Bachelor's Thesis

Load Balancing and Failover for Isolated, Multi-Tenant Layer 2 Virtual Private Networks

by
Richard Wohlbold

Supervisors
Prof. Dr. Holger Karl
Internet Technologies and Softwarization

Hasso Plattner Institute at University of Potsdam

July 27, 2023

Abstract

In large Wi-Fi networks, broadcast traffic can take up lots of airtime. Common Wi-Fi systems filter broadcast traffic, breaking applications that rely on layer 2 (L2) functionality. WiMoVE is a Wi-Fi architecture that takes a different approach. By using L2 overlay networks on top of a layer 3 (L3) network, WiMoVE reduces broadcast transmissions while providing users with unfiltered L2 networks. In WiMoVE, a gateway forwards traffic into and out of the overlay networks. Previously, only a single gateway was considered. This leads to dependability and throughput issues. In this thesis, I consider a multi-gateway architecture. First, I propose a network architecture in which each gateway serves an L3 router and uses an interior gateway protocol (IGP) to advertise one route per overlay network. I then introduce the notion of gateways being responsible for overlay networks and identify a single-responsibility approach to be most appropriate. Afterward, I discuss potential load-balancing strategies and decide to use randomized load balancing. I show how to implement randomized load balancing using the IGP and WiMoVE control plane. Following that, I present three implementations that use different control plane combinations. I compare their performance based on measured failover times for varied numbers of overlay networks. The results indicate the implementation based on the Virtual Router Redundancy Protocol (VRRP) to perform best. Even though the implementation handles over 2000 overlay networks, large deployments require tens of thousands of overlay networks. I identify and analyze scalability issues in the implementations and discuss potential solutions.

Contents

List of Figures

List of Tables

Acronyms

AP access point

ARP Address Resolution Protocol

AS autonomous system

BGP Border Gateway Protocol

BUM broadcast, unknown-unicast and multicast traffic

CARP Common Address Redundancy Protocol

DHCP Dynamic Host Configuration Protocol

DPDK Data Plane Development Kit

ECMP equal-cost multi-path

EVPN Ethernet Virtual Private Network

FRR FRRouting

HRW highest random weight

ICMP Internet Control Message Protocol

IGP interior gateway protocol

IPSec Internet Protocol Security

L2 layer 2

L2TP Layer 2 Tunneling Protocol

L3 layer 3

LSA link-state advertisement

MAC medium access control

MSC message sequence chart

MTTR mean time to repair

NA Neighbor Advertisement

NAT network address translation

ND Neighbor Discovery

OSPF Open Shortest Path First

PoC proof-of-concept

RIDS Reachability Information Distribution System

RPC remote procedure call

RTNL routing table netlink

VM virtual machine

VPN virtual private network

VRRP Virtual Router Redundancy Protocol

VXLAN Virtual Extensible LAN

1 Introduction

In large Wi-Fi systems, station broadcast traffic can cause performance issues. Broadcast traffic has to be transmitted by each access point (AP), often at low rates. This takes up lots of airtime, reducing station throughput and increasing packet delay. In a Wi-Fi deployment with 500 APs and 20 stations per AP, basic Address Resolution Protocol (ARP) traffic can result in 167 packets per second that are transmitted by each AP, occupying 15 % of the available airtime [33].

Commercial Wi-Fi solutions often solve this issue by filtering broadcast, unknown-unicast and multicast traffic (BUM) traffic. Filtering is not transparent to the users and frequently breaks applications relying on L2 functionality. To address the broadcast issues, Schlitt et al. proposed the alternative Wi-Fi architecture WiMoVE [33].

1.1 WiMoVE

WiMoVE partitions stations into multiple overlay L2 networks. Broadcast traffic inside an overlay network only has to be transmitted by APs with stations inside that L2 overlay network. This reduces airtime, especially in scenarios with many overlay networks with few stations each.

While standard Wi-Fi systems require L2 connectivity between APs, WiMoVE only requires L3 connectivity between APs. Overlay networks are then implemented by encapsulating stations' L2 packets into L3 packets.

Like [33], I define an *endpoint* to be any network host that encapsulates overlay L2 packets into L3 packets and decapsulates L3 packets into L2 packets. A *device* is a network host that participates in an overlay network and has a medium access control (MAC) address for sending and receiving packets in that overlay network. I call a device *attached* to an endpoint when the endpoint encapsulates L2 packets originated by the device. Examples of endpoints are APs; examples of devices are stations. The gateway is a device in all overlay networks as well as an endpoint.

Using L2 overlay networks requires endpoints to know which device is attached to which endpoint. This information is called *reachability information*, defined as a partial function:

$$r : \text{MACAddresses} \to \text{OverlayNetworkIDs} \times \text{EndpointAddresses}$$

In WiMoVE, a control plane is responsible for distributing reachability information.

There are two operations on r: REACH and UNREACH. Let m be a MAC address, i an overlay network ID and e an endpoint address. REACH(m, i, e) marks the device with reachable at e inside the overlay network, overwriting the previous reachability information for m. UNREACH(m, i, e) marks the device as unreachable inside the overlay network, overwriting the previous information only if m was previously reachable at endpoint e in

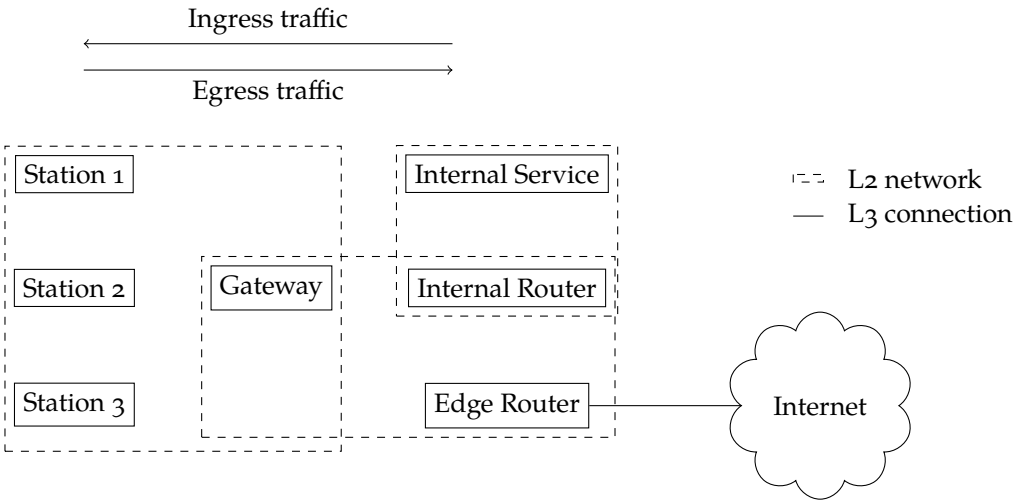


Figure 1.1: Station view of the network structure. L2 networks are drawn as dashed rectangles, L3 connections as edges. Ingress and egress traffic directions are annotated as arrows.

overlay network *i*. WiMoVE uses a component called Reachability Information Distribution System (RIDS) to distribute reachability information among the endpoints.

Another central component in WiMoVE is the *gateway*. The gateway is a member of each overlay network and serves as an IP router between the overlay networks and the outer network.

Figure 1.1 shows the overall network architecture from the point of view of a station. In the figure, two traffic directions are defined: I use *egress traffic* for network traffic which originates inside WiMoVE and is forwarded outside WiMoVE by the gateway. Similarly, I use *ingress traffic* for network traffic which originates outside WiMoVE and is forwarded into WiMoVE by the gateway.

The original WiMoVE architecture assumes exactly one gateway. This causes two problems:

- *Dependability*. The gateway is a single point of failure: If it fails, all stations lose Internet access. Since there is no migration protocol, performing maintenance on the gateway also leads to downtime.
- *Load Balancing*. The gateway is a bottleneck for all wireless Internet traffic. If the gateway becomes overloaded, packet loss and latency increase, decreasing station Internet service quality.

One natural idea is to use multiple gateways instead of one gateway. In this thesis, I explore the viability and limitations of this idea. I will formulate requirements for a multi-gateway system in the next section. In Chapter 2, I will look at literature relevant to the design. I will go into detail on necessary design decisions and arrive at a system design in Chapter 3. I will evaluate and compare different implementations in Chapter 4. In Chapter 5, I conclude my findings.

1.2 Requirements

In the previous section, I named two main goals for the multi-gateway architecture: The design should enable failover between gateways and provide load balancing. To achieve these goals, I break them down into specific requirements.

I first look at gateway failover. On failover, I differentiate between *planned* and *unplanned* shutdowns (i.e. failures) since the system should guarantee higher service quality in the case where a shutdown is planned. Therefore, I identify the following requirements:

R1	On planned shutdown, devices have continuous Internet access.
R2	On unplanned shutdown, the failure is detected and fixed in a way that all devices have Internet access again.

After a gateway has shut down, it must be able to join the system again. Therefore, I add the following requirement:

R3	Gateways can join the system dynamically.

The second goal is to provide load balancing between gateways. As more overlay networks are created and more stations join the system, three resources become limited on the gateways: processor time, memory, and link saturation. I assume that the control plane uses little of these resources compared to the data plane. Therefore, I only look at load balancing for the data plane. Processor time is mainly used for forwarding, encapsulating and decapsulating packets. Memory is used for packet queues. Therefore, reducing the data rate at the gateway can effectively lower the consumption of all three resources. Hence, I define the load-balancing requirement as follows:

R4	Data rate must be split between gateways such that no gateway is overloaded.

Let us look at a failover scenario. Some solutions restore Internet connectivity, but break all station flows in the process. This could happen because the gateways keep per-flow state that is lost on failover. Another reason could be that a station's public IP address changes during a failover, planned shutdown, or gateway startup. In these scenarios, even if the communication partners retransmit their packets, the flow is permanently broken. We will see one such solution in Section 3.2.1. Therefore, I add the following requirement:

R5	Stations' flows must be maintained on gateway startup and shutdown.

Lastly, I maintain the requirements that Schlitt et al. identified in [33]. For this thesis, the following two requirements are most important:

R6	The system is transparent to the stations; no changes to the way they operate are required.
R7	Stations must have Internet access and IPv4 and IPv6 connectivity if the L3 infrastructure allows for it.

1.3 Metrics

One key metric is *failover latency*. For the definition, I assume that one gateway fails and some WiMoVE devices lose Internet connectivity. Failover latency measures the time between the failure of a gateway and restored Internet connectivity.

For a station to have Internet connectivity, ingress and egress packets have to be delivered. Therefore, I define *ingress failover latency* as the time between a gateway failure and the time when ingress packets can be delivered again. Similarly, I define *egress failover time* as the time between a gateway failure and the time when egress packets can be delivered again. For a viable multi-gateway architecture, the mean failover latency should be smaller than the mean time to repair (MTTR) of a single gateway.

Other metrics include the gateway startup time and the gateway shutdown time. I define these as the duration between a gateway startup and the time when the system reaches a steady state and between the gateway shutdown initiation and the shutdown completion. These metrics are interesting for operational aspects of the system, but I do not consider them as critical as the failover latency. I will therefore not consider them further in this thesis.

2 Background

Besides the original paper, there is no literature on WiMoVE. Still, some systems have architectural similarities to WiMoVE but serve different purposes. I specifically look at components I can use during the system design. In the first section, I will discuss virtual private networks (VPNs). There, I identify one solution used create redundancy, router redundancy protocols, which I address afterward. Lastly, I look at load-balancing strategies.

2.1 Virtual Private Networks

The structure of WiMoVE is similar to that of an L2 VPN. A VPN is a virtual network built on top of existing physical networks that can provide a secure communication mechanism for data and IP information transmitted between networks or between different nodes on the same network [31]. VPNs are implemented through tunneling protocols that use encapsulation to move data between VPN members. VPNs often allow users to route traffic outside the VPN, using an intermediate server as the router [11]. Common VPN implementations operate either on L2 or L3. Examples of L2 VPNs include Layer 2 Tunneling Protocol (L2TP) and SoftEther [38, 34]. Examples of L3 VPNs include Internet Protocol Security (IPSec), OpenVPN, and WireGuard [3, 29, 18].

In WiMoVE, each overlay network can be seen as a VPN. Then, the gateway is the intermediate server that forwards packets outside the VPN. In contrast to traditional VPNs, the security aspect is less relevant in WiMoVE. Since WiMoVE encapsulates L2 packets to L3 packets, it can be compared to an L2 VPN.

One more feature of WiMoVE as a VPN is multi-tenancy: There is more than one virtual network. This is different from e.g. a corporate network where all VPN users are part of the same virtual network. Multi-tenant VPNs are often used in commercial settings, two of which I will look at in the following sections.

