:2380/64 scope link
       valid_lft forever preferred_lft forever


[root@node01 ~]# docker info
Containers: 2
 Running: 2
 Paused: 0
 Stopped: 0
Images: 1
Server Version: 1.12.5
Storage Driver: devicemapper
 Pool Name: docker-253:0-9781-pool
 Pool Blocksize: 65.54 kB
 Base Device Size: 10.74 GB
 Backing Filesystem: xfs
 Data file: /dev/loop0
 Metadata file: /dev/loop1
 Data Space Used: 32.64 MB
 Data Space Total: 107.4 GB
 Data Space Available: 39.09 GB
 Metadata Space Used: 634.9 kB
 Metadata Space Total: 2.147 GB
 Metadata Space Available: 2.147 GB
 Thin Pool Minimum Free Space: 10.74 GB
 Udev Sync Supported: true
 Deferred Removal Enabled: false
 Deferred Deletion Enabled: false
 Deferred Deleted Device Count: 0
 Data loop file: /var/lib/docker/devicemapper/devicemapper/data
 Metadata loop file: /var/lib/docker/devicemapper/devicemapper/metadata
 Library Version: 1.02.107-RHEL7 (2015-10-14)
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: bridge macvlan host null overlay
Swarm: inactive
Runtimes: runc
Default Runtime: runc
Security Options: seccomp
Kernel Version: 3.10.0-327.el7.x86_64
Operating System: CentOS Linux 7 (Core)
OSType: linux
Architecture: x86_64
CPUs: 1
Total Memory: 489 MiB
Name: node01
ID: OBMQ:CHY6:DO2Z:Y3W3:4QLW:FC6D:SOOA:PLJB:YR2L:UB4S:KO4C:PV56
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): false
Registry: https://index.docker.io/v1/
WARNING: bridge-nf-call-iptables is disabled
```

```
WARNING: bridge-nf-call-ip6tables is disabled
Insecure Registries:
 127.0.0.0/8

[root@node01 ~]# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
143ee5ac2254        bridge              bridge              local
51e08f4e7c23        host                host                local
133d171ded8e        macvlan_pub         macvlan             local
728d0f0e524d        none                null                local

[root@node01 ~]# docker network inspect macvlan_pub
[
    {
        "Name": "macvlan_pub",
        "Id": "133d171ded8ed55ff146755590817a6594b2219a4ce92431a4a9c2350ada7514",
        "Scope": "local",
        "Driver": "macvlan",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "192.168.1.0/24",
                    "Gateway": "192.168.1.100"
                }
            ]
        },
        "Internal": false,
        "Containers": {
            "68f48d6a9927896490a7efe48e62751467e84d54d810c220fde984055c1db443": {
                "Name": "peaceful_hamilton",
                "EndpointID": "a5e6a274c1b71c17a38a59781784517008f5f8da78edb734041b6744472eb8df",
                "MacAddress": "02:42:c0:a8:01:ca",
                "IPv4Address": "192.168.1.202/24",
                "IPv6Address": ""
            },
            "a1fef7565c233153f010069607d1e17c944648636407a13e4dff1120d1e2320f": {
                "Name": "desperate_raman",
                "EndpointID": "c6d76075da6a558ed235460c7af5448cac2e71aafe9c969766a8aed3b925e674",
                "MacAddress": "02:42:c0:a8:01:c9",
                "IPv4Address": "192.168.1.201/24",
                "IPv6Address": ""
            }
        },
        "Options": {
            "parent": "enp0s8"
        },
        "Labels": {}
    }
]
```

route host

```
[root@my-keystore ~]# ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 08:00:27:5a:e9:e7 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic enp0s3
       valid_lft 78858sec preferred_lft 78858sec
    inet6 fe80::a00:27ff:fe5a:e9e7/64 scope link
       valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 08:00:27:ac:4f:00 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.100/24 brd 192.168.1.255 scope global enp0s8
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:feac:4f00/64 scope link
       valid_lft forever preferred_lft forever

[root@my-keystore ~]# iptables -t nat -L -n
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

```
Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
SNAT       all  --  192.168.1.0/24       0.0.0.0/0            to:10.0.2.15
```

steps

```
docker network create -d macvlan --subnet 192.168.1.0/24 --gateway 192.168.1.100 -o parent=enp0s8
macvlan_pub

docker run --net macvlan_pub --ip 192.168.1.201 -tid alpine /bin/sh

docker run --net macvlan_pub --ip 192.168.1.202 -tid alpine /bin/sh
```

