# FanNetworking

**Ubuntu's Fan Networking: User Documentation**

## Introduction

Containers enable tremendously dense virtualisation -- it is easy to run hundreds or even thousands of containers on a single host machine. Whether for whole machine containers (LXD) or process containers (Docker), it is easiest for these containers to be managed as separate networking entities, which means they need their own IP addresses. The proliferation of containers thus creates demand for additional network address space. Typically, the number of extra addresses needed is roughly constant across each container host.

A new solution to this problem of increased demand for IP address space is the fan. The fan is a mapping between a smaller network address space (typically a /16 network) and a larger one (typically a /8), which assigns subnets from the larger one to IP addresses on the smaller one, and enables automatic and simple tunnelling and routing between systems on the larger address space. Fan addresses are assigned as subnets on a virtual bridge on the host, which are mathematically related to the primary (or *underlay*) IP address behind which they are mapped. The fan system can be considered "address expansion," as it simply multiplies the number of available IP addresses on the host, providing an extra 253 usable addresses for each host IP address on the /16.

This document describes the implementation of the fan on Ubuntu, beginning with the principles on which it's based. Next up are outlines of some common use cases for the fan. The user-mode fan tools are then described, along with how to create a persistent configuration. Finally, detailed configuration instructions for using the fan with both LXC/LXD and Docker are presented.

## Fan Principles

Consider the case of VPCs on EC2:

- EC2 VPCs allow you to designate the /16 address space you wish to use. This document adopts the 172.16.0.0/16 network as an "underlay" network.

- EC2 by default allows you to have 5 VPCs, so it would be normal to have 172.16.0.0/16, 172.17.0.0/16, etc.

That in turn means that address expansion to anywhere in the 172.0.0.0 range is problematic, as it might cause parts of your VPC to be unable to route to other VPCs in your organisation.

These are highly specialised assumptions that are not always true of traditional network infrastructures, but they are true in many of today's public cloud infrastructures, which is where we believe the fan overlay is going to be extremely valuable as a bridging story to IPv6. The following sections elaborate on how the fan can help you get the most out of the address space provided by such a set of addresses.

## How it Works

Fan trades access to one user-selected /8 (potentially external) address range for an expanded pool of "organisation-internal" addresses to be used by containers or virtual machines. Fan does this by mapping the addresses in a way that can be computed, rather than one that requires maintenance of distributed state (e.g., routing tables).

### Trading IP Address Ranges

We trade the ability to route to a /8 network, such as 10.0.0.0/8, for an additional 253 IP addresses accessible behind every local IP address. This is suitable for container environments where it avoids the need to track a database of arbitrary overlay addresses for each container mapped to host addresses, and simplifies routing because a single fan route accounts for the entire system on each host.

In other words, for each local underlay IP address you might have on a machine, say 172.16.3.4, you gain an additional 253 overlay IP addresses that can be used by containers on that machine, each within the single /8 selected. This /8 network cannot be used for its original purpose inside the fan. Figure 1 presents a high-level overview of the fan.