2.1.1 Cloud Providers

In cloud computing, customers use VPNs to connect their on-premise network to their virtual network inside the cloud. The cloud provider implements customers' virtual networks as overlay networks. VPN gateways connect these networks, mapping between them and encapsulating and decapsulating packets.

In [2], Arashloo et al. discuss a design for the VPN gateways of a large cloud provider. They identify similar requirements: The VPN gateway must participate in many overlay networks, tracking millions of tunnels. However, the authors optimize for different metrics: Instead of focusing on load balancing and failover, they prioritize high throughput and cost-efficiency. Instead of scaling out, they opt for one commodity server and switch.

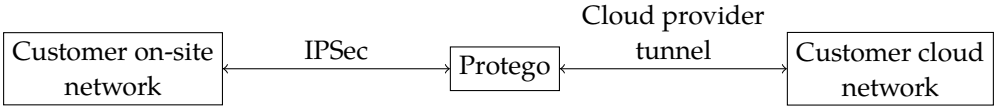


Figure 2.1: Conceptual structure of *Protego* as gateway between two networks

In [35], Son et al. design *Protego*, a distributed IPSec gateway service for a cloud provider. Figure 2.1 shows the conceptual structure of Protego as gateway between two networks. They use a distributed design with active redundancy to achieve high availability, assuming multi-tenancy with a dynamic number of gateway virtual machines (VMs). The authors' key goal is efficient resource provisioning: Instead of using one IPSec gateway per tenant, there are multiple tenants per gateway. Other goals include failover as well as providing users with an agreed-upon throughput. As their load-balancing algorithm, the authors use a greedy algorithm for bin packing with thresholds. I will look at the load-balancing algorithm in more detail in Section 2.3. While the paper solves a similar problem, the specifics are different. First, Protego makes use of dynamic resource provisioning of the cloud. This is not the case for many on-premise networks where WiMoVE is deployed. Also, WiMoVE and Protego deal with different VPN control and data planes. While the authors emphasize IPSec L3 tunnel migration, I deal with WiMoVE L2 device migration. On the egress side, I will deal with an IGP while Protego deals with a provider-internal VPN. These differences lead to different trade-offs and different design decisions. Therefore, besides the load-balancing algorithm, few of the design choices from Protego apply to our problem.

2.1.2 Commercial VPNs

Besides cloud providers, VPN gateways are also used by commercial VPN vendors. Commercial VPN vendors let users connect to their servers using a tunneling protocol. Their servers decapsulate the packets and forward them to the Internet. Commercial VPNs often use private IP address spaces for the virtual networks themselves. Therefore, the VPN gateways perform network address translation (NAT) [15].

NAT is a method by which IP addresses are mapped from one address realm to another, providing transparent routing to end hosts. It allows hosts on a private network to transparently communicate with destinations on an external network and vice-versa [16]. This results in users' IP addresses being masked, providing anonymity for them. I will discuss whether to do NAT in Section 3.2.

Few open-source VPN solutions contain a failover and load-balancing mechanism and few commercial VPN vendors document their solutions. One vendor that does document their solution is *OpenVPN*. According to [30], their VPN product contains a "failover mode" that uses an implementation of the Common Address Redundancy Protocol (CARP), a router redundancy protocol [8]. I look at router redundancy protocols in the next section.

2.2 Router Redundancy Protocols

Router redundancy protocols provide failover for IP routers. There are many router redundancy protocols, most notably VRRP, which is standardized in RFC 5798 [27].

The idea of VRRP is to multi-home a virtual router between multiple physical routers. Every physical router is either in *Master* or *Backup* state. VRRP coordinates the router states using a priority-based election mechanism. Routers multicast periodic advertisements containing their priority. Each physical router sets its state to master if and only if it has the highest priority out of all physical routers. After not receiving advertisements from the master for a certain amount of time, the physical routers elect a new master.

The virtual router has a separate IP and MAC address. These addresses are not used to send packets, only to receive them. Hosts use the virtual IP address as their next hop. The master router is responsible for sending gratuitous ARP responses and unsolicited Neighbor Advertisements (NAs). Using backward learning, switches in the L2 network then associate the MAC address of the virtual router with the port of the master router. The master router then receives all traffic addressed to the virtual router.

In WiMoVE, the gateways can be viewed as IP routers between L2 overlay networks and the rest of the network. Before using VRRP for WiMoVE, multiple questions remain:

- VRRP assumes to be used in an Ethernet network. In WiMoVE, there are many isolated L2 overlay networks. Are WiMoVE and VRRP compatible? What is the relationship between WiMoVE and VRRP concepts?
- How can load balancing be incorporated into VRRP in the WiMoVE context?

I will address these questions in Chapter 3.

2.3 Load-Balancing Strategies

According to R4, incoming and outgoing traffic should be load-balanced between the gateways. This requires a load-balancing strategy.

Traditional load-balancing literature deals with $n \in \mathbb{N}$ servers with tasks arriving as part of a Poisson process. The task of the load-balancing algorithm is to assign tasks to servers. A common approach is the supermarket model where each task independently and uniformly chooses $d \in \mathbb{N}$ out of the n servers. Out of the d servers, the one with the smallest queue is chosen [23]. There is no clear way of applying this model to multiple gateways within WiMoVE. First, it is unclear on which level to load-balance: Should individual devices be load-balanced, or should all devices within the same overlay network be handled by the same gateway? Also, there is no equivalent notion of "arrival" within WiMoVE. Devices may be attached for a long time, therefore reassigning tasks between gateways could be necessary.

There are also load-balancing strategies more specific to our problem. In [32], the authors propose a load-balancing strategy for ingress traffic in the context of an autonomous system (AS). They model the load-balancing problem as an assignment problem of users to edge links. At each time step, the data rate used by each user is measured and users are reassigned. The authors show an optimal assignment scheme to be NP-complete and propose a greedy algorithm that performs the assignment.

Similarly, in [35], while designing a multi-tenant VPN gateway, Son et al. model the assignment problem as a bin-packing problem. Bin-packing deals with $n \in \mathbb{N}$ objects, each of a different size $s_i \in \mathbb{N}$, and some bins of equal capacity. The goal is to assign the objects to the bins using as few bins as possible. Bin-packing is NP-complete, but there are

approximation algorithms such as best-fit-decreasing [5]. In both papers, all nodes report their metrics to a central server that performs the assignment using an approximation algorithm. I call algorithms for which gateways report metrics *metric-based*.

Instead of using a deterministic algorithm, load balancing can also be performed at random. For example, devices could be assigned uniformly to gateways. By using hash functions for the assignment, no communication between the gateways is required. A naive solution, linear hashing, leads to lots of reassignments when a gateway joins or leaves the system. Instead, a strategy that minimizes reassignments should be used: When a gateway joins, only users assigned to that gateway should be moved. The same goes for a failing gateway. The most common techniques are consistent hashing [19] and highest random weight (HRW) hashing (also called rendezvous hashing) [9].

HRW hashing assumes a situation where there are servers and objects to distribute among the servers. Assuming servers $s_1, \ldots, s_n$, each object should be located on a random k-subset of servers. HRW ensures that for all objects, the servers agree on the k-subset. Let S be the set of servers, O the set of objects and $\hat{w} \in \mathbb{Z}^+$ the maximum hash value. All servers use the same hash function $h : S \times O \to \{1, \ldots, \hat{w}\}$ that takes an object name $o \in O$ and server $s \in S$ as input and produces a weight as output. Let $o \in O$ be an object. For each server $s_i, 1 \leq i \leq n$, the weight $h(s_i, o)$ is calculated. The k-subset for o is defined as the k servers with the highest weight. This results in all servers agreeing on a random mapping of objects to servers. The scheme also minimizes reassignments when servers join or leave the system. Compared to consistent hashing, HRW hashing requires less communication between nodes than consistent hashing, so it should be preferred for random assignment.

3 System Design

In this chapter, I will discuss design questions, possible solutions and associated trade-offs. At first, I will briefly talk about the number of gateways I use. Then, I discuss the network design, especially address assignment and NAT. Afterward, I introduce a differentiation between ingress and egress gateways. I use these definitions to arrive at suitable gateway responsibility models and decide between them. Then, I discuss the question of load balancing. With the load-balancing strategy, I design suitable ingress and egress control planes. Finally, I summarize the decisions into a single system architecture.

3.1 Static or Dynamic Number of Gateways

Before going into the design, we need to think about the number of gateways to use. Fundamentally, there are two options: The number of gateways could be dynamic or static. *Static* means that even though gateways may join the system dynamically (see R3), the control plane does not automatically provision gateways. With a dynamic number of gateways, the control plane can provision additional gateways to take load away from existing gateways.

Using a static number of gateways is simpler: There is no need to think about when to scale out or in. It also imposes fewer requirements on the underlying infrastructure since there may not be a way to programmatically create network hosts in every environment.

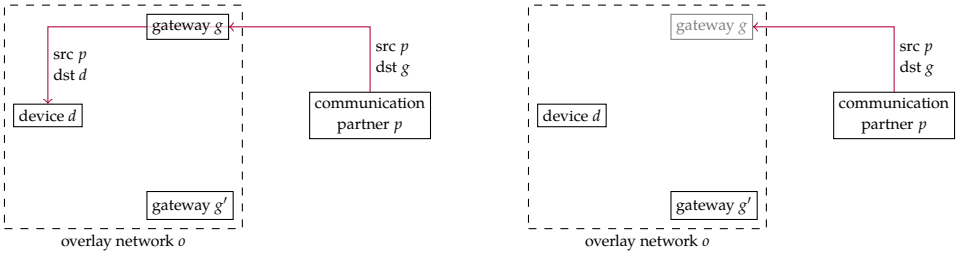
Using a dynamic number of gateways allows to dynamically scale out: In situations with high system load, e.g. a large event, new gateways are provisioned automatically, if there is enough capacity. When the load decreases again, the assignment algorithm can scale in, freeing resources for other tasks.

In large Wi-Fi systems, the gateways could be run on-site or in the cloud. If the gateways are run on-site, there may not necessarily be a way to automatically provision new gateways. Still, the architecture should be applicable to these scenarios. Therefore, I make the following design decision:

Decision 1. *The number of gateways is static.*

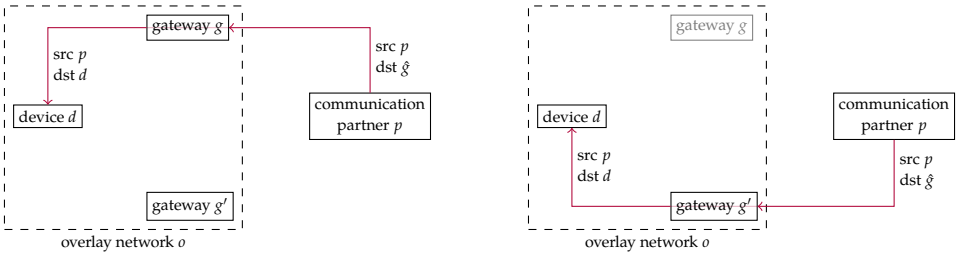
3.2 Network Design

An important design question is whether the gateways perform NAT. I defined the term in Section 2.1.2. If the gateways performed NAT, the private address realm would be one or multiple overlay networks. The external network would be the rest of the private network, e.g. the company intranet, as well as the Internet. We only need to consider NAT if it is done by the gateways. If any other routers perform NAT in the network, we can treat the network scenario the same way we would treat a scenario without NAT. The same is true for cascaded NAT which behaves like a single NAT for us.



(a) Before failure **(b)** After failure

Figure 3.1: Flow between device d and communication partner p through gateway g with basic NAT. Gateways use their own IP address for NAT. I only look at the communication direction $p \rightsquigarrow d$. After failure of g, the flow is broken.



(a) Before failure **(b)** After failure

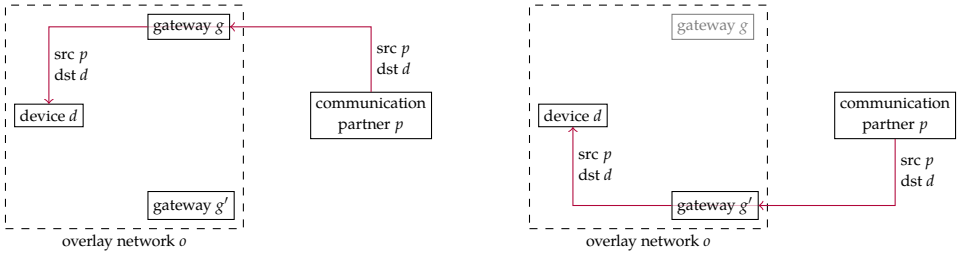
Figure 3.2: Example flow between device d and communication partner p through gateway g with distributed NAT. I only look at the communication direction $p \rightsquigarrow d$. g distributes the NAT state to g'. $\hat{g}$ is an anycast address to g and g'. After failure of g, traffic is anycasted to g', maintaining the flow.

