Virtual Subscriber Gateway (vSG)

Introduction

Consumer Premises Equipment (CPE)—often called a “home router” or “residential gateway” in a residential environment—runs a collection of essential functions (e.g., DHCP, NAT) and optional services (e.g., Firewall, Parental Control, VoIP) on behalf of residential subscribers. More sophisticated enterprise functions are also common (e.g., WAN Acceleration, IDS), but this paper focuses on residential functions. By extending the capabilities of CPE in the cloud, new value-add services as well as customer care capabilities can be provided more easily.

Our virtualized version of CPE, called virtual Subscriber Gateway (vSG), runs a bundle of subscriber-selected functions, but does so on commodity hardware located in the Central Office rather than on the customer’s premises. There is still a device in the home (which we still refer to as the CPE), but it can be reduced to a bare-metal switch, with most of the functionality that ran on the original CPE moved into CO and running in a virtual compute instance (e.g., a VM or container) on commodity servers.

Design Options

The CORD architecture permits a wide range of implementation choices for vSG. There are two underlying design issues: functionality and performance.

The first issue is how to implement the set of features (consumer services) that make up the bundle’s functionality. Options range from highly configurable systems like Click, to to an ad hoc combination of Linux capabilities (e.g., iptables, tc, dnsmasq). This first issue is primarily about functionality and agility.

To date, we have focused on a simple subscriber bundle implemented in Linux. CORD supports a Northbound Interface (NBI) that lets subscribers and operators select and control individual features (e.g., set parental control parameters on specific devices), with the corresponding feature implemented by a particular configuration of Linux (e.g., using dnsmasq to forward DNS traffic to an external parental control service).

A second issue is how to map subscriber bundles onto the underlying hardware resources. Options include packaging each bundle in its own VM, in a lightweight container running on bare metal, in a container embedded in a VM, or in a collection of containers. This second issue is primarily about performance and scalability.

To date, we have experimented with several strategies for implementing bundles on the underlying servers. These include:

- Each subscriber bundle is implemented in a Docker container running on bare metal. This approach ran into OvS implementation limits at roughly 1000 subscribers (containers) per server.
- Each subscriber bundle is implemented in a Docker container running in a VM. Testing of various configurations (e.g., 200 to 400 containers in each of 5 VMs) showed the target servers capable of supporting 1000 to 2000 subscribers each.
- Each subscriber bundle is implemented as a Linux namespace running in a VM. Although functionality was limited to only implementing DHCP, results show the configuration capable of supporting up to 4000 subscribers per server.

Detailed performance studies are reported elsewhere.

Current Design – Functionality

The current design implements a bundle of subscriber features in Linux, running in the container bound to that subscriber. The reference implementation supports basic Internet connectivity, as well as a collection of optional features, including:

- Suspend/Resume: Operators can suspend and resume a subscriber’s connectivity. Suspension is implemented by configuring the container to not forward traffic from the customer premises to the Internet, but the customer can still reach the services running in the vSG (e.g., DHCP, DNS).
- Restricted Access: Operators can temporarily restrict subscriber access by redirecting all traffic to a select website (e.g., a billing site for subscribers with delinquent accounts, a copyright training site for subscribers that have be flagged for copyright infringement). Restricted access is implemented by using iptables to redirect the subscriber’s HTTP traffic to a local web server that proxies for the remote site.
- Parental Control: Subscribers can apply parental control filters to different devices in the home. Parental control is implemented by redirecting DNS requests from a specific device (matched by MAC address) to a local dnsmasq that forwards them to an external parental control service like OpenDNS’s FamilyShield or Akamai’s AnswerX.
- Bandwidth Metering: Operators can set the upstream and downstream bandwidth available to subscribers. Bandwidth metering is implemented using tc’s Hierarchical Token Bucket queueing discipline.
- Access Diagnostics: Operators can run simple diagnostic tools on the subscriber’s connection. Diagnostics are implemented by running the selected tool (e.g., ping, traceroute, tcpdump) inside the vSG and returning the outputted text.
- Firewall: Operators and subscribers can set firewall rules for all traffic into the subscriber’s home. The firewall is implemented using iptables.

These features should be viewed as exemplars in the reference implementation. We make no claim to their being the only (or the best) possible implementation.

Current Design – Performance

The current design settles on a variant of a Container-in-VM configuration, but this is subject to change as we better understand the performance limits. The key challenge is how the container is logically connected to the network. We summarize the current implementation as follows.

A Docker container is allocated to each subscriber, with a set of containers co-located in a VM. Each container has two network interfaces: a “LAN-side” interface managed by vOLT and a “WAN-side” interface managed by vRouter. The overall configuration is shown in Figure 1.

On the LAN-side, vOLT assigns an s-tag/c-tag pair to each subscriber. The s-tag is bound to the VM; OvS does not strip the s-tag from the packets, but simply forwards the packets to the appropriate VM. The c-tag is bound to a container within the VM. Both s-tag and c-tag are stripped from the packets inside the VM, and then the packets are forwarded to the appropriate container.

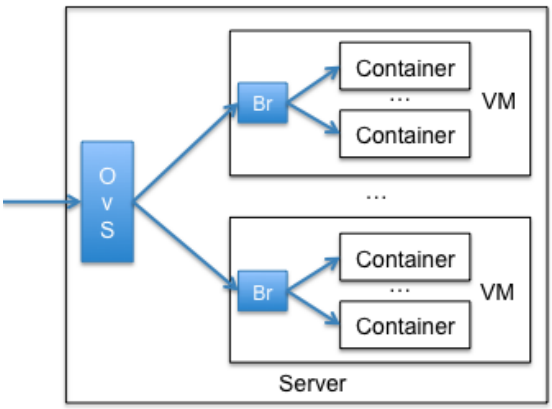

Figure 1. Per-Subscriber containers nested inside VMs.

On the WAN-side, both incoming and outgoing packets are labeled with a distinguished VLAN (tag=500) to indicate that they should be (a) sent to the container’s WAN interface, and (b) processed by vRouter. This tag is added/stripped inside the VM.

Subscriber API

CORD defines a Northbound API for controlling subscriber access, for example, adjusting the uplink and downlink speeds, suspending and resuming connectivity, setting parental control levels, and so on. Strictly speaking, this is an interface to CORD’s subscriber object (not the vSG service per-se), where CORD dispatches each requested operation to the appropriate service in the service graph.

The current API is sufficient to support the Customer Care portal in the reference implementation, but it is undergoing significant refactoring. An updated specification will be included in future versions of this document.