INTRODUCTION TO CARP

BY MARIUSZ ZABORSKI

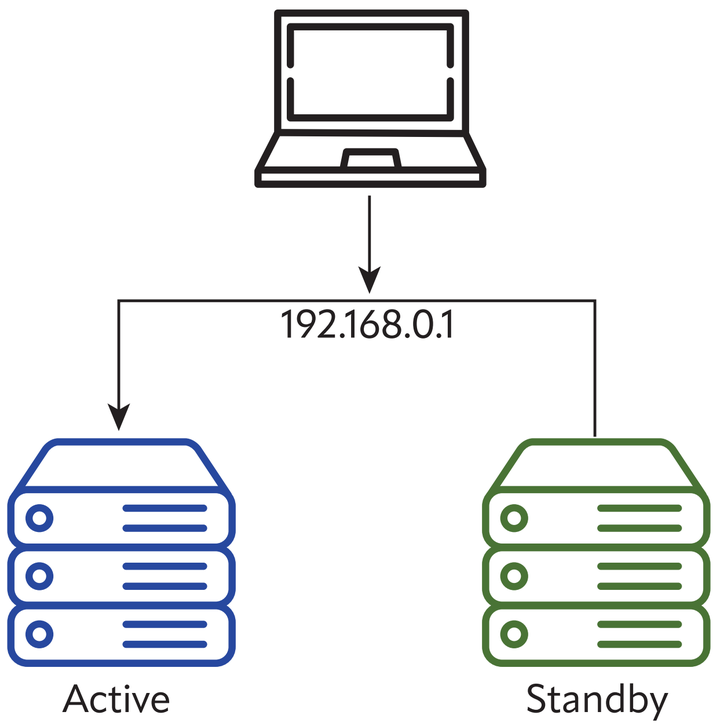
High availability topics might be challenging and complicated in large networks which is why we should look for solutions that are simple and easy to maintain and understand. CARP protocol, without any doubt, is one of them. CARP stands for *Common Address Redundancy Protocol* and its basic functionality is to allow multiple hosts to share a set of IP addresses.

The CARP protocol isn't new — it was first introduced in 2003 in OpenBSD as an alternative to CISCO protocol VRRP. CARP was to replace VRRP protocol because of patent issues, which are beyond the scope of this article. After CARP was introduced in OpenBSD, it was later integrated into FreeBSD and NetBSD. Finally, the ucarp was introduced — a userland implementation of CARP protocol — which brought an alternative to kernel implementations and made it available on Linux.

CARP Background

CARP allows a redundancy group — a set of hosts that share IP addresses. However, physically, only one interface has these IP addresses assigned (it is called the active host). In the case where an active host disappears (e.g., it was turned off or there is some issue with the network), other hosts in the redundancy group notice this and a new active host is elected. This situation is shown in Figure 1. There are two machines in the redundancy group — blue and green, but only the blue one is an active node, so other machines in the network don't have an issue choosing which to connect to.

Figure 1. Active and passive CARP nodes



In CARP, the active node is broadcasting its activity. This process is shown in Figure 2. Because the blue host is an active node, it is broadcasting CARP packages. The green and or-