Whether to do NAT influences how IP addresses can be assigned inside the overlay networks. It also decides if and how changes in the overlay networks are propagated through the network. Lastly, it determines whether the gateways have to keep per-connection state. In the following sections, I will discuss the advantages and disadvantages of performing NAT in WiMoVE.

3.2.1 NAT

I first look at a network setup where each gateway performs NAT. Let F be a flow between a device d that is inside an overlay network o and a communication partner p outside WiMoVE. Let the flow be forwarded via gateway g. Figure 3.1a shows this scenario.

d sends IP packets with its own source address and destination address of p via g. In these packets, g replaces the source address of d with their own unicast address. p sends IP packets with its own source address and destination address of g. Before forwarding these packets to d, g replaces its own destination address with the address of d. This makes F sticky to g: There is no way to tell p to send packets of flow F to another gateway g'. On failure of g, this breaks F since packets from p do not reach d anymore. Figure 3.1b shows the situation after g has failed. The flow F stays broken until g starts up again. Therefore, using the gateways' unicast addresses as NAT addresses violates requirement R5.



(a) Before failure

(b) After failure

Figure 3.3: Example flow between device d and communication partner p through gateway g without NAT. I only look at the communication direction $p \rightsquigarrow d$. After failure of g, traffic gets routed via g', maintaining the flow.

This issue can be fixed by having a set of gateway IP addresses that are anycasted to the gateways. In the above scenario, if g were to fail, packets would still be sent to g'. For other hosts to stop forwarding packets to g, a dynamic routing protocol is necessary. To forward packets from p to d, g' needs the NAT state of g. Therefore, all NAT state is now shared state that needs to be distributed between gateways. Figure 3.2 shows a failover scenario with distributed NAT. We observe that if g' receives the NAT state from g, the flow between d and p is maintained.

Distributed NAT setups are usually pretty complex, even large network operators try to avoid them where possible [21]. This is in part due to complicated failure cases: In the above scenario, if the flow between d and f has been established and g fails without having propagated the NAT state, the NAT state is lost and F is permanently broken. Waiting for NAT state propagation is not an option since that increases latency on each flow creation. Therefore, a distributed NAT setup cannot guarantee to fulfill R5.

Another disadvantage of NAT is that p cannot establish a flow to d since the address of d is not guaranteed to be routable for p. This limits some applications which require creating flows to a WiMoVE device from outside.

Despite these disadvantages, a NAT-based network architecture also has some benefits: Most prominently, it does not impose any restrictions on the overlay network address ranges. For example, all overlay networks could use the same IP addresses without any problems.

3.2.2 No NAT

I will now look at a network architecture without NAT. In this scenario, I treat all overlay networks as regular IP subnets. There, gateways serve as L3 routers.

To enable gateway failover, a dynamic routing protocol is necessary so that packets are no longer forwarded to the failed gateway. I use the term IGP for the dynamic routing protocol. Therefore, gateways take part in the WiMoVE control plane as well as the IGP.

Figure 3.3 shows the same scenario as in the last section but without NAT. We observe that traffic is rerouted through another gateway g'.

The advantage of this setup is that a distributed NAT installation is not required. The main disadvantage is that the IP address ranges of all overlay networks need to be disjoint.

According to R7, the system needs to support IPv4 and IPv6. Therefore, I look at the address assignment for both protocol versions.

For IPv6, using disjoint address ranges for all overlay networks is not an issue. In IPv4, address space is more limited. Therefore, using disjoint address ranges creates an upper bound for the number of overlay networks. Let us estimate the size of this upper bound. WiMoVE deployments can only use addresses in the private IP address ranges 192.168.0.0/16, 172.16.0.0/12, and 10.0.0.0/8. This results in $(1 + 16 + 256) \cdot 256 = 69\,888$ subnets. If ten addresses per subnet are reserved for network services, this would support 17.7 million devices, assuming the most efficient allocation. In realistic deployments, this upper bound is high enough to not become a problem.

Besides the address space limitations, not using NAT also increases the load on the routing infrastructure. Depending on the route granularity, this approach introduces a high number of routes into the routing infrastructure. I address the route granularity in Section 3.2.3. In real IGPs, this may cause link-state database overflows, leading to inconsistent views of the network and possibly incorrect routing [25]. It is unclear whether common routing protocols and their implementations can deal with the load that WiMoVE adds to the system. I will gather evidence on this question in Chapter 4.

I decide that the reduced complexity of avoiding a distributed NAT setup is worth the drawbacks:

Decision 2. *Overlay networks are regular IP subnets. Gateways function as regular L3 routers, taking part in the WiMoVE control plane and the IGP.*

3.2.3 Route Granularity

In the last section, I decided that gateways serve as IP routers to the overlay networks. The last network design decision is the route granularity: What is the smallest unit that is advertised by the gateways via the IGP?

There are two main options: One option is to advertise one route for each device. The other option is to aggregate multiple devices or overlay networks into one route.

Advertising one route for each device improves load balancing: The load-balancing algorithm gets additional degrees of freedom. Therefore, it can achieve a more even load distribution. The disadvantage is that more routes have to be propagated and installed by the routing protocol. If there were one route per device, routing changes would have to be propagated whenever a device becomes attached or detached.

WiMoVE is designed to use many overlay networks, e.g. one per device, such that broadcast traffic can be reduced effectively. This seems to give enough degrees of freedom to the load-balancing algorithm; the improvements in routing performance seem worth the cost. Therefore, I make the following design decision:

Decision 3. *The smallest unit of assignment is the overlay network.*

3.3 Gateway Responsibility Models

Another central design question is which gateway is responsible for which overlay network. In this section, I first define the notion of responsibility, differentiating between ingress

and egress responsibility. For both ingress and egress, I look at how single and shared responsibility can be achieved, discuss the involved trade-offs and decide on the model I use.

In a multi-gateway WiMoVE network, ingress and egress traffic are inherently different: Ingress traffic is routed to the gateways using the IGP while egress traffic is forwarded using the WiMoVE control plane. We observe that ingress and egress routing does not have to be symmetric: The gateway that routes traffic into an overlay network does not have to be the same as the gateway that routes traffic from the overlay network into the rest of the network.

I also want to differentiate between the device inside the overlay networks that sends L2 packets and the network hosts that serve as IP routers. Therefore, I use the following definitions:

Definition 1 (Ingress Gateway)**.** *An* ingress gateway *is a device inside an overlay network that is responsible for forwarding traffic from the Internet into the overlay network.*

Definition 2 (Egress Gateway)**.** *An* egress gateway *is a device inside an overlay network that serves as the IP next hop for all other devices inside that network, forwarding traffic from the L2 overlay network into the Internet.*

Definition 3 (Physical Gateway)**.** *A* physical gateway *is a host that serves as ingress or egress gateway for any overlay network.*

Definition 4 (Responsibility)**.** *A physical gateway is* responsible *for an ingress or egress gateway when the tasks associated with the ingress or egress gateway are performed by the physical gateway.*

Two notes on the terminology in the following chapters: I say that a physical gateway is responsible for an overlay network if it is clear from context if I am referring to an ingress gateway or an egress gateway. If I say "gateway" without an adjective, I mean physical gateway.

I now look at different responsibility models. For these models, I only look at steady states, i.e. the network is not in a failover or migration process. The definitions are given for ingress responsibility; I use equivalent definitions for egress responsibility. For both ingress and egress traffic, there are two basic responsibility models: First, there is the *single-responsibility* model: In this model, for all overlay networks o, there is exactly one gateway that forwards traffic leaving o. Second, there is the *shared-responsibility* model: In this model, for all overlay networks o, there are multiple gateways that forward traffic leaving o. Since the advantages and disadvantages differ when looking at ingress or egress responsibility, I treat both cases separately.

3.3.1 Ingress Gateway Responsibility

In most IGPs, ingress traffic can be influenced using link costs. Let G be the set of all gateways and R be the set of all other IGP routers, i.e. $G \cap R = \emptyset$. Let O be the set of overlay networks. I treat the L3 routing structure as a weighted, undirected graph with node set $V = R \cup G \cup O$. I deliberately include the overlay networks as nodes, representing all devices inside an overlay network $o \in O$ by o.

The mental model is the following: An L3 network with arbitrary link costs is given. For each gateway $g \in G$ and overlay network $o \in O$, we can control the weight of edge $\{g, o\}$.

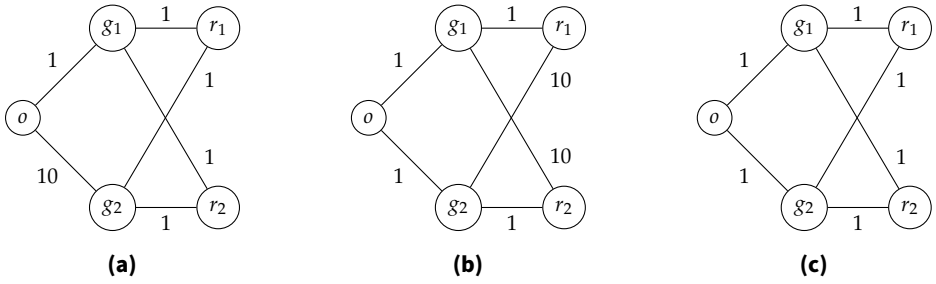


Figure 3.4: Single responsibility in different scenarios. $G = \{g_1, g_2\}, R = \{r_1, r_2\}, O = \{o\}$.
(a) g_1 is singly-responsible: For all $r \in R$ and all shortest paths $P : r \rightsquigarrow o$, $g_1 \in P$.
(b) No single responsibility: The shortest path from r_1 to o is $P_1 : r_1 g_1 o$ and the shortest path from r_2 to o is $P : r_2 g_2 o$.
(c) No single responsibility: The shortest paths from r_1 to o are $P_1 : r_1 g_1 o$ and $P_2 : r_1 g_2 o$.

During the next sections, I introduce the single-responsibility and shared-responsibility models and analyze how I can implement them using link costs.

3.3.1.1 Single Responsibility

Definition 5. *Let $g \in G$ be a gateway and $o \in O$ an overlay network. g is* singly-responsible *for o if and only if for all $r \in R$, for all shortest paths $P : r \rightsquigarrow o$, $g \in P$.*

Figure 3.4 shows three example scenarios and applies the above definition. I now look at conditions on the costs with which o is advertised by the gateways. Let $c : V \times V \to \mathbb{N}$ be the path cost between two nodes. Let $g \in G$ be singly-responsible for o and let g advertise o with cost $d \in \mathbb{N}$. Then, all other gateways $g' \in G \setminus \{g\}$ must advertise o with a cost $> d$ since otherwise, the path $P' : g'o$ would be shorter than any path containing go. Let $d' \in \mathbb{N}$ be the lowest cost with which o is advertised by another gateway $g' \in G \setminus \{g\}$. Let us also fix an IGP router $r \in R$.

Lemma 1. *If g is singly-responsible for o, then $d' - d > c_{r,g} - c_{r,g'}$.*

Proof. Let us assume that $d' - d \leq c_{r,g} - c_{r,g'}$. It follows that $c_{r,g'} + d' \leq c_{r,g} + d$. Then, there exists a shortest path $P' : r \rightsquigarrow o$ with $g' \in P'$. Therefore, g cannot be singly-responsible for o. □

Stated informally, I have the following result: Since paths from routers to gateways have different costs, advertisement costs for o need to be sufficiently different to achieve single responsibility for o.

One way to achieve single responsibility is to choose a set of cost levels that are sufficiently far apart. All gateways advertise all overlay networks with one of these costs.

Let $\Delta c = \max_{r \in R}[(\max_{g \in G} c_{r,g}) - (\min_{g \in G} c_{r,g})]$. Informally, for each router $r \in R$ and gateway $g \in G$, we look at the length of the shortest paths from r to g. For each router, we calculate the difference between the distances to the farthest gateway and the closest gateway. Δc is the maximum difference over all routers. Figure 3.5 shows two example networks and their Δc. I define the cost levels as $C = \{k(\Delta c + 1) \mid k \in \mathbb{Z}_0^+\}$.