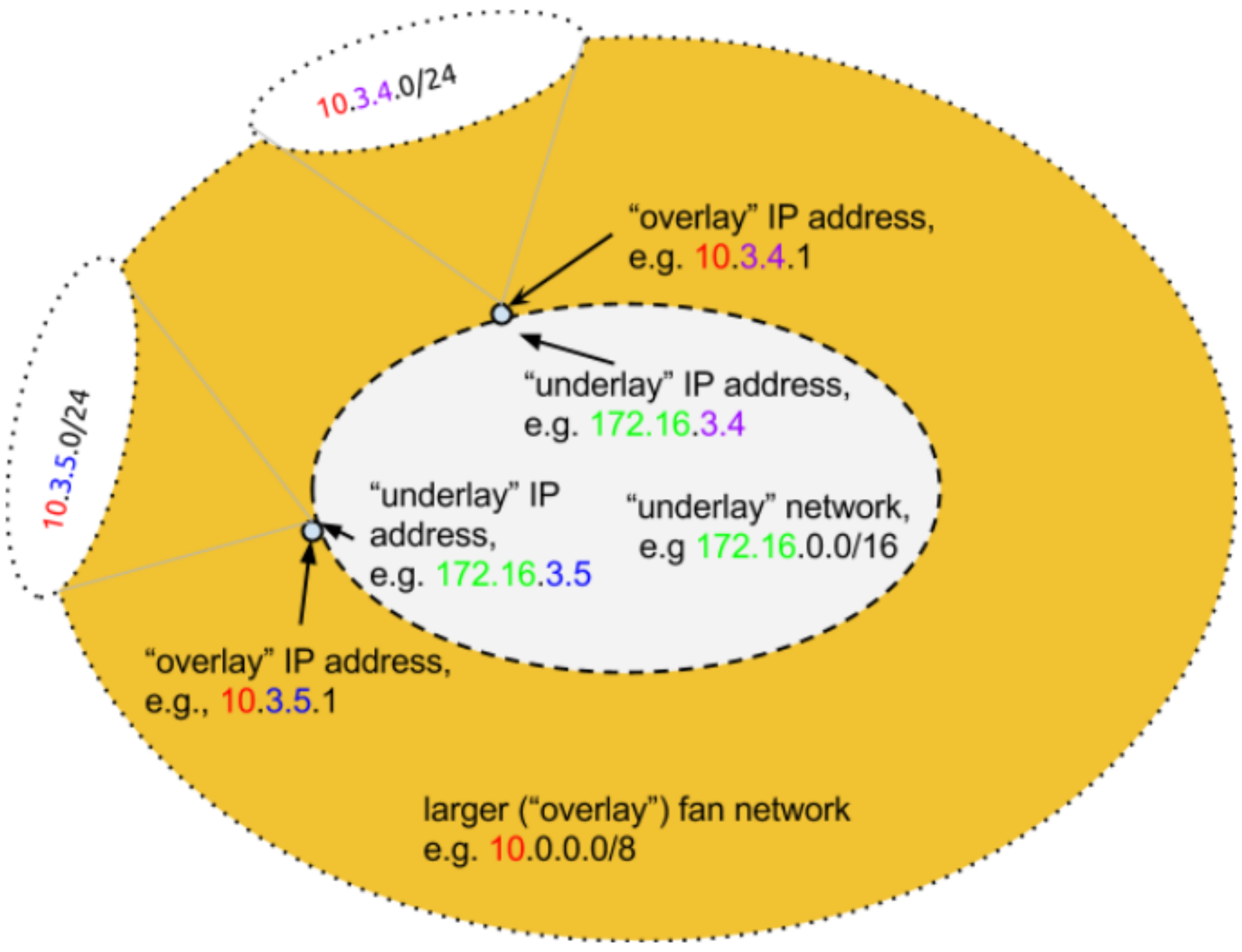
10.3.4.0/24
10.3.5.0/24
"overlay" IP address, e.g. 10.3.4.1
"underlay" IP address, e.g. 172.16.3.4
"underlay" IP address, e.g. 172.16.3.5
"underlay" network, e.g 172.16.0.0/16
"overlay" IP address, e.g., 10.3.5.1
larger ("overlay") fan network e.g. 10.0.0.0/8

**Figure 1: The fan uses IP addresses on the underlay network that bridge to larger overlay networks.**

A set of underlay IP addresses can see one another on the existing local network. There may be as many of these as you like from your standard 172.16.0.0/16 VPC address space (a maximum of 65,534 addresses). These may be divided into whatever structure of subnets you like, with whatever routing and firewalling you desire between those subnets. They may also be allocated to virtual machines in whatever pattern you like -- you might have a single network interface per machine with one or more local addresses, or even multiple network interfaces per machine, as long as all of those interfaces have addresses on the 172.16.0.0 space.

Each machine running the fan overlay system now gains a set of virtual network bridges, one for each IP address on the machine from the VPC.

Thus, a machine with two network interfaces that has a total of two IP addresses would gain two virtual bridges internally. Each of those virtual bridges would have a /24 subnet routed to it, onto which you can connect container network interfaces and allocate addresses from that /24 subnet.

The net effect is that each single address on the /16 network gains an additional 253 IP addresses on its corresponding host-specific virtual network bridge. This is where we get the 253-fold multiplier of address space.