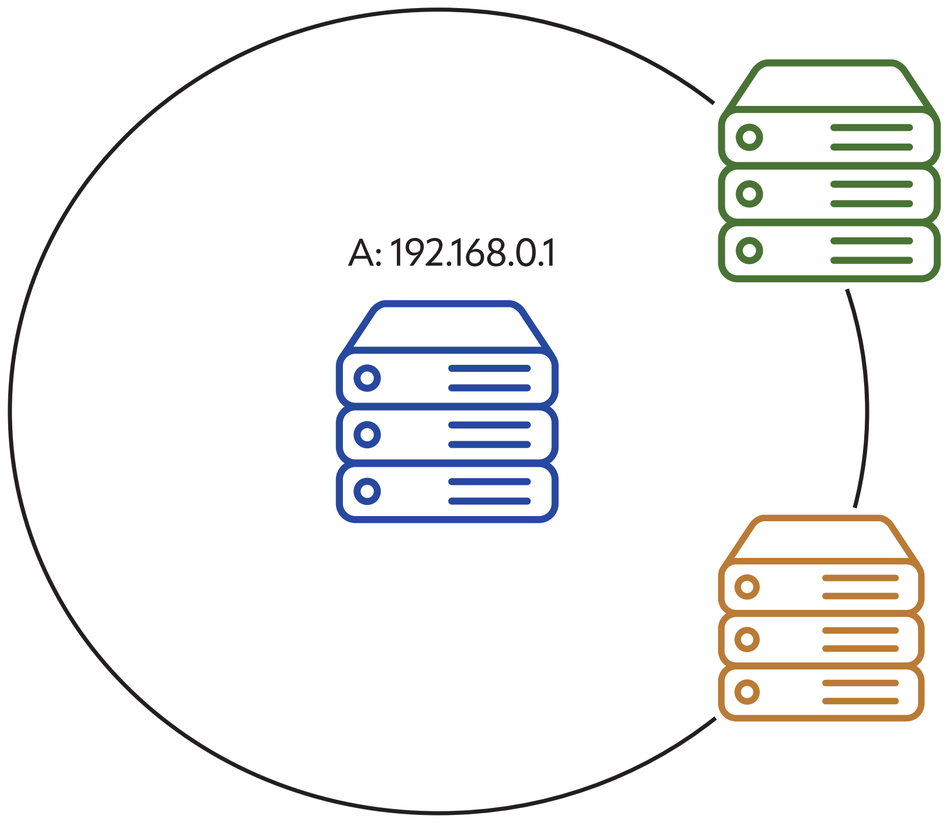
ange nodes are on standby and do not send any packages. The CARP package is quite small and contains only minimal information like:

- vhid (Virtual server ID), the identification of the redundancy group; all machines in the redundancy group have to share the same vhid
- Information about the CARP version and type of CARP package.

All packages in CARP are cryptographically signed, meaning each node in the redundancy group has to share the secret. CARP will never send its password in plaintext to the network. It is very important that every machine in the redundancy group be configured with exactly the same set of IP addresses. These IP addresses aren't sent over the network, however — they are used to calculate the cryptographical signature.

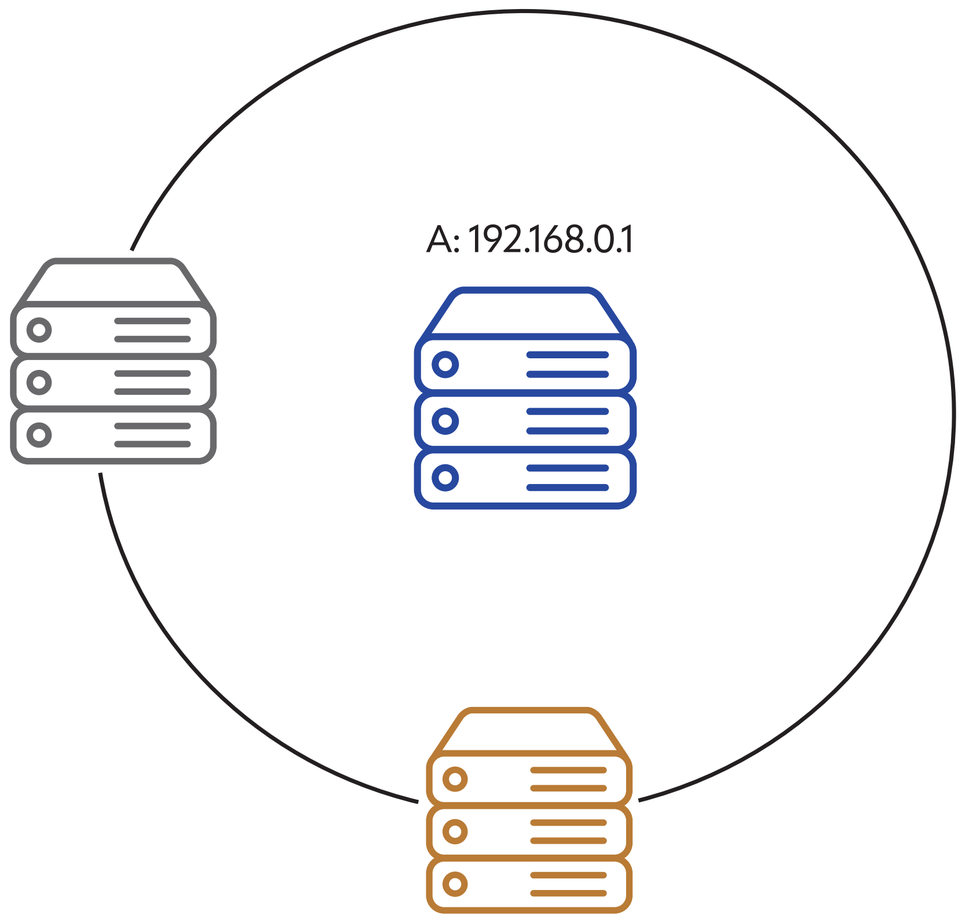
In the case shown in Figure 2, as long as the blue server is announcing cryptographically correctly signed packages with the given vhid, the other nodes don't do anything and just listen.