I can ping the External network on docker host

```
# ping github.com
PING github.com (192.30.253.112) 56(84) bytes of data.
64 bytes from 192.30.253.112: icmp_seq=1 ttl=43 time=247 ms
```

check route table in container

```
[root@node01 ~]# docker exec -ti peaceful_hamilton route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.1.100   0.0.0.0         UG    0      0        0 eth0
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
```

check in another container

```
[root@node01 ~]# docker exec -ti desperate_raman route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.1.100   0.0.0.0         UG    0      0        0 eth0
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
```

Now test ping

```
[root@node01 ~]# docker exec -ti peaceful_hamilton ping 192.168.1.201
PING 192.168.1.201 (192.168.1.201): 56 data bytes
64 bytes from 192.168.1.201: seq=0 ttl=64 time=0.049 ms

[root@node01 ~]# docker exec -ti desperate_raman ping 192.168.1.201
PING 192.168.1.201 (192.168.1.201): 56 data bytes
64 bytes from 192.168.1.201: seq=0 ttl=64 time=0.048 ms
```

container can ping each other,but ,container can not ping the gateway 192.168.1.100

```
[root@node01 ~]# docker exec -ti desperate_raman ping 192.168.1.100
PING 192.168.1.100 (192.168.1.100): 56 data bytes
^C
```

additional:
All my actions are executed on the virtual machine created through vagrant in virtualbox

anyone can help me ?

lawre commented on Jan 26, 2017

We have discovered a change in behavior (or a bug) with the 4.X linux kernel that was not present with 3.X.

When creating a macvlan in linux kernel 4 the parent interface's MAC address will also be changed. This may explain why some of you are getting "disconnected" when setting up a macvlan. when your eth0 MAC address changes this may trigger promiscuous rules with your switch and drop frames with the new MAC address (I can confirm this happens with ESXi virtual switches... just change the promiscuous rule to "accept").

Steps to reproduce:

- We used CentOS 7.2 with kernel version 4.9.5 from https://www.elrepo.org

- take note of the MAC address of the parent interface (eth0 in our case)
- Create a macvlan and define a MAC address
- Look at the parent interfaces's MAC address. It will have changed to match the macvlan's MAC.

For my use case, this is game breaking. Since the macvlan only seems to affect its parent, we are able to workaround this issue by creating a bridge interface between the eth0 and macvlan interfaces to take the MAC change.

Can anyone else confirm this behavior on the 4.x kernel?

1

kenzodeluxe commented on Jan 30, 2017

> anyone can help me ?

I would like to understand this better as well. I have been playing with macvlan and do not understand the limitation of not being able to connect to a gateway if it's locally available on the host itself. What is the reason for this "manual limitation"?
From a use case point of view, if I use a macvlan based network on various hosts, they'll all be able to use a dedicated gateway that's part of the VLAN. However, if I wanted to run another container on the same gateway node, it won't be able to utilize the same configuration.
Can someone give some insight on the underlying reasons for that?

tgpfeiffer commented on Feb 6, 2017

I am trying to use the macvlan network driver to add my hosts to the physical network so that they can host services such as Zookeeper etc. I play with the driver in a VirtualBox setup like so:

- on host1 (10.0.2.15@eth0)
  - `docker network create -d macvlan --subnet 10.0.2.0/24 --ip-range 10.0.2.96/28 --gateway 10.0.2.2 -o parent=eth0 pub_net`
  - `docker run --rm --net=pub_net -it alpine /bin/sh`
- on host2 (10.0.2.17@enp0s3)
  - `docker network create -d macvlan --subnet 10.0.2.0/24 --ip-range 10.0.2.112/28 --gateway 10.0.2.2 -o parent=enp0s3 pub_net`
  - `docker run --rm --net=pub_net -it alpine /bin/sh`

Now from both running containers,

- I can ping the outside internet (8.8.8.8)
- I can ping the gateway (10.0.2.2)
- (I can not ping the host interface as intended)
- the containers cannot ping each other (timeout)
- the containers cannot even ping the other container host (timeout)

Now I am wondering if there is any firewall rule added by Docker that filters incoming packages or is there any configuration missing on the hosts to bridge the packets to the containers? Isn't there any configuration necessary to tell the attached network interface to forward packets?

tgpfeiffer commented on Feb 6, 2017

I resolved my issue above, the solution was to change the VirtualBox network adapter to "PCnet-FAST III" and "Promiscuous mode", then I can ping containers on other nodes.