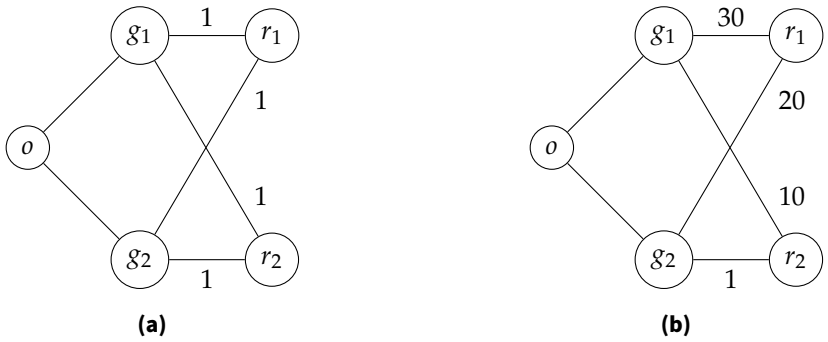


Figure 3.5: Δc in different scenarios. $G = \{g_1, g_2\}, R = \{r_1, r_2\}, O = \{o\}$.
(a) $c_{r_1,g_1} = c_{r_1,g_2} = c_{r_2,g_1} = c_{r_2,g_2} = 1$, therefore $\Delta c = 0$.
(b) $c_{r_1,g_1} = 30$ and $c_{r_1,g_2} = 20$, therefore $\Delta c = 10$.

Lemma 2. *Let $g \in G$ be a gateway and $o \in O$ be an overlay network. Let g advertise o with $d \in C$. Let all other gateways $g' \in G \setminus \{g\}$ advertise o with costs $d_{g'} \in C$ with $d_{g'} > d$. Then, g is singly-responsible for o.*

Proof. Let $r \in R$ and let $g' \in G \setminus \{g\}$ be any other gateway.

Let $P : r \rightsquigarrow g$ be a shortest path from r to g. P then has cost $c_{r,g}$ and Po has cost $c_{r,g} + d$. Let $P' : r \rightsquigarrow g'$ be a shortest path from r to g'. P' then has cost $c_{r,g'}$ and $P'o$ has cost $c_{r,g'} + d'$.

According to the definition of Δc,

$$c_{r,g} - c_{r,g'} \leq \Delta c.$$

Then,

$$\begin{aligned} c_{r,g'} + d' - (c_{r,g} + d) &= c_{r,g'} + d' - c_{r,g} - d \\ &> c_{r,g'} - c_{r,g} + \Delta c \\ &\geq -\Delta c + \Delta c \\ &= 0 \end{aligned}$$

Therefore,

$$c_{r,g'} + d' > c_{r,g} + d.$$

Therefore, P is the shortest path from r to o and g is singly-responsible for o. □

Most network protocols have a maximum cost $\hat{c} \in \mathbb{N}$. Therefore, there are $\lfloor \frac{\hat{c}}{\Delta c + 1} \rfloor + 1$ cost levels available for a single responsibility model.

I now look at the advantages and disadvantages of a single-responsibility model. One advantage is that a single-responsibility model simplifies the load-balancing algorithm: Instead of a graph problem, the algorithm needs to solve an assignment problem. Therefore, a single-responsibility model makes load balancing much simpler. The disadvantage is the limited number of cost levels: For some networks, there may be few cost choices left. This limits the assignment algorithms used for load balancing. I will look at specific algorithms and their limitations in Section 3.5.

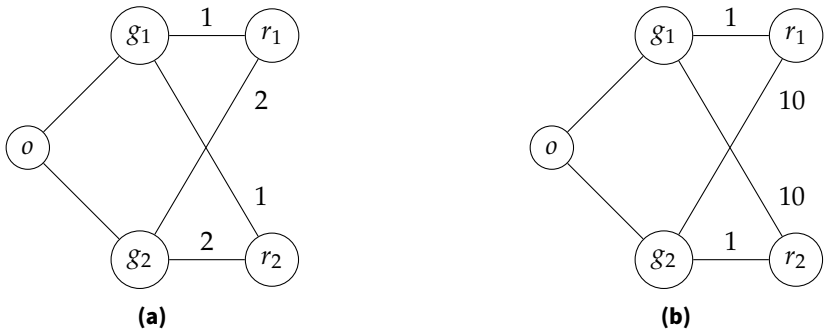


Figure 3.6: Shared responsibility in different scenarios. $G = \{g_1, g_2\}, R = \{r_1, r_2\}, O = \{o\}$.
(a) Shared responsibility is achievable if g_1 advertises o with cost 2 and g_2 advertises o with cost 1.
(b) Shared responsibility is not achievable.

3.3.1.2 Shared Responsibility

Definition 6. *Let $g_1, g_2 \in G$ be two gateways and $o \in O$ be an overlay network. g_1 and g_2 share responsibility for o if for all $r \in R$, there is a shortest path $P_1 : r \rightsquigarrow O$ with $g_1 \in P_1$ and there is a shortest path $P_2 : r \rightsquigarrow o$ with $g_2 \in P_2$.*

I deliberately chose a strict definition because, for cases where different routers have different shortest paths to o, load-balancing quality depends on the ratio of ingress traffic via the different edge routers. Using this definition, routers can forward packets to an overlay network using equal-cost multi-path (ECMP) routing. This could improve load balancing in cases where there are few overlay networks causing lots of traffic.

For $|R| > 1$, advertising costs for o on g_1 and g_2 may not exist. Figure 3.6 shows one such example. Therefore, I make the following decision:

Decision 4. *I use a single-responsibility model for ingress traffic.*

3.3.2 Egress Gateway Responsibility

I now look at responsibility models for egress traffic. In contrast to ingress traffic, we control egress traffic through the WiMoVE control plane.

3.3.2.1 Single Responsibility

In a steady state, single responsibility is easy to implement for egress traffic: The egress gateway serves as a WiMoVE endpoint and device with MAC address m. It answers ARP requests and neighbor solicitations for the device with m.

To fulfill R1, there has to be a process to migrate an egress gateway from one gateway g_1 to another gateway g_2. For this process, I need to look at the relationship between the MAC and IP addresses of the egress gateways.

I now look at the IP addresses that virtual gateways use inside the overlay networks. Let us assume that g_1 and g_2 different IP addresses for the egress gateway. On migration, the IP next hop of all devices has to be reconfigured. Manual reconfiguration is not an option. In an IPv4 scenario, the Dynamic Host Configuration Protocol (DHCP) [10] could be used

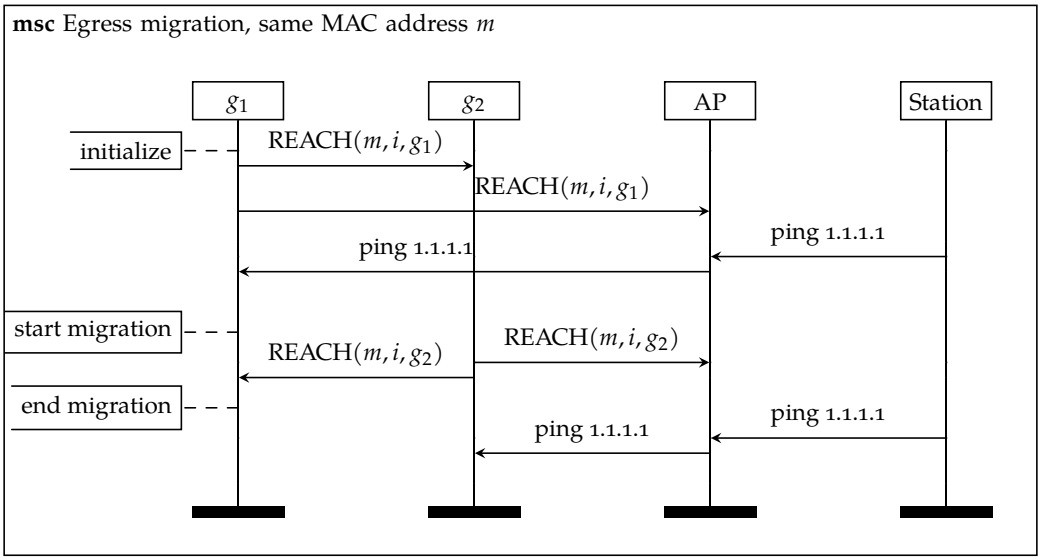


Figure 3.7: MSC of an egress migration with the same MAC address m on both gateways. Regular ARP messages and the RIDS are left out for brevity. The station periodically pings an Internet server at address 1.1.1.1.

for reconfiguration. Even though [36] specifies an extension for dynamic reconfiguration of DHCP clients, it is not implemented in common DHCP clients [12]. Therefore, using different IP addresses for the egress gateway on g_1 and g_2 is not an option. Thus, g_1 and g_2 use the same IP addresses.

For the MAC addresses, there are two options: Both egress gateways use the same MAC address or they use different MAC addresses.

Let us assume that g_1 and g_2 use the same MAC address during migration. Therefore, g_1 and g_2 both accept packets with destination MAC m. Figure 3.8 shows a message sequence chart (MSC) of a migration in this scenario. On migration, g_2 issues REACH(m, i, g_2) where i is the overlay network ID. The gateways then wait for propagation. Note that g_1 and g_2 need additional MAC addresses if serve as ingress gateways inside the overlay network, so e.g. ARP responses and NAs reach the correct gateway. The advantage of this approach is that it is transparent to other devices. This approach is used by VRRP [27].

Let us assume that g_1 and g_2 use different MAC addresses. Figure 3.7 shows an MSC of a migration in this scenario. During the migration, the ARP/Neighbor Discovery (ND) cache of all devices inside the overlay network must be updated. Therefore, g_2 needs to send gratuitous ARP packets and unsolicited NAs. g_2 also needs to handle lost updates; therefore, it needs to send periodic gratuitous ARP packets and unsolicited NAs.

In both cases, packets could be sent simultaneously to g_1 and g_2. If g_1 and g_2 used the same MAC address, this would be the case if a REACH has not been fully propagated. If g_1 and g_2 used different MAC addresses, this could happen if the gratuitous ARP and unsolicited NA have not been fully propagated. Sending packets simultaneously to g_1 and g_2 is not an issue since gateways can regularly decapsulate and forward packets.

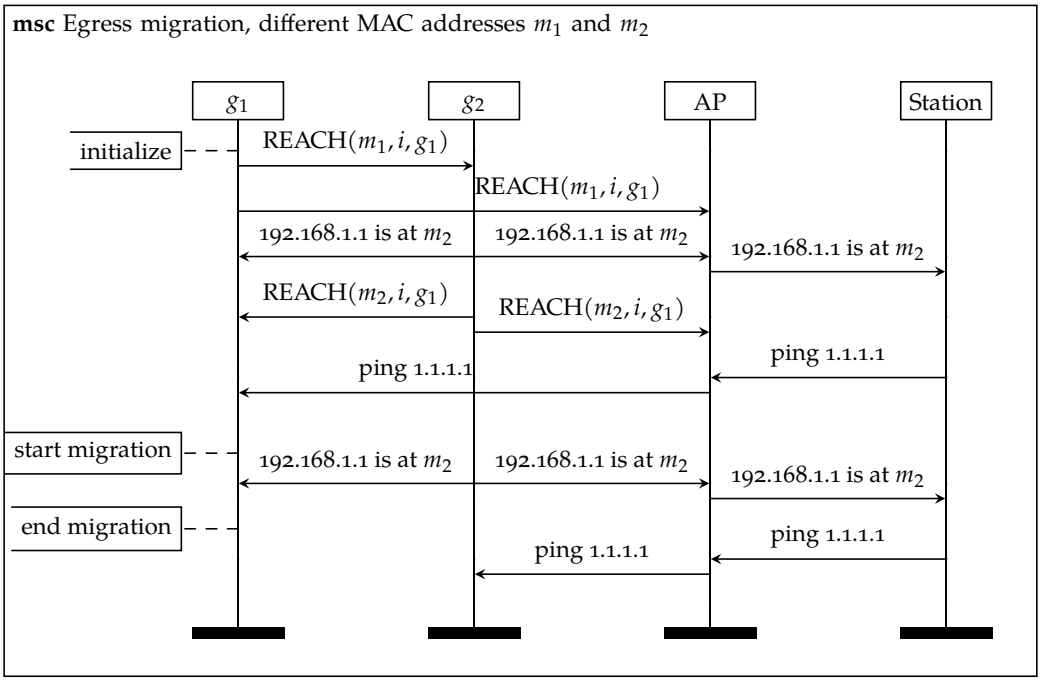


Figure 3.8: MSC of an egress migration with different MAC addresses m_1 and m_2 on the gateways. Regular ARP messages and the RIDS are left out for brevity. The station periodically pings an Internet server at address 1.1.1.1. The station uses 192.168.1.1 as its default route.

Using the same MAC address on g_1 and g_2 is transparent to the devices and requires sending fewer packets. Therefore, I implement the single-responsibility model for egress gateways by sharing MAC and IP addresses between the gateways.

3.3.2.2 Shared Responsibility

Shared responsibility for the egress gateway could be achieved by configuring different next-hop IPs for devices inside the overlay network. This has the disadvantage that on gateway failure, the next-hop IPs would have to be updated. I ruled out this option in the above section.