Figure 2. Announcing CARP packages



When a node stops receiving CARP packages for a while, another node decides to step in and become an active node. This situation is shown in Figure 3. The blue node, for some reason, stopped announcing the package, the green node noticed it, and now it has started to announce the CARP packages. When the blue node comes back, it will notice that the green node is now an active node and it will stay passive.

Figure 3. New active node



All examples above show a single IP address in the redundancy group. However, the redundancy group can have multiple IP addresses, and hosts can be in multiple redundancy groups — this is accomplished by different vhid. Thanks to that, we can also do some kind of load balancing among services in the network. For example, the green node can be an active node in the redundancy group, which provides the web server service, and the blue one can be an active node in the redundancy group, which provides the time service. If one of the nodes disappears, the other will become an active node in the other group.

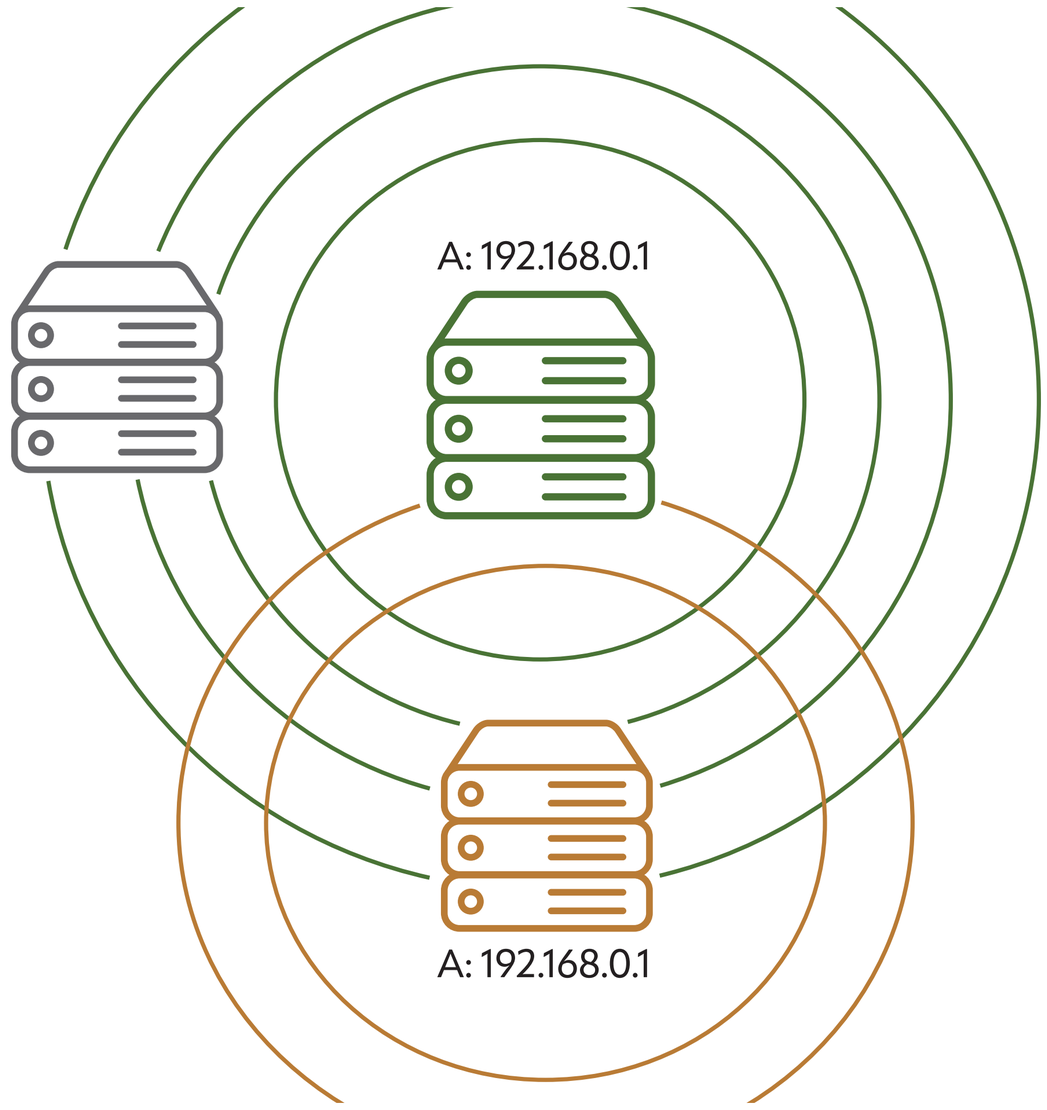
CARP and Split-brain

In a situation where two nodes notice, at the same time, that the node disappeared, both might want to become an active node. This is called a split-brain situation, where there are multiple active nodes. This situation might also occur when the link between the nodes is broken, and they stop seeing packages from each other and, after a while, the situation is fixed.

The split-brain issue is shown in Figure 4. CARP also solves this situation. When both hosts are active, both are announcing CARP packages. The node that announces more packages in a shorter period of time is the preferred node to become a new master. This is controlled in CARP with priority. Lower priority means packages are sent more often. When the other node sees that the CARP packages are announced more often than it is doing, it switches back to passive mode.

In the case when both nodes send packages with the same priority, the node will be chosen randomly.

Figure 4. Split-brain situation