Addresses created on the fan can reach the Internet (or other private addresses) by use of NAT from their host. Because they are behind NAT on their host, they themselves cannot be reached directly from non-fan addresses unless special port mapping arrangements have been made. Generally, it is easiest for containers on the same fan to talk to one another.

### Address Mapping

The cunning part of the fan is in the way these host-specific virtual bridge subnets are allocated. We map from the /16 local address space into a larger /8 address space in order to create the illusion of a larger but still "flat" address space for these container IP addresses.

We need to know two things:

- The specific local address space being used for the underlay address space on the local network. On a VPC this could be 172.16.0.0/16. You want to know the first two octets (172.16, in this example) of these addresses. This is the underlay address.
- The /8 network to which you do not care to route from this VPC at all, such as 10.0.0.0/8. As this /8 network is utilized for the overlay, if the selected /8 is publicly routed on the Internet, you will lose all connectivity to that /8 network address space from all hosts participating in the fan.

In this example, the address mapping is simply:

```
172.16.3.4 -> 10.3.4.0/24
```

In other words, the machine which has an IP address 172.16.3.4 will also have a private fan bridge called fan-10-3-4 on which the subnet 10.3.4.0/24 is hosted.

Behind the scenes, the fan mapping device encapsulates any traffic routed through it and addresses the outer packet to the appropriate underlay IP address. For instance, suppose a process or container on 172.16.5.6 wanted to communicate with a container that announced its address as 10.3.4.25. The fan-10 would encapsulate that packet and address it to the appropriate underlay address, which is 172.16.3.4. When the packet arrives at 172.16.3.4, it will be unwrapped, leaving the packet addressed to

10.3.4.25. The fan-10-3-4 device on that system will be hosting 10.3.4.0/24 and so the packet is neatly delivered to that bridge and hence the container on the bridge. This arrangement is illustrated in Figure 2. Of course, the containers need not be on separate hosts, but direct communication does not require encapsulating their traffic.