Therefore, I assume to use the same next-hop IP on all devices inside the overlay network. Gateways could send gratuitous ARP packets and unsolicited NAs, but these update the MAC address for the egress gateway for all devices. Therefore, all devices inside the overlay network must use the same MAC address for all egress packets. In the WiMoVE control plane, for a given overlay network, there is at most one endpoint associated with each MAC address. Therefore, changes to the WiMoVE control plane are required for shared egress responsibility. This seems like an unnecessary effort compared to the single-responsibility model. The advantage of such an approach would be improved load balancing when there are few overlay networks, similar to Section 3.3.1.2.

Overall, load-balancing quality depends on the load-balancing quality of ingress and egress traffic. I decided to use a single-responsibility model for ingress traffic. I assume that the ingress and egress data rates are in a similar order of magnitude. Then, even if egress

load balancing was significantly improved, the overall load-balancing quality would still be limited by the ingress load balancing. Therefore, the egress load-balancing improvement is not worth the control plane change and I make the following design decision:

Decision 5. *I use a single-responsibility model for egress traffic.*

3.4 Load Balancing

In Decision 3, I decided that the smallest unit of assignment is the overlay network. According to Decision 4 and Decision 5, each overlay network needs to be assigned exactly one ingress and egress gateway. In this section, I will look at metric-based and randomized load-balancing algorithms. I assume that ingress and egress gateways for o are assigned to the same gateway, although this is not strictly necessary.

The advantage of random assignment is that it is a simpler approach: There is no need to report metrics and the assignment algorithm can be much simpler, giving more implementation options to a control plane. But does random assignment effectively load-balance in WiMoVE? This depends on the distribution assumptions for data rates in the overlay networks. I will look at an example scenario and analyze the load balancing achieved by a randomized approach.

I use the Chaos Communication Camp 2019 as an example of a large Wi-Fi since the organizers published data about their infrastructure. During the camp, there was a maximum of 5785 stations with a maximum Wi-Fi data rate of 3.596 Gbit s^{-1} [24]. This is an average data rate of 621.6 kbit s^{-1} per station.

Let $n \in \mathbb{N}$ be the number of stations and $b \in \mathbb{N}$ be the number of overlay networks. Data rates in real networks are often log-normally distributed [1]. Even though a log-normal distribution is unbounded, Wi-Fi data rates are limited. The latest standard, 802.11ax (Wi-Fi 6E), could theoretically achieve a data rate of 9607.8 Mbit s^{-1}, but real-world speeds are often limited to about 1600 Mbit s^{-1} [17, 20]. Therefore, I use truncated log-normal random variables with unit Mbit s^{-1} to model the stations' data rates:

$$X_1, \ldots, X_n \sim_{\text{iid}} \text{Lognormal}(\mu, \sigma^2, 0, 1600).$$

There is no data on the variance of the data rates. Therefore, I use different parameters $\sigma \in \mathbb{R}^+$ and calculate $\mu \in \mathbb{R}$ such that $\mathbb{E}[X_1] = 0.6216$. Figure 3.9 shows probability density functions for different values of σ.

I now want to sample from the load distribution. I assume that there are two gateways g_1 and g_2, that overlay networks are assigned randomly to the gateways, and stations are assigned randomly to overlay networks. Since the assignment of overlay networks to gateways is uniform, the number of overlay networks k at g_1 follows a binomial distribution:

$$k \sim \text{Bin}\left(b, \frac{1}{2}\right)$$

Therefore, the probability that a station is assigned to g_1 is $\frac{k}{b}$ and the number of stations Y at g_1 follows a binomial distribution:

$$Y \sim \text{Bin}\left(n, \frac{k}{b}\right).$$

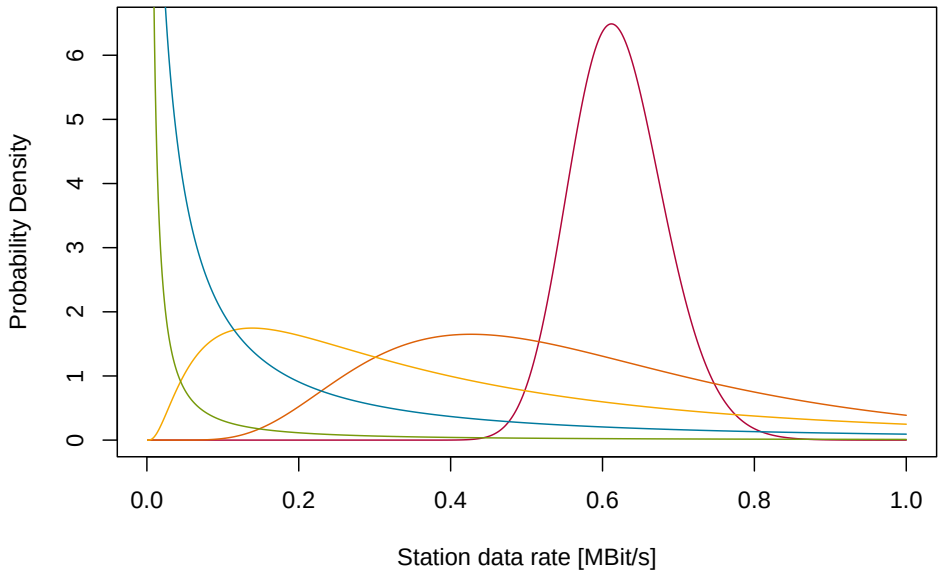


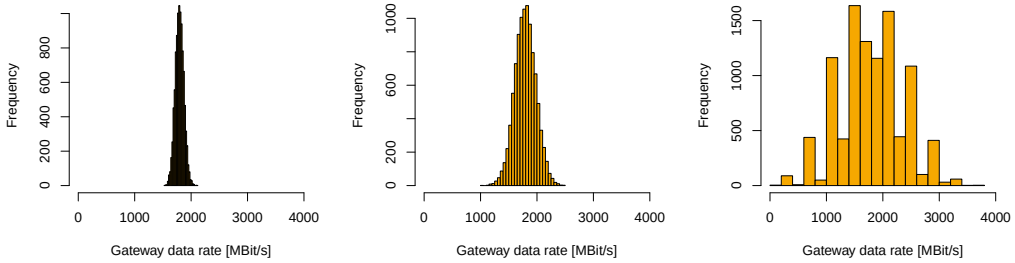
Figure 3.9: Probability densities of data rates at stations for $\sigma = 0.1$ (red), $\sigma = 0.5$ (orange), $\sigma = 1$ (yellow), $\sigma = 2$ (blue) and $\sigma = 4$ (green).

Then, the load at g_1 is $X = \sum_{i=1}^{Y} X_i$. I expect that for distributions that are more heavily right-skewed, randomized load balancing performs worse. Therefore, I simulated the load on g_1 for $\sigma \in \{1, 2, 4\}$ with $b \in \{10, 100, 1000\}$. For each scenario, I sampled the load distribution 10 000 times.

Figure 3.10 shows histograms of the simulated data rates at g_1 for $\sigma = 1$. We observe that for ≥ 100 overlay networks, the distribution is symmetric and has a small standard deviation compared to its mean. Therefore, both gateways will have similar data rates. Thus, randomized load balancing is a good fit for this scenario. The same effect can be observed for $\sigma = 2$ in Figure 3.11. Figure 3.12 shows the histograms for $\sigma = 4$. There, for all numbers of overlay networks, the distribution is much more right-skewed. Also, the distribution has a large standard deviation compared to its mean. Therefore, both gateways could have very different data rates. In the cases of $\sigma = 4$ or $b = 10$, random assignment provides poor-quality load balancing.

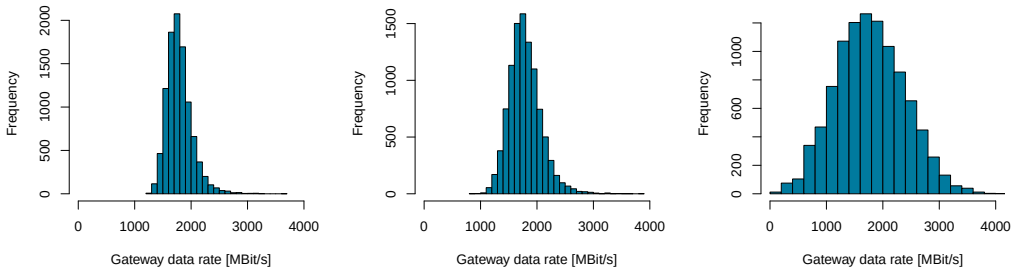
WiMoVE is designed for large numbers of overlay networks. We observed that randomized assignment provides good load balancing in many scenarios while allowing for a much simpler control plane. Therefore, I make the following design decision:

Decision 6. *I assign ingress and egress gateways randomly and uniformly to the gateways.*



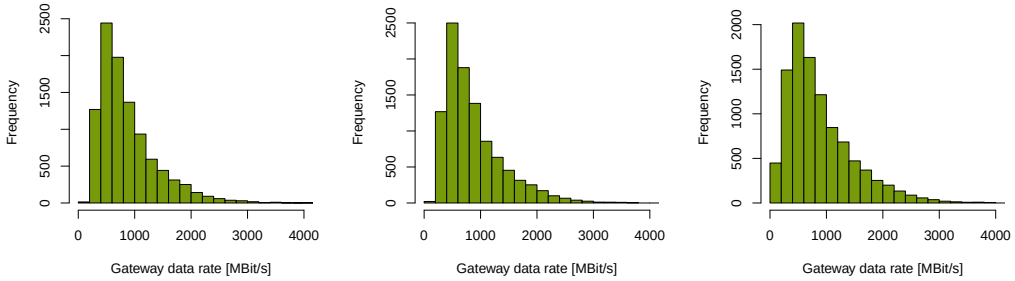
(a) $b = 1000$ **(b)** $b = 100$ **(c)** $b = 10$

Figure 3.10: Simulated data rates for $\sigma = 1$ for varying numbers of overlay networks $b \in \mathbb{N}$.



(a) $b = 1000$ **(b)** $b = 100$ **(c)** $b = 10$

Figure 3.11: Simulated data rates for $\sigma = 2$ for varying numbers of overlay networks $b \in \mathbb{N}$.



(a) $b = 1000$ **(b)** $b = 100$ **(c)** $b = 10$

Figure 3.12: Simulated data rates for $\sigma = 4$ for varying numbers of overlay networks $b \in \mathbb{N}$.

3.5 Ingress Gateway Assignment

I will now look at how to implement random assignment of ingress gateways to gateways. I discuss three implementation possibilities:

- *Random costs*: Use only IGP costs, set them randomly.
- *Intelligent costs*: Use an additional control plane that coordinates the mapping of ingress gateways to gateways. Set costs accordingly on each gateway.
- *Semi-intelligent costs*: Do not use any runtime control plane, but avoid cost ties during configuration.

One aspect relevant to all assignment implementations is whether they need a control plane at runtime. This comes from a practical standpoint: An additional control plane creates overhead. Assuming that the control plane is not integrated into the routing software, it reconfigures the routing software at runtime. For lots of overlay networks, this could become costly because of the required inter-process communication as well as the need to re-announce routes after reconfiguration. I provide data on this in Chapter 4. For now, I try to avoid an additional control plane.

3.5.1 Random Costs

According to Section 3.3.1.1, there is a set of cost levels C that can be used to achieve single responsibility. Let $o \in O$ be an overlay network. If each gateway picks a unique cost $c \in C$ for o, then there is a gateway that is singly-responsible for o. I look at the following strategy: for each gateway and overlay network, I pick an advertisement cost independently and uniformly from C. This gives us failover: When a gateway fails, the IGP will recognize the failure and all routes are updated. Using random costs does not require an additional control plane and reduces overhead.

One potential issue are multiple gateways picking the same cost for an overlay network. In that case, traffic could be distributed arbitrarily among the tied gateways.

Let us now look at a gateway g_1 that always receives all the load in the event of a tie. g_1 serves as an upper bound for the load of a gateway in the event of a tie. If I set $s = |C|$ and $n = |G|$, the probability that an overlay network is assigned to g_1 is

$$\sum_{k=1}^{s} \left(\frac{k}{s}\right)^{n-1} \frac{1}{s}.$$

As $s \to \infty$, this term converges to $1/n$. Since it converges faster for higher n, I look at $n = 2$ as the worst case. In that case,

$$\sum_{k=1}^{s} \left(\frac{k}{s}\right)^{n-1} \frac{1}{s} = \sum_{k=1}^{s} \frac{k}{s^2} = \frac{1}{2} + \frac{1}{2s}.$$

Figure 3.13 shows the probability of assigning an overlay network to g_1 for different numbers of cost levels s. I observe that for a moderate number of cost levels, the assignment is almost uniform. Therefore, for many networks, this may be a viable option to implement random assignment. Using random costs requires no run-time control plane, reducing overhead.