FreeBSD Kernel Module CARP Configuration

CARP module is included in the default FreeBSD installation. From FreeBSD 10.0, CARP is no longer a pseudo-interface and it is configured directly on the interface. Listing 1 shows a basic configuration of CARP. First, we have to load a FreeBSD CARP module, which is accomplished by `kldload(8)` command. Then using `ifconfig(8)`, we define on which in-

terface the CARP should work (in our case it's **em0**). Next, we define a redundancy group ID (vhid is set to 1). Another important configuration is the passphrase used to calculate the checksum; this passphrase has to be shared among all hosts in the redundancy group. In the command, we also define the priority (or the advertisement interval). This is controlled by two parameters: **advbase** (advertisement base), which is specified in seconds, and **advskew** (advertisement skew — it is not shown on Listing) which is measured in 1/256 of a second. Just as a reminder — the lower priority means the host advertises more often, which means that it is a preferred node. Finally, we define which is the floating address.

On the same listing, we have two runs of **ifconifg(8)**; the parts not regarding CARP were omitted. In the first run, we can see that the redundancy group is in BACKUP state, which means that the interface is in standby mode and listening for CARP packages. Because there are no CARP packages in the network, it is switched to the **MASTER** (active) state, and the node starts to announce it. In Figure 5, we can see the captured CARP package, which is using a second static IP address for announcing the CARP packages to the multicast IP address. So, an additional IP address besides the shared one must be configured.

Listing 1. Configuration of CARP in FreeBSD

```
# kldload carp
# ifconfig em0 vhid 1 pass randompass advbase 1 alias 192.168.1.50/32
# ifconfig
em0:
     inet 192.168.1.50 netmask 0xffffffff broadcast 192.168.1.50 vhid 1
     carp: BACKUP vhid 1 advbase 1 advskew 0
# ifconfig
em0:
     inet 192.168.1.50 netmask 0xffffffff broadcast 192.168.1.50 vhid 1
     carp: MASTER vhid 1 advbase 1 advskew 0
     status: active
```