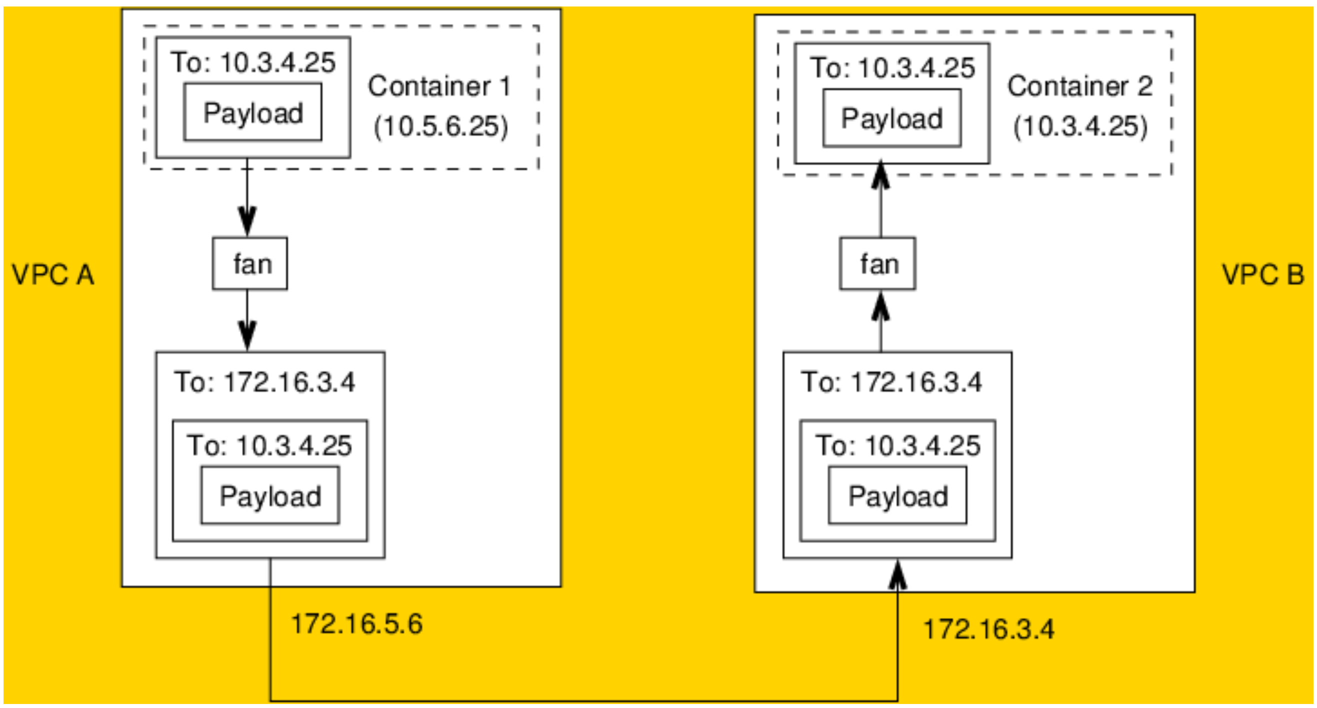

**Figure 2: The fan uses network encapsulation and computable rules for selecting addresses to deliver packets appropriately.**

## Fan Design Goals

A pair of overall design goals apply to the fan: It should be transparent to its consumers and it should follow Unix norms. Selection of your overlay address space is also, by design, flexible.

### Fan Consumers

As noted earlier, the fan is intended to provide additional addresses for any particular host system in a local network to allow exposure of containers and VMs (and thereby services) to that network. A fan *consumer* is anybody who makes use of the expanded address space -- network administrators, Juju or MAAS, network server software, or a container such as LXC/LXD. These fan consumers should not be burdened with the intricacies of the underlying mechanism. Their interest should focus on the interface and ability to opt in to specific address ranges, gaining addresses in those ranges, and allocating those to their services. The fan consumer would find it desirable to expose additional services onto the local network and/or to configure these new networks and consume them much as if they were normal networks.

### Unix Norms

In evaluating the interface, the intention was to make it accessible, comfortable, and intuitive to those who will primarily use it. As we will see from the use cases (outlined below), much of this configuration is going to be performed by a system administrator or by tooling which modifies system configuration. We would like the tools to fit naturally into the networking tools available today and to be easily used from the existing persistent network configuration systems we already have. Similar tools already exist, for example we have `brctl` for managing ethernet bridges and `ip` for managing addresses. These are commonly used directly in `/etc/network/interfaces` for persistent configuration, such as adding alias addresses to an interface.

### Additional Address Space Possibilities

If using the 10.0.0.0/8 address space for the fan is impractical, other possibilities exist. The IANA IPv4 Address Space Registry (http://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xhtml) specifies /8 address space assignments. If you decide you don’t need to communicate with one of these network blocks, you can use it instead of the 10.0.0.0/8 block used in this document. For instance, you might be willing to give up access to Ford Motor Company (19.0.0.0/8) or Halliburton (34.0.0.0/8). The Future Use range (240.0.0.0/8 through 255.0.0.0/8) is a particularly good set of IP addresses you might use, because most routers won't route it; however, some OSes, such as Windows, won't use it.

## Fan Compared to Other Solutions

Of course, the fan is not the only approach to solving the problem of an exploding need for IP addresses in a heavily-containerised data center. Contrasting the fan with software-defined networking and other overlay systems (such as Flannel) may be helpful.

### Software-Defined Networking

The fan is not a software-defined network, in that it cannot provide arbitrary address virtualisation. You cannot, for example, choose where a particular fan address will be hosted -- it must be hosted on a specific host machine, behind the underlay address to which its subnet is mapped.

The fan cannot be used in cases where live migration of an interface from one host to another is expected, because the IP address could not live migrate from one host to another.

While live migration of containers is possible, notably in the case of LXD where containers behave just like virtual machines, the fan is mainly aimed at Docker-style use cases where a process would be terminated in one location and launched in a different location rather than migrated from one host to another.

### Flannel and Other Overlay Systems

A common workaround for container addressing is to create a database of containers and IP addresses mapped to hosts, and to arrange for arbitrary point-to-point tunnels or routes between host machines to enable containers to talk to one another. The fan is a simpler approach in cases where migration is not needed, and also where the typical number of overlay addresses is similar for every host in the system.