kkapoor29 commented on Feb 25, 2017

Hi,

If we need to inspect the traffic of macvlan containers what would be the way to tap that traffic ?

chicco785 commented on Mar 21, 2017

@mavenugo is there any progress either toward making global-scope the ipvlan driver, or making swarm support local scope drivers?
we are working on VNF deployment over docker, and usage of overlay networks is not a good option since it degrade performance quite a lot. Of course we could manage them per single host, but then we will loose all the advantages coming from having swarm managing the cluster.

praparn commented on Aug 19, 2017 • edited by thaJeztah

Dear All,

Greeting from thailand. We are try to initial lab for swarm mac-vlan on google cloud by 3 node of swarm like detail below:

```
========================================================================
#Compute Farm#
NAME         ZONE          MACHINE_TYPE   PREEMPTIBLE  INTERNAL_IP     EXTERNAL_IP     STATUS
swarm-mng    asia-east1-a  n1-standard-1               192.168.99.200  35.194.247.60   RUNNING
swarm-node1  asia-east1-a  n1-standard-1               192.168.99.201  35.194.144.119  RUNNING
swarm-node2  asia-east1-a  n1-standard-1               192.168.99.202  35.187.158.143  RUNNING


#Create MACVLAN on each local machine#
#swarm-mng#
#ip range: (192.168.99.129 - 192.168.99.158)#
docker network create --config-only --subnet 192.168.99.0/24 \
 --gateway 192.168.99.1 -o parent=ens4 \
--ip-range 192.168.99.129/27 macvlannet

#swarm-node1#
#ip range: (192.168.99.161 - 192.168.99.190)#
docker network create --config-only --subnet 192.168.99.0/24 \
--gateway 192.168.99.1 -o parent=ens4 \
--ip-range 192.168.99.161/27 macvlannet

#swarm-node2#
#ip range: (192.168.99.193 - 192.168.99.222)#
docker network create --config-only --subnet 192.168.99.0/24 \
--gateway 192.168.99.1 -o parent=ens4 \
--ip-range 192.168.99.193/27 macvlannet


#Create Global MACVLAN on Swarm#
#swarm-mng#
docker network create -d macvlan --scope swarm --config-from macvlannet swarm-macvlan
docker network ls

#Result#
NETWORK ID          NAME                DRIVER              SCOPE
9ba217a5bb8f        bridge              bridge              local
63a8a77b28df        docker_gwbridge     bridge              local
810ea4ddbf72        host                host                local
bxch17e0ojmy        ingress             overlay             swarm
a320770f9521        macvlannet          null                local
4e51be11f04e        none                null                local
xfjw25nw01q2        swarm-macvlan       macvlan             swarm
========================================================================
```

After finished to create macvlan on swarm. We try to create 2 simple alpine linux (with small web on port 3000) service for attach with "swarm-macvlan" and check ip address of them as below:

```
#swarm-mng#
#create simple alpine service#
docker service create -dt --name alpine1 \
--replicas 1 --network swarm-macvlan \
labdocker/alpineweb:latest node hello.js

docker service create -dt --name alpine2 \
--replicas 1 --network swarm-macvlan \
labdocker/alpineweb:latest node hello.js

#check location of service#
docker service ls
ID                          NAME                MODE                REPLICAS            IMAGE
PORTS
oaxk3ro3pet0        alpine2             replicated          1/1
labdocker/alpineweb:latest
qi0c1t6anhtr          alpine1             replicated          1/1
labdocker/alpineweb:latest

docker service ps alpine1
```

```
ID                  NAME                IMAGE                         NODE                DESIRED
STATE       CURRENT STATE            ERROR               PORTS
ml2dgppfj3y9        alpine1.1           labdocker/alpineweb:latest    swarm-node1         Running

docker service ps alpine2
ID                  NAME                IMAGE                         NODE                DESIRED
STATE       CURRENT STATE            ERROR               PORTS
hpj009ex200q        alpine2.1           labdocker/alpineweb:latest    swarm-node2         Running


#check ip address on service alpine1#
praparn@swarm-node1:~$ docker inspect $(docker ps -q)| grep IPAddress
            "SecondaryIPAddresses": null,
            "IPAddress": "",
                    "IPAddress": "192.168.99.160",

#check ip address on service alpine2#
docker inspect $(docker ps -q)| grep IPAddress
            "SecondaryIPAddresses": null,
            "IPAddress": "",
                    "IPAddress": "192.168.99.192",


================================================================================
```