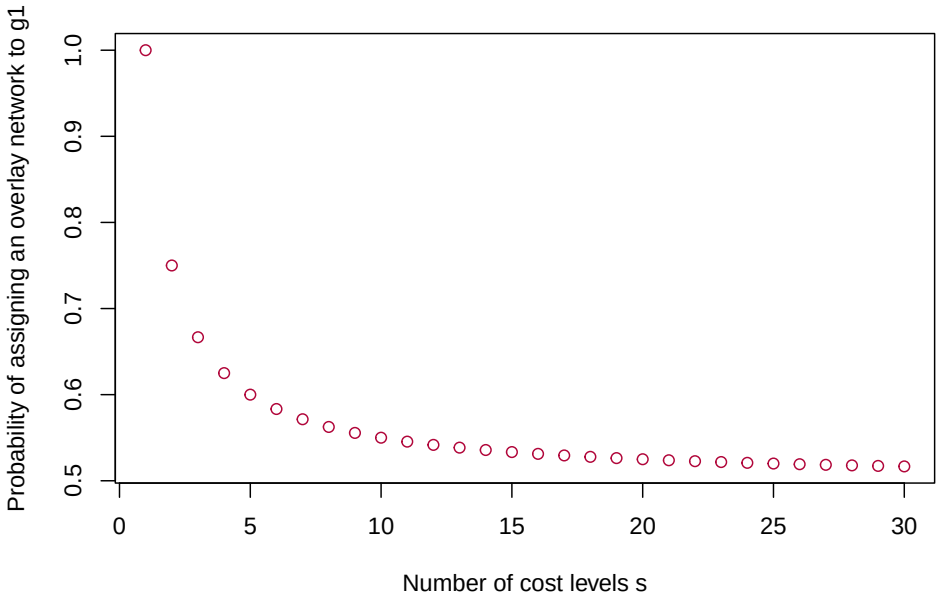


Figure 3.13: Probability of assigning an overlay network o to g_1 with s cost levels. I assume a network of two gateways, g_1 and g_2 where g_1 is assigned o when costs are tied between g_1 and g_2.

Still, some scenarios could only have a low number of cost levels. Therefore, I will look at other options.

3.5.2 Intelligent Costs

Another option is to use an additional control plane. This control plane must determine which gateway is responsible for which ingress gateway. If the gateway g is responsible for overlay network o, g_1 advertises a route to o with the minimum possible cost. All other gateways $g' \neq g$ advertise o with the maximum possible cost.

The control plane could be implemented by assigning each gateway a unique ID. The gateways then use a peer discovery protocol to announce their IDs. HRW hashing is performed on the unique IDs. As a result, for each overlay network $o \in O$, all gateways agree on the responsible gateway for o.

The advantage is that this assignment scheme can deal with an arbitrary number of cost levels. Also, the number of messages between the gateways is independent of the number of overlay networks. Still, compared to randomly choosing costs, the additional control plane seems unnecessary. Therefore, I look at a combination of the two approaches.

3.5.3 Semi-Intelligent Costs

For each overlay network, the costs should be disjoint without needing a runtime control plane. Let $n \geq |G|$ be the maximum number of gateways in the system. I require $|C| \geq n$. Let the cost levels be $C_1, C_2, \ldots, C_{|C|}$. For each overlay network $o \in O$, I uniformly and independently choose a random permutation $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$. Then, gateway g_i advertises o with cost $C_{\pi(i)}$.

For each overlay network $o \in O$, this assures that the costs are disjoint. Therefore, there is a gateway that is singly-responsible for o.

As an example, assume that $n = 3$ with gateways g_1, g_2, g_3 and $C_1 = 5, C_2 = 10, C_3 = 15$ with $C = \{C_1, C_2, C_3\}$. Let $o \in O$ be an overlay network. I now choose a random permutation $\pi : \{1, 2, 3\} \to \{1, 2, 3\}$. Let $\pi(1) = 2, \pi(2) = 1, \pi(3) = 3$. Then, g_1 advertises o with cost $C_2 = 10$, g_2 advertises o with cost $C_1 = 5$ and g_3 advertises o with cost $C_3 = 15$.

This makes the gateway choice uniform and independent, even if only a subset of gateways is active. But how do all gateways agree on the permutation π?

I assume that each gateway has a unique gateway ID $k \in \{1, \ldots, n\}$ and knows n. Let o be an overlay network with ID i. Again, I use HRW hashing.

Each gateway now computes the set of gateway hashes $H = \{h(i, k') \mid 1 \leq k' \leq n\}$. If $|H| < n$, the gateway deterministically alters h and recomputes H until $|H| = n$. It then finds the rank of $h(i, k)$ among H and chooses the cost level from C with the same rank.

I use the same example as previously. Let i be the ID of overlay network o. We assume that the image of h is $\{1, \ldots, 1000\}$ and $h(i, 1) = 375, h(i, 2) = 89$ and $h(i, 3) = 742$. From these hashes, each gateway can determine π and the cost used to advertise o.

Overall, this assignment scheme does not require a control plane at runtime while providing uniform load balancing even for small cost levels, as long as there are at least as many cost levels as gateways. Therefore, this is the best assignment scheme out of the three options:

Decision 7. *To implement random assignment of overlay networks to gateways, gateways advertise routes into overlay networks using semi-intelligent costs.*

3.6 Egress Gateway Assignment

Implementing random assignment of overlay networks to gateways is more complicated in the egress case than in the ingress case. Random assignment could be implemented using random costs, but there are no costs in the WiMoVE control plane. I will sketch how costs could be integrated in the next section. Afterward, I will discuss how to implement random assignment without costs.

3.6.1 Random Costs

One option to randomly assign overlay networks to gateways would be to introduce costs to the WiMoVE control plane. Then, an assignment scheme similar to Section 3.5.3 could be used. Therefore, I look at how costs could fit into the WiMoVE control plane.

I define a maximum cost $\hat{c} \in \mathbb{Z}^+$ and define the costs as $C = \{1, \ldots, \hat{c}\} \cup \{\infty\}$. I then change the definition of reachability information. For each VNI and MAC address, reachability at multiple endpoints may be defined, with different costs. Therefore, I define *reachability information* as the following partial function:

$$r : \text{MACAddresses} \times \text{OverlayNetworkIDs} \times \text{EndpointAddresses} \to C$$

Instead of separate REACH and UNREACH messages, I only use REACH messages. Let m be the MAC address of a device, i an overlay network ID, e an endpoint, and $c \in C$ a cost. $\text{REACH}(m, i, e, c)$ updates r to r' such that $r'(m, i, e) = c$. Previous UNREACH messages are implemented by setting c to ∞. When a station with MAC address m connects to an overlay network with ID i at AP e, the AP issues $\text{REACH}(m, i, e, c_0)$, and for each $e' \neq e$: $\text{REACH}(m, i, e', \infty)$ where $c_0 \in C$.

In an overlay network with ID i, L2 packets to MAC address m are forwarded to the endpoint e where

$$e = \arg\min_{e' \in E} r(m, o, e').$$

This control plane implements all features of the original WiMoVE control plane.

Using this control plane, each gateway can advertise a MAC address m in each overlay network with a random cost. This results in failover and load balancing for the gateways, similar to Section 3.5.1. The advantage of this approach is that it makes each gateway stateless. The approach also removes the need for a control plane which could reduce failover latency.

3.6.2 Intelligent Costs

One option would be to use an additional control plane. This control plane decides which gateway is responsible for which ingress gateway. This is the same control plane as described in Section 3.5.2. For WiMoVE, "costs" are binary, meaning that gateways advertise routes with a REACH and withdraw routes with an UNREACH.

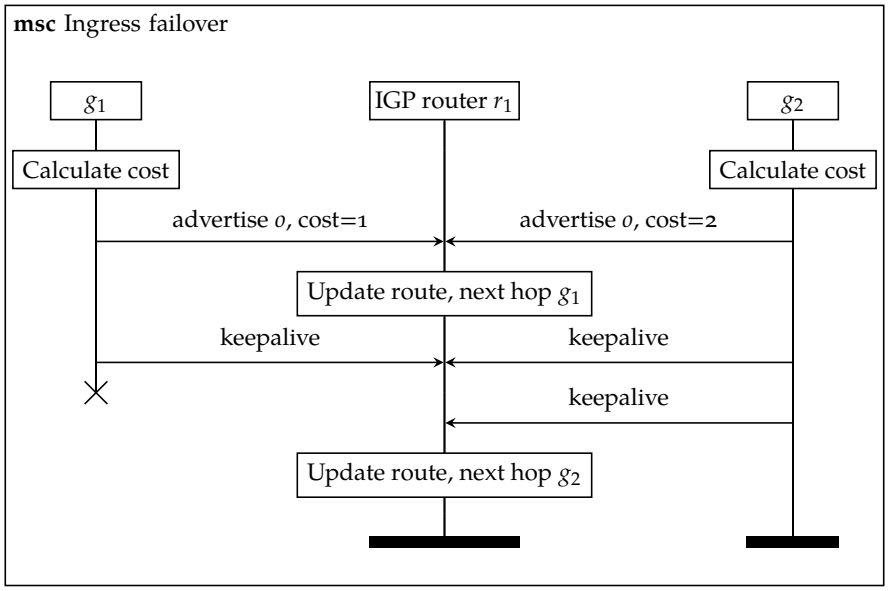


Figure 3.14: MSC for an ingress failover of overlay network o from gateway g_1 to g_2. All messages are from a generic IGP. r_1 is adjacent to g_1 and g_2. Initially, packets are routed over g_1. After failure of g_1, the IGP router recalculates its routes, routing overlay network traffic over g_2.

An advantage is that the approach does not require any changes to the WiMoVE control plane. I estimate that implementing the new control plane is relatively straightforward while implementing changes to the WiMoVE control plane is more involved. The disadvantage is that using intelligent costs results in runtime reconfiguration overhead of the routing software. Still, I make the decision to introduce an additional control plane:

Decision 8. *I use an additional control plane paired with intelligent costs to determine which gateway is responsible for each overlay network.*

3.7 Architecture

I now summarize the design decisions into a single architecture.

Each gateway advertises all overlay networks via the IGP with semi-intelligent costs. These costs are computed at gateway startup via the HRW-based method described in Section 3.5.3. Ingress failover is provided via the failover mechanism of the IGP. Figure 3.14 shows an ingress startup and failover process.

In my proposed system, gateways serve as WiMoVE devices and endpoints as well as IGP routers. Gateways send periodic advertisement messages containing their ID to notify other gateways of their presence. They use the information to determine egress gateway responsibility via HRW hashing. After responsibility changes, they issue the appropriate REACH and UNREACH messages into the WiMoVE control plane. Figure 3.15 shows an egress startup and failover process.

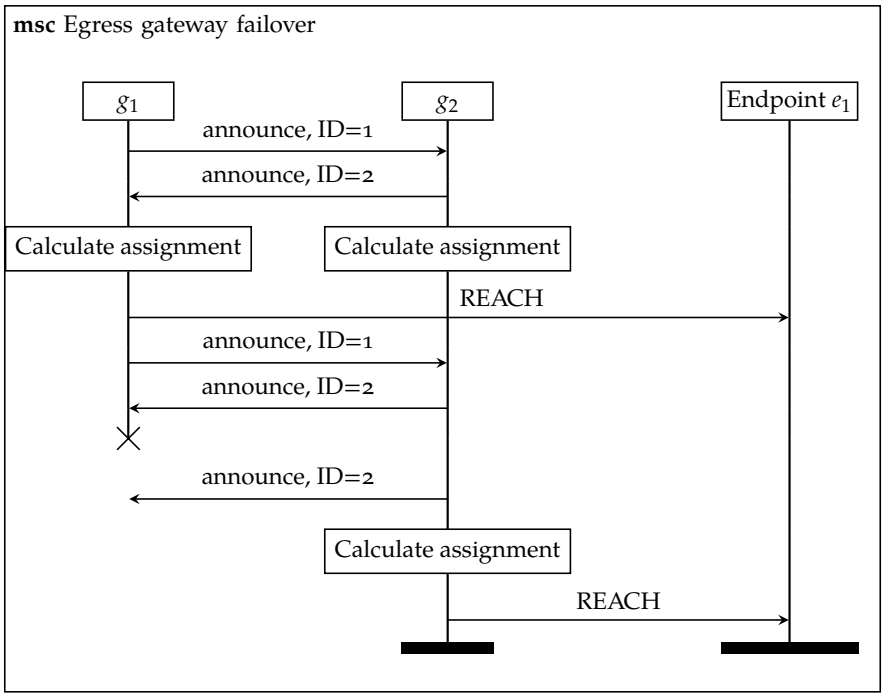


Figure 3.15: MSC of an egress gateway failover from gateway g_1 to g_2. For simplification, I left out the RIDS relaying messages.