## Use Cases

The fan may be used in a variety of ways and by a variety of users. These range from manual setup by individual system administrators to automatic use by tools such as Juju and MAAS. Most uses are expected to use containers such as LXC/D or Docker.

### System Administrator/Machine Admin

System administrators are interested in configuring networking for their use. Specifically for the fan, they are interested in configuring specific /24 blocks from fans and making those available to use. For short term testing they will use the `fanctl` command interface directly. For persistent setup they will use the extended network configuration via `/etc/network/interfaces`. Where dynamic addresses are configured, they wish to be able to configure the interface based on its address.

### Juju and MAAS

Machine provisioning and modelling need to define fan subnets for consumption by the provisioned objects. This will generate fan configurations and be stored in the extended network configuration.

### Service Addressing

Where addresses are to be used directly by the host of the fan, we will simply bring up a fan as normal and bind services to those addresses.

### LXC/D/Docker

LXC/D and Docker are our primary use case. They need addresses to consume. These containers will be started attached to the appropriate fan bridge. For LXC we might create a fan specific template to indicate the specific bridges to use rather than the default lxcbr0.

## Configuring the Fan

As noted earlier, the fan's user-space tools fit neatly into the standard Unix/Linux model. These include both command-line tools used for one-time configuration changes and configuration file settings that can be used for more permanent settings.

### Fan Interface

We want a clear and simple interface which speaks to fan consumers' needs. We do not want to burden them with understanding of the underlying mechanism, just allow them to simply request the addressing they need. In particular, we are interested in viewing the interface from the point of view of the fan bridges, i.e. the pieces of address space which are available on the local host. We also want this interface to be general. We want it to express the semantics of the operation from the fan consumer's view. For example, their experience shouldn't be tied to knowing whether we are using a specific encapsulation on the wire, etc., as this may change going forward, or indeed be purposely different on varying substrates based on performance.

A fan-specific control program, `fanctl`, controls all creation, configuration, and tear-down of fan bridges and their associated mappings. This program is invoked to create and destroy fan bridges on demand, either from the command line or more likely from `/etc/network/interfaces` (see Persistent Configuration below):

```
sudo fanctl up 10.0.0.0/8 172.16.3.4/16
sudo fanctl down 10.0.0.0/8 172.16.3.4/16
```

Currently the fan mappings are limited to /8 on /16 only. We may in the future wish to be able to define fans with different subnet sizes, so these are specified when creating and deleting fan mappings. A benefit of the `fanctl` interface is that we also gain the ability to intuitively allow configuration of higher level functions as part of configuring the fan bridges. As an example, it is possible to specify whether a fan subnet has manual or automatic address allocation, which determines whether there is a `dnsmasq` configured on this fan subnet. The up command can take options to express this:

```
sudo fanctl up 10.0.0.0/8 172.16.3.4/16 dhcp
```

Further, where the local interface is not known, which happens when we have a persistent configuration on an underlying dynamically allocated network, we can use the interface name in place of the local address and `fanctl` maps that to the address as needed:

```
sudo fanctl up 10.0.0.0/8 eth0/16 dhcp
```

Where the local host wishes to talk to the fan segments both on and off the machine, it can use the same mechanism. The host can instantiate the fan bridge as normal and bind to the .1 address it is already allocated to communicate with other systems in the fan. It seems probable we would want to allocate more than a single address to the host in this scenario:

```
sudo fanctl up 10.0.0.0/8 172.16.3.4/16 host-reserve 4
```

This adds 4 additional addresses to the bridge for host use, 10.1.0.1-10.1.0.4

## Persistent Configuration

As alluded to above, in the common case we will want the fan configuration to be persistent and automatically applied across reboots. The existing `/etc/network/interfaces` network configuration can hold the fan configuration, as is commonly done for bridges and for alias addresses. The existing up/down command hooks can be used in a similar way to call out to `fanctl` to manage the attached fans.