So we are try to check connect across host via macvlan and inside connecter test ping/curl to other host/gateway in every test case it cannot operate:

```
#swarm-mng ==> alpine 1 (on swarm-node1), alpine2 (on swarm-node2):
192.168.99.200 ==> 192.168.99.160
192.168.99.200 ==> 192.168.99.192
Result: request timeout / no arp / curl connection timeout

#alpine 1 (on swarm-node1) ==> gateway 192.168.99.100, swarm-mng
192.168.99.160 ==> 192.168.99.1
192.168.99.160 ==> 192.168.99.200
Result: request timeout / no arp

#alpine 2 (on swarm-node2) ==> gateway 192.168.99.100
192.168.99.192 ==> 192.168.99.1
192.168.99.192 ==> 192.168.99.200
Result: request timeout / no arp

#alpine 1 (on swarm-node1) ==> #alpine 2 (on swarm-node2)
192.168.99.160 ==> 192.168.99.192
Result: request timeout / no arp / curl connection timeout

#alpine 2 (on swarm-node2) ==> alpine 1 (on swarm-node1)
192.168.99.192 ==> 192.168.99.160
Result: request timeout / no arp / curl connection timeout

================================================================================
```

After this problem we try to test connect of container inside same host by scale service from 1 = 6 on both alpine1,alpine2

```
#swarm-node1
docker inspect $(docker ps -q)| grep IPAddress
            "SecondaryIPAddresses": null,
            "IPAddress": "",
                    "IPAddress": "192.168.99.162",
            "SecondaryIPAddresses": null,
            "IPAddress": "",
                    "IPAddress": "192.168.99.163",
            "SecondaryIPAddresses": null,
            "IPAddress": "",
                    "IPAddress": "192.168.99.161",
            "SecondaryIPAddresses": null,
            "IPAddress": "",
                    "IPAddress": "192.168.99.160",

#swarm-node2
docker inspect $(docker ps -q)| grep IPAddress
            "SecondaryIPAddresses": null,
            "IPAddress": "",
                    "IPAddress": "192.168.99.195",
            "SecondaryIPAddresses": null,
            "IPAddress": "",
                    "IPAddress": "192.168.99.193",
            "SecondaryIPAddresses": null,
            "IPAddress": "",
                    "IPAddress": "192.168.99.194",
```

```
            "SecondaryIPAddresses": null,
            "IPAddress": "",
                    "IPAddress": "192.168.99.192",

#Test ping inside host
#192.168.99.192 ==> 192.168.99.195
Result: successful

#192.168.99.162 ==> 192.168.99.161
Result: successful
```

From result lab above we found connect can establish on same host only and cannot access to outside. Could you please suggestion or any thing we need to configure on iptable host ?

```
========================================================================
Remark:
#OS information
#uname -a (all node with same version)
Linux swarm-mng 4.8.0-56-generic #61~16.04.1-Ubuntu SMP Wed Jun 14 11:58:22 UTC 2017 x86_64
x86_64 x86_64 GNU/Linux

#promisc mode (all node)
ip link set dev ens4 promisc on

#docker info (all node with same version)
docker info
Containers: 4
 Running: 4
 Paused: 0
 Stopped: 0
Images: 1
Server Version: 17.06.1-ce
Storage Driver: overlay2
 Backing Filesystem: extfs
 Supports d_type: true
 Native Overlay Diff: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: bridge host macvlan null overlay
 Log: awslogs fluentd gcplogs gelf journald json-file logentries splunk syslog
Swarm: active
 NodeID: d1mtchl37k02njhl6fgsv4eai
 Is Manager: true
 ClusterID: 4qpj9lyh4zj4k6jjv4dxn1b8h
 Managers: 1
 Nodes: 3
 Orchestration:
  Task History Retention Limit: 5
 Raft:
  Snapshot Interval: 10000
  Number of Old Snapshots to Retain: 0
  Heartbeat Tick: 1
  Election Tick: 3
 Dispatcher:
  Heartbeat Period: 5 seconds
 CA Configuration:
  Expiry Duration: 3 months
  Force Rotate: 0
 Root Rotation In Progress: false
 Node Address: 192.168.99.200
 Manager Addresses:
  192.168.99.200:2377
Runtimes: runc
Default Runtime: runc
Init Binary: docker-init
containerd version: 6e23458c129b551d5c9871e5174f6b1b7f6d1170
runc version: 810190ceaa507aa2727d7ae6f4790c76ec150bd2
init version: 949e6fa
Security Options:
 apparmor
 seccomp
  Profile: default
Kernel Version: 4.8.0-56-generic
Operating System: Ubuntu 16.04.2 LTS
OSType: linux
Architecture: x86_64
CPUs: 1
Total Memory: 3.613GiB
Name: swarm-mng
ID: LCJX:TLXM:B3GR:RCKE:3HXU:EP3J:RZJY:2K2Y:M7PC:DMDF:3C3P:GXVB
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): false
```

```
Registry: https://index.docker.io/v1/
Experimental: false
Insecure Registries:
 127.0.0.0/8
Live Restore Enabled: false
```