4 Evaluation

In this chapter, I evaluate different implementations. All implementations fulfill the functional requirements I described in section 1.2. Each implementation implements randomized load balancing using a different combination of an ingress and egress control plane. Table 4.1 shows a summary of the control plane combinations.

I first discuss the common evaluation scenario. For each implementation, I describe the reasons for looking at the implementation and describe its behavior. I then show measurement results and analyze my findings. To compare the implementations, I use failover latency as the key metric, as described in Section 1.3.

4.1 Scenario

There are a few requirements that my evaluation method had to fulfill:

- *Realistic*: The evaluation scenario should closely replicate a potential WiMoVE network structure.
- *Repeatable*: Measurements should be repeatable without requiring manual infrastructure configuration.
- *Simple*: There should be as few components as possible.
- *Small Implementation*: Little custom code should be required, instead I try to reuse existing projects.

To be as realistic as possible, I started with the proof-of-concept (PoC) implementation described in the original WiMoVE paper [33]. The implementation uses Border Gateway Protocol (BGP) Ethernet Virtual Private Network (EVPN) as the control plane and Virtual Extensible LAN (VXLAN) as the data plane.

As the IGP, I decided to use Open Shortest Path First (OSPF) because of its wide use in real-world networks and its good software support [26]. Since I focused on the gateway implementations and their failover times, I decided to replace any wireless links with wired links. Therefore, using the custom AP daemon *wimoved* from the WiMoVE PoC was unnecessary. For easier and more repeatable measurements, I decided to virtualize all network

Implementation	Ingress Control Plane	Egress Control Plane	Communication
wmgwd	intelligent costs	intelligent costs	etcd
VRRP	semi-intelligent costs	intelligent costs	VRRP
wmgwd2	semi-intelligent costs	intelligent costs	custom

Table 4.1: Control-plane combinations of the three implementations

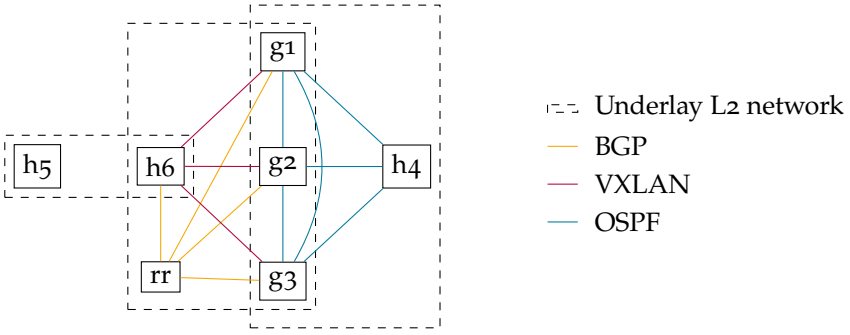


Figure 4.1: Topology used in the implementation tests. L2 networks on the underlying infrastructure are represented by dashed rectangles, network protocols are represented by colored lines.

hosts inside the network emulator *Mininet* [22]. Using Mininet, I could programmatically create and tear down virtual network infrastructure.

As the routing software, I used FRRouting (FRR) [14]. FRR already implements BGP EVPN, OSPF, and VRRP. This reduced my implementation size as I could use existing network protocol implementations.

For all tests, I used a topology that mimics the network topology in WiMoVE with three gateways, g1, g2, and g3. The gateways share an L2 network with another IGP router, h4. They also share an L2 network with an AP, h6, and a BGP EVPN route reflector, rr. The host h5 represents a wireless station that is connected to h6. The gateways and h4 exchange routes via OSPF. The gateways and h6 serve as WiMoVE endpoints. This means that they speak BGP EVPN and decapsulate and encapsulate VXLAN packets. Figure 4.1 shows a graphical representation of the network topology.

In all tests, I make sure that g1 is responsible for the overlay network of h5. To simulate the failure of g1, I then set the state of all its network interfaces to *down*. This stops it from communicating with all other hosts.

To determine the ingress and egress failover time, h5 periodically sends packets to h4 and h4 periodically sends packets to h5. These packets contain sequence numbers, so lost packets can be detected by looking at the sequence numbers of received packets at h5 and h4. If hosts detect missing packets, they can look at the timestamp t of the last packet before the missing packets and at the timestamp t' of the first packet after the missing packets. $t' - t$ then gives an estimate of the failover time. The smaller the packet sending interval, the more exact the estimate will be. I send Internet Control Message Protocol (ICMP) requests at an interval of 10 ms since this provides good resolution while not overloading the test setup.

Since I used a network emulator, jitter is only caused by the operating system. Therefore, I assume jitter to be small compared to the failover times, and $t' - t$ provides an approximate upper bound for the failover time. Thus, I get an upper bound on the ingress failover time through the packet capture on h5 and an upper bound on the egress failover time through the packet capture on h4. By repeating the measurements, I get data on the distribution of ingress and egress failover times.

FRR uses a default hello interval of 10 s and a default dead interval of 40 s. These are relatively long compared to the rest of the failover time. The high hello interval also causes

high variance in the measurements. The same is true for BGP where the default keepalive interval is 60 s and the default hold time is 180 s. Therefore, I decided to use shorter timers. For OSPF, I used a hello interval of 1 s and a dead interval of 3 s. For BGP, I used a keepalive timer of 1 s and a hold timer of 3 s.

The main way in which measurements failed was FRR hanging or crashing. Even if FRR was restarted after a crash, information entered into the configuration interface could be lost and connectivity between h4 and h5 will never be restored. Therefore, I had to introduce a timeout on the maximum time the measurement process waits for h4 and h5 to restore connectivity. Since I expected maximum failover times of 60 s, I set the timeout to 300 s.

4.2 Wmgwd

The goal of the first implementation was to evaluate how intelligent costs as outlined in Section 3.5.2 and Section 3.6.2 could be implemented. The core idea is to explicitly manage which gateway advertises which OSPF route and which gateway advertises which egress gateway within WiMoVE. For simplicity, a gateway advertises an ingress gateway for an overlay network o if and only if it advertises an egress gateway for o.

For that purpose, I wrote the daemon *wmgwd*[1] Wmgwd integrates into FRR by configuring it via its command-line interface *vtysh*.

To advertise or withdraw routes for an ingress gateway, wmgwd changes OSPF link costs through vtysh. To issue a REACH or UNREACH for an egress gateway, it uses route maps. In FRR, route maps can be used to control advertised BGP routes and their parameters. Route maps can be used to match on the overlay network ID of a route and decide whether to advertise the route. By reconfiguring them at runtime, wmgwd controls whether a FRR instance advertises BGP routes for a given overlay network.

Another design decision for wmgwd was to have all egress gateways for an overlay network o use the same IP, but different MAC addresses. Therefore, ARP and NA must be disabled by default on the egress network interfaces. ARP and NA are enabled only for interfaces of overlay networks for which the gateway is responsible.

Let us consider a migration process of an ingress and egress gateway from gateway g_1 to g_2. The following steps need to be performed:

1. Issue REACH to o on g_2 and wait for propagation.
2. Announce route to o on g_2 and wait for propagation.
3. Withdraw route to o on g_1 and wait for propagation.
4. Enable ARP/NA on g_2.
5. Disable ARP/NA g_1.
6. Send gratuitous ARP and unsolicited NA from g_2 into o and wait for propagation.
7. Issue UNREACH to o on g_1 and wait for propagation.

The order follows from the following constraints:

1. One gateway must always be able to answer ARP requests and neighbor solicitations.

[1] See `https://github.com/rgwohlbold/wmgwd/tree/e76a6dbfcc3d1e9af1f33b3f240814774816039b`, last accessed 2023-07-21.

2. Gratuitous ARP packets must only be sent from a gateway g when ARP is enabled on g and disabled on the other gateway (same for NA).
3. There must always be an IGP route into o via g_1 or g_2.

Therefore, the migration process must be coordinated. The process is most easily represented by a state machine which is shown in Figure 4.2.

For easier programming, I made the design decision to use a distributed database to implement the state machines. Since the daemon only stores the state of each overlay network, a key-value store is sufficient.

For wmgwd to be safe, the key-value store needs to provide certain consistency guarantees. By definition of a state machine, the state history for each overlay network needs to be ordered. Therefore, wmgwd needs a sequentially consistent database. The key-value store also needs to support failover: If one database instance fails, but all others still operate, the system needs to continue to make progress. Therefore, I decided to use etcd as the distributed database [4].

I use one database instance on each gateway. The daemon communicates with the local replica of the distributed database. Figure 4.3 shows a diagram of the system components.

I implemented failure detection using time-bound leases within etcd. Wmgwd processes ask etcd for a lease for a specified amount of time. Each process creates a key and attaches a lease which it periodically renews. Once the lease expires, the key is deleted. If a gateway fails, the wmgwd process is unable to extend the lease, and the corresponding key is deleted. Other processes notice the deletion and transition the affected overlay networks to *Unassigned*.

Wmgwd assigns overlay networks randomly to gateways using consistent hashing. The reason for this is a practical one: I did not know about HRW hashing when implementing wmgwd. For wmgwd, there is little difference since the gateways do peer discovery via etcd anyway.

To implement consistent hashing, the processes generate random IDs and write them into etcd. Each gateway can then run the assignment algorithm locally to determine differences between current assignments and target assignments. Gateways can then start an assignment process by transitioning an overlay network into *FailoverDecided* or *MigrationDecided*.

Because of etcd, there are now two failover scenarios: The etcd instance on the failing gateway could be a follower or a leader. In the latter scenario, after the failure, the rest of the cluster performs a leader election. During this time, the database is read-only. In this scenario, failover times are larger than in the former scenario. To control for these differences, I ensured that the database instance on the failing gateway is always a follower. This makes the scenario less realistic, but significantly reduces measurement variance.

For my measurement setup, I used varying numbers of overlay networks. I evenly spaced the numbers of overlay networks on the log scale. As the daemon is proof-of-concept software, not all measurements are successful (see Section 4.1). Therefore, I also look at the number of successful measurements in my analysis.

4.2.1 Results

I conducted measurements in the range between 13 and 400 overlay networks. Table 4.2 shows the number of measurements, the number of successful measurements and the