A basic unmanaged (no-DHCP) fan configuration resembles the following:

```
iface eth0 static
    address 172.16.3.4
    netmask 255.255.0.0
    up fanctl up 10.0.0.0/8 172.16.3.4/16
    down fanctl down 10.0.0.0/8 172.16.3.4/16
```

A simple managed fan on a DHCP upstream looks like this:

```
iface eth0 dhcp
    up fanctl up 10.0.0.0/8 eth0/16 dhcp
    down fanctl down 10.0.0.0/8 eth0/16
```

Multiple fans can be defined simultaneously. Note here we have multiple local addresses and have chosen to use one of those in 10/8 and two of those in 241/8 (See "Additional Address Space Possibilities," earlier, for the rationale on using 241/8):

```
iface eth0 static
    address 10.1.0.1
    netmask 255.255.0.0
    up ip addr add 172.16.2.4/16 dev eth0
    up fanctl up 10.0.0.0/8 172.16.3.4/16
    down fanctl down 10.0.0.0/8 172.16.3.4/16
    up fanctl up 241.0.0.0/8 172.16.3.4/16
    down fanctl down 241.0.0.0/8 172.16.3.4/16
    up fanctl up 241.0.0.0/8 172.16.3.5/16
    down fanctl down 241.0.0.0/8 172.16.3.5/16
```

## Persistent Configuration (part deux)

It is also possible to define sets of fans via the new configuration file `/etc/network/fan`. This file defines one fan mapping per line, in the same form used on the `fanctl up` command line. This following example defines two fans 10/8 and 241/8 on a system with two underlay IP addresses:

```
# fan 240
10.0.0.0/8 172.16.3.4/16 dhcp
10.0.0.0/8 172.16.3.5/16 dhcp off
# fan 241
241.0.0.0/8 172.16.3.4/16 dhcp
241.0.0.0/8 172.16.3.5/16 dhcp
```

Note that comments are introduced by a `#` character in the first column and that blank lines are ignored. Further note that a new flag keyword is introduced which allows a line to be present but disabled.

Fan mappings defined in this table are brought up as below. This is performed on system boot:

```
fanctl up -a
```

All active fans defined in this table are taken down with:

```
fanctl down -a
```

## Use Examples

Some example configurations will help clarify how the fan is used in practice. As containerisation is an important use case for the fan, LXC and Docker configurations are shown here.

### Configuring LXC to Use the Fan

After configuring the fan as described above, install the `lxc` packages.

Edit `/etc/default/lxc-net`, set `USE_LXC_BRIDGE="false"`

Edit `/etc/lxc/default.conf`:

First, change `lxc.network.link` to specify the fan bridge, for example `fan-10-3-4` by default:

```
lxc.network.link = fan-10-3-4
```

Second, add a line specifying the container MTU, as follows:

```
lxc.network.mtu = 1480
```

At this point, new containers will be created using the fan overlay network. Existing containers may have their configuration file edited as described above to use the fan; the configuration file for a container is found in `/var/lib/lxc/[container name]/config`.

### Configuring LXD to Use the Fan

By default, LXD containers use the `default` profile, which gives them one eth0 nic attached to the `lxcbr0` bridge. To change the default profile to use the fan bridge, type `lxc profile edit default`, and change the entry to look like:

```
name: default
config: {}
devices:
  eth0:
    mtu: "1498"
    nictype: bridged
    parent: fan-250-0-127
    type: nic
```

In other words, update the parent to be the fan device, and add a mtu of 1498.

Alternatively, if you would like some containers to remain on lxcbr0, some on the fan bridge, and some on both, you could create a new profile called `fan` and assign one or both profiles to containers as desired.

## Configuring Docker to Use the Fan

After configuring the fan as described above, install the Docker packages, then edit `/etc/default/docker.io`, adding:

```
DOCKER_OPTS="-d -b fan-10-3-4 --mtu=1480 --iptables=false"
```

You must then restart `docker.io`:

```
sudo service docker.io restart
```

At this point, a Docker instance started via, e.g.,

```
docker run -it ubuntu:latest
```

will be run within the specified fan overlay network. Currently only one fan network subnet may be used with a running Docker instance.

FanNetworking (last edited 2015-06-22 16:12:29 by serge-hallyn)