Figure 5. Captured CARP traffic using Wireshark

```
▸ Frame 3775: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface em0, id 0
▸ Ethernet II, Src: IETF-VRRP-VRID_01 (00:00:5e:00:01:01), Dst: IPv4mcast_12 (01:00:5e:00:00:12)
▸ Internet Protocol Version 4, Src: 192.168.1.157, Dst: 224.0.0.18
▾ Common Address Redundancy Protocol
  ▾ Version 2, Packet type 1 (Advertisement)
      0010 .... = CARP protocol version: 2
      .... 0001 = CARP packet type: Advertisement (1)
    Virtual Host ID: 1
    Advertisement Skew: 0
    Auth Len: 7
    Demotion indicator: 0
    Adver Int: 1
    Checksum: 0x04bd [correct]
    [Checksum Status: Good]
    Counter: 10009968722567644910
    HMAC: fb7f7879a6498338bee8bdf427c2b3c3a3c23fd0
```

What might come in handy is that FreeBSD **devd(8)** demon allows running additional scripts when the state has changed. Listing 2 shows an example of such a configuration from the FreeBSD man page. When the redundancy group changes its state, the **/root/carpcontrol.sh** script will be executed. The first parameter will be `vhid@inet,` and the second parameter will be the current state of the group.

Listing 2. devd(8) configuration for CARP

```
notify 0 {
      match "system"            "CARP";
      match "subsystem"         "[0-9]+@[0-9a-z.]+";
      match "type"              "(MASTER|BACKUP)";
      action "/root/carpcontrol.sh $subsystem $type";
};
```

ucarp

Additionally, a very promising project was a ucarp, the userland implementation of CARP protocol. It reduced the amount of code in the kernel space. Also, in the case of kernel space implementation, that might be slightly different. In this case, the code base was shared by multiple platforms. However, the project seems to have been abandoned--the GitHub project is closed, and the ucarp domain has expired. However, you can still find a ucarp distributed on different operating systems, so if you are looking for cross platform implementation, we still recommend you take a look at that project.

The configuration options are quite similar to the kernel implementations. Listing 1 shows how to install ucarp on a FreeBSD box. The next line shows its basic usage. Most options are self-explanatory at this point. Let's look into the **`upscript`** and **`downscript`** options. Because the ucarp was designed as a multiplatform tool, it doesn't know how to add an IP address to the interface — this responsibility was moved to the administrator. The user has to define his/her own scripts that add the IP addresses to the right interface.

Listing 3. Basic usage of ucarp

```
# pkg install carp
# ucarp --interface=eth0 --srcip=192.168.1.157 --vhid=1 --pass=randompass
--addr=192.168.1.50  --upscript=up.sh --downscript=down.sh
```

Another small caveat about ucarp is that we can define only one single floating IP address for the protocol. We can add many IP addresses in the **`upscript`** and **`downscript`**; however, only one (from parameter **`addr`**) will be added to the cryptographic signature. This means if we would like to mix the kernel and userland CARP implementations, it won't work with multiple floating addresses in the single redundancy group, because the checksum won't match.

Summary

Carp is a simple but very powerful tool that allows us to provide high availability in our network. There are two major CARP implementations: the kernel space (which each BSD operating system has) and one userland ucarp which is a cross-platform (and also works on Linux). Unfortunately, the userland implementation seems to have been abandoned. However, if you are looking for an easy and simple solution that will provide you with a floating address, you should still consider its use.

Bibliography

- CARP on Wikipedia — https://en.wikipedia.org/wiki/Common_Address_Redundancy_Protocol
- UCARP GitHub Project — https://github.com/jedisct1/UCarp
- CARP in FreeBSD Handbook — https://docs.freebsd.org/en/books/handbook/advanced-networking/#carp
- CARP FreeBSD man page — https://www.freebsd.org/cgi/man.cgi?query=carp&sektion=4

Acknowledgment

Figures in this article use resources from flaticon.com

MARIUSZ ZABORSKI currently works as a security expert at 4Prime. He has been the proud owner of the FreeBSD commit bit since 2015. His main areas of interest are OS security and low-level programming. In the past, Mariusz worked at Fudo Security, where he led a team developing the most advanced PAM solution in IT infrastructure. In 2018, he organized the Polish BSD user group. In his free time, Mariusz enjoys blogging at https://oshogbo.vexillium.org.