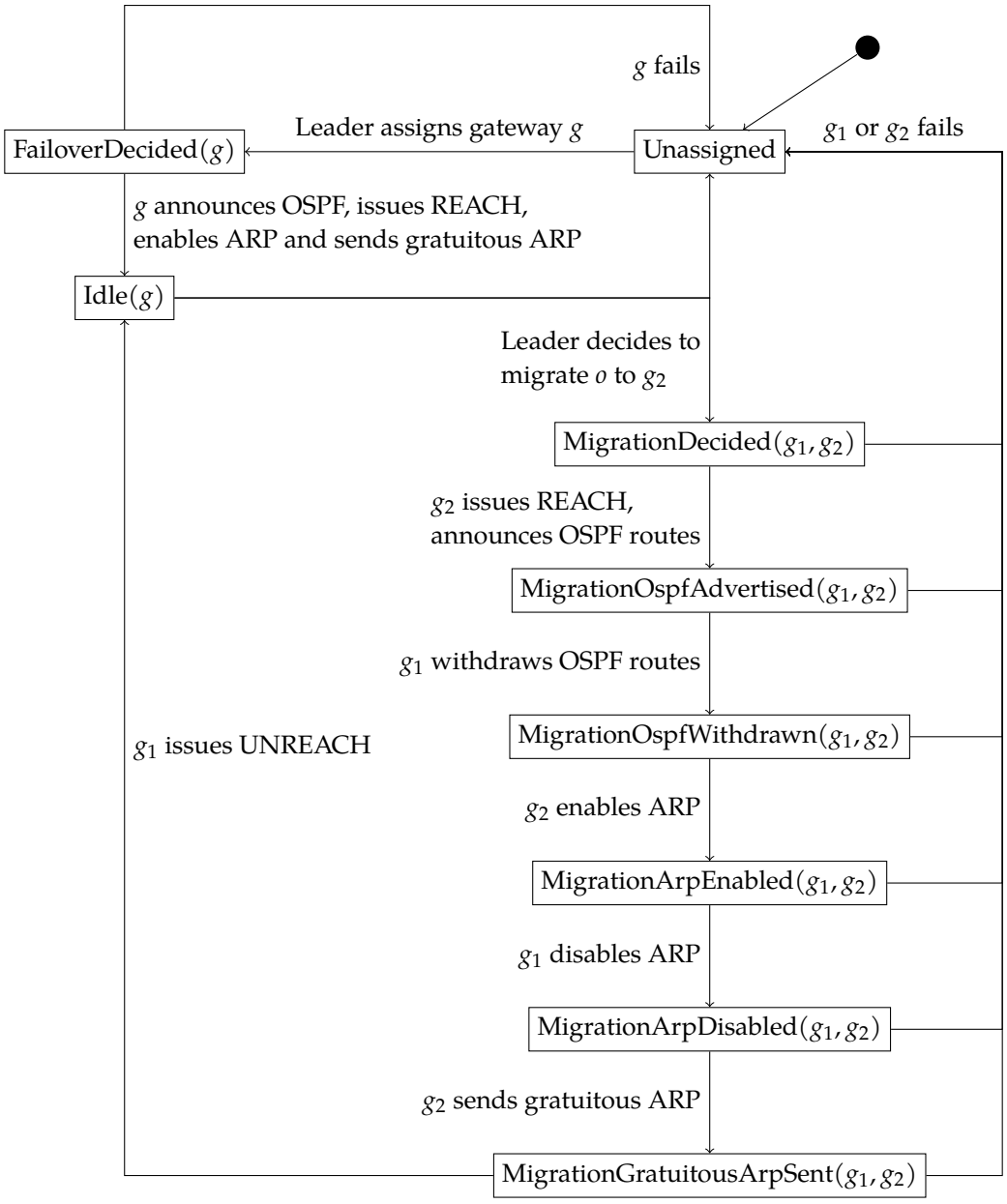


Figure 4.2: State machine of wmgwd for an overlay network o

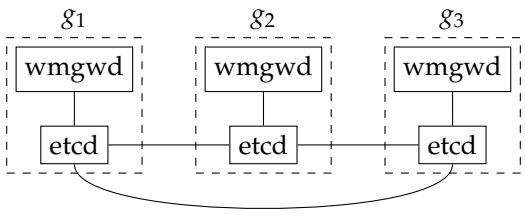


Figure 4.3: System components in the wmgwd-based implementation

Overlay networks	13	18	25	35	50	71	100	141	200	282	400
Measurements	131	131	131	131	131	131	191	136	92	91	66
Successes	114	130	124	130	126	127	157	111	87	81	38
Success rate	0.87	0.99	0.95	0.99	0.96	0.97	0.82	0.82	0.95	0.89	0.58

Table 4.2: Number of measurements for different numbers of overlay networks in the wmgwd-based implementation

success rate for all numbers of overlay networks. For ≥ 400 overlay networks, measurements frequently ran into the timeout of 300 s. Therefore, I only analyze the data for ≤ 282 overlay networks.

I first look at the egress failover times. Figure 4.4 shows boxplots of the egress failover times for different numbers of overlay networks. The plot shows that there is a monotonic relationship between the number of overlay networks and the ingress failover time. During failover, for each failed overlay network, wmgwd transitions the state machine and reconfigures FRR. Therefore, I have the hypothesis that there is a linear relationship between the number of overlay networks X and the ingress failover time Y. I use a linear model to fit the data, meaning that $Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$ where $\epsilon_i \in \mathbb{R}$ is the regression error. The coefficients β_0 and β_1 are significant at $\alpha = 0.95$ with both p-values $< 2 \times 10^{-16}$. Figure 4.5 shows a scatter plot and the regression line. The plot shows that a linear model is a poor fit for the data. The relationship between the number of overlay networks and the egress failover time seems to be superlinear. I look at a possible reason in Section 4.4.

I now look at the ingress failover times. Figure 4.6 shows boxplots of the ingress failover times for different numbers of overlay networks. With an increasing number of overlay networks, the median and variance of the egress failover times seem to increase. Like before, even though a linear model between the number of overlay networks and ingress failover times results in significant coefficients, it is a bad fit.

One additional interesting observation is that for small numbers of overlay networks, there are outliers ≈ 3 s above the median. I will look at potential reasons in the next section.

4.2.2 Analysis

The data shows that this approach does not scale to 400 or more overlay networks. According to the design of WiMoVE, there should be one overlay network per user to effectively reduce broadcast traffic [33]. Assuming a Wi-Fi system with thousands of users, this implementation is insufficient.

I investigated the reasons for this poor performance. The critical part is the communication between wmgwd and FRR. It seems like the FRR configuration interface is not designed for frequent programmatic changes, but instead for one-off manual changes. FRR extensively validates and re-applies the configuration. If this takes too long, the process is killed and a new instance is started. This leads to vtysh hanging for wmgwd. Instead of configuring FRR via vtysh, there is also a remote procedure call (RPC) configuration interface. Since this interface uses the same validation and application routines, my hypothesis is that using the RPC interface would not change the configuration latency. Therefore, if a program separate from FRR is used, it must limit the use of the configuration interface to avoid performance problems.

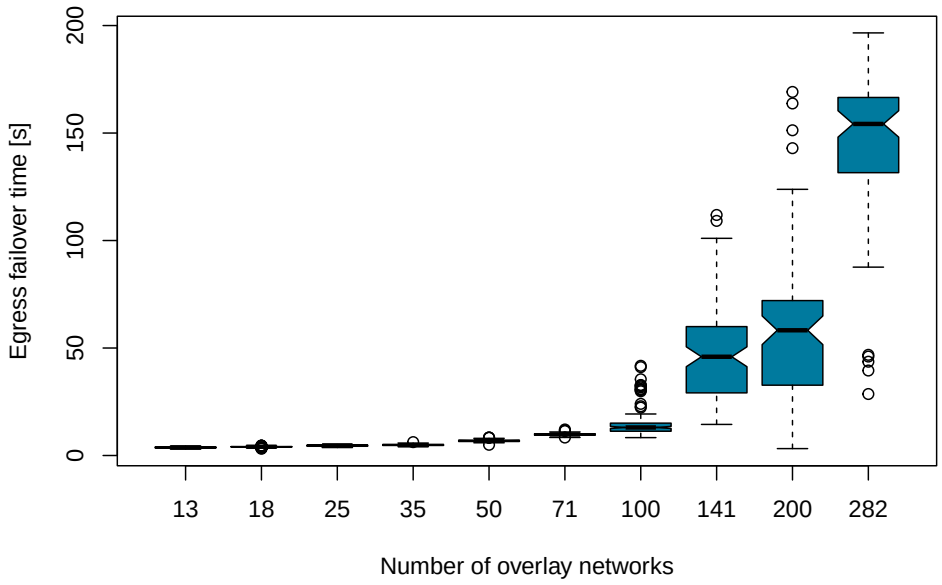


Figure 4.4: Egress failover times for different numbers of overlay networks in the wmgwd-based implementation. 95% confidence intervals for the medians are marked by the notches on the plot.

One option is to only use functionality already integrated in FRR or extend it with the necessary features. I show one possible implementation in Section 4.3.

If an extra daemon besides FRR is used, it must communicate with FRR in some other way. One example would be to configure FRR once and then communicate only by setting network interfaces down or up. Setting an interface up or down triggers an event in FRR which sends update link-state advertisements (LSAs) via OSPF. Since this happens whenever a link is connected or disconnected to an OSPF router, this is a standard code path. I show one possible implementation in Section 4.4.

I also looked at the outliers ≈ 3 s above the median for small numbers of overlay networks. When FRR receives many LSAs in a short timeframe, it defers running the shortest-path algorithm to save processing time. The mechanism uses timeouts that are incremented on each received LSA. This explains the outliers: In some cases, the relevant LSA arrives first, triggering an immediate shortest-path calculation. In other cases, it arrives a bit later and FRR waits for the throttling timer to expire. The failover latency could be made more consistent by changing the throttling timeouts in FRR. As I do not use intelligent costs as the ingress control plane for the other implementations, I do not further look into the issue.

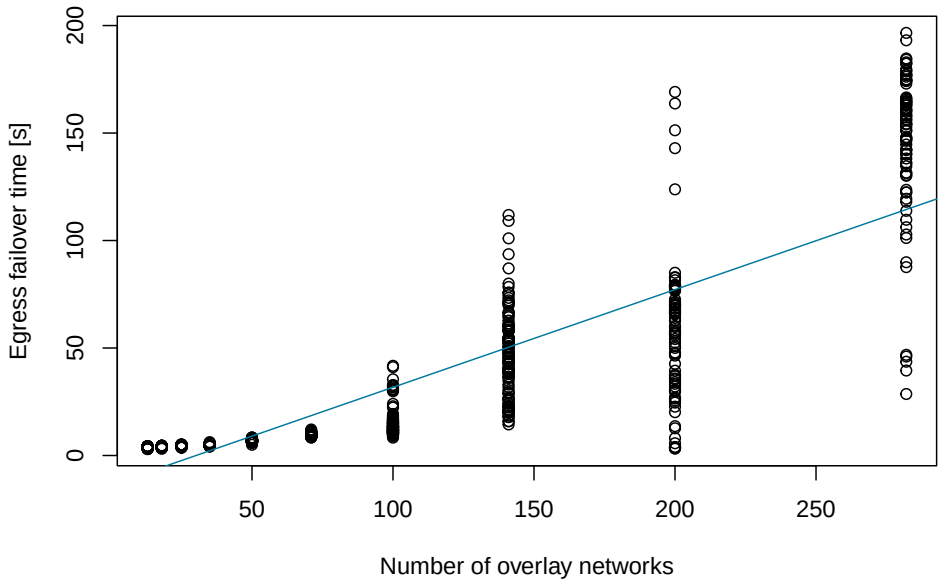


Figure 4.5: Linear regression of failover times for different numbers of overlay networks in the wmgwd-based implementation. The linear model is not a good fit for the data.

4.3 Virtual Router Redundancy Protocol

According to the last section, live reconfiguration of FRR is not an option for more than 100 overlay networks. Therefore, the control plane that manages assignments must be better integrated into FRR.

In Section 2.2, I looked at VRRP. Since VRRP is already implemented in FRR, I wanted to try using it as the egress control plane.

In VRRP, a *virtual router* is multi-homed between multiple *physical routers*. According to the previous definitions, a virtual router corresponds to an ingress gateway and a physical router to a physical gateway. Therefore, there must be one virtual router per overlay network. VRRP supports priorities: the physical router with the highest priority becomes the master router for a given overlay network. At startup, priorities can be assigned using the mechanism described in Section 3.5.3. This results in uniform assignment of overlay networks to physical gateways while eliminating runtime reconfiguration. Still, this approach uses intelligent costs, since it does not introduce costs into WiMoVE.

I left the VRRP hello interval at its default, 1.5 s. A physical router is declared dead after three times the hello interval, i.e. 4.5 s.

The VRRP implementation only supports Linux and requires network interfaces of type *macvlan* that function as ingress gateways. These interfaces use the MAC addresses of the virtual VRRP routers. Two such interfaces are needed, one for IPv4 and one for IPv6. One issue I encountered is that for BGP EVPN operations, a bridge interface with an attached

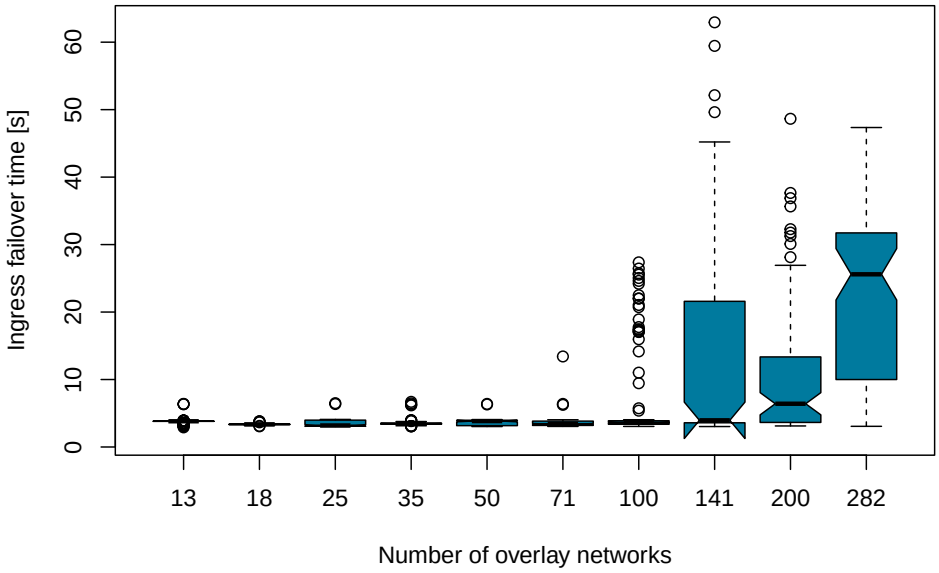


Figure 4.6: Ingress failover times for different numbers of overlay networks in the wmgwd-based implementation. 95% confidence intervals for the medians are marked by the notches on the plot. For 141 overlay networks, the lower notch is outside the hinge, meaning that the lower confidence interval bound is less than the 25th percentile.