Br,
Praparn L

djmaze commented on Oct 11, 2017 Contributor

At least with docker 17.09.0, I discovered swarm will overwrite a macvlan's subnet with its own (default?) configuration.

```
$ docker network create -d macvlan --scope swarm -o parent=eth0.50 --subnet=192.168.50.0/24 --gatewa
$ docker network inspect dmz | grep Subnet
                    "Subnet": "192.168.50.0/24",
$ docker service create --name debug --network dmz busybox sleep 10000
# (.. wait until container is running ..)
$ docker network inspect dmz | grep Subnet
                    "Subnet": "192.168.0.0/20",
$ docker service rm debug
# (.. wait until container is stopped ..)
$ docker network inspect dmz | grep Subnet
                    "Subnet": "192.168.50.0/24",
```

Is this how it is supposed to work?

WyseNynja commented on Feb 2

Has anyone had success getting the host and guest able to communicate? I know this is blocked by default, but I need my host able to get to one of my containers. The docs have some steps for this, but they seem incomplete.

More info at docker/docker.github.io#5899

Antauri commented on Feb 5 • edited

Hello guys. Finally I got the time to give some feedback here. We've deployed IPVLAN on 10.x/16 network for the better of 2 months now. We've done performance tests against it and reliability tests for 2 weeks (constantly having 2 stacks communicate a few GBs between them and monitoring availability, speed, error rates) before actually committing to using it on a daily basis.

We took "the leap of faith jump" and put it production (with a fallback method on host networking with manually added IP addresses before each container is started). I'd love if this moved out of the "experimental" feature flag anytime soon 👍

At first, we had the same issues of pinging the host or from the host to container, but since we've configured both the physical machines and the IPVLAN on the machines on the 10.x/16 subnet and using an "ipvlan" virtual device, we're able to resolve the host/guest communicate/ping each other. Basically, the host has an "ipvlan" virtual device and the host and Docker networks share the same "10.x/16" subnet.

The work-around is covered in the last paragraphs of the IPVLAN white-paper, you also need to add the host IP to the "ipvlan" device and if you "overlap" the physical and container subnets (sharing it) you will get connectivity between the host and containers without issues as both worlds (physical hosts and containers) will share the same network subnet and default gateway. We are in an "eth1/2" to bond0 to vlan to ipvlan hierarchy (quite complex hierarchy to be honest).

```
.... a typical host network (ip addr show)

8: ipvlan0@vlan123: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc noqueue state UP group
default qlen 1000
    link/ether 24:6e:96:55:c2:e8 brd ff:ff:ff:ff:ff:ff
    inet 10.x.y.z/16 scope global ipvlan0
       valid_lft forever preferred_lft forever

... docker network created with the same subnet.
... configured to use the phyiscal host gateway.
... connectivity achieved both ways (host to containers, containers to host).
```

- Note that hosts are configured with 10.x.y.z subnets and 10.x.y.1 is the gateway configured for both hosts and the IPVLAN external network in Docker. So both containers and hosts share the same default gateway.

We keep track of which are IPs for containers and which are for hosts, through a simple "reserve" procedure (by using Netbox from Digital Ocean) with some pre-deployment tests (duplicate address detection) to avoid conflicts, because the IPVLAN implementation is "host-scoped" not network scoped, so it may assign duplicate IP addresses to 2 different containers.

In the past 2 months there are no issues to report in terms of connectivity/stability/availability.

In the good, old and "moby" repository tradition, a picture of a nice, cute, animal:

dbloms commented on Feb 9

I also vote for moved out of the "experimental" feature flag. We used it for several weeks and have had no issues on debian 9 and docker-ce 18.01.

praparn commented on Feb 11 • edited

@dbloms Totally agree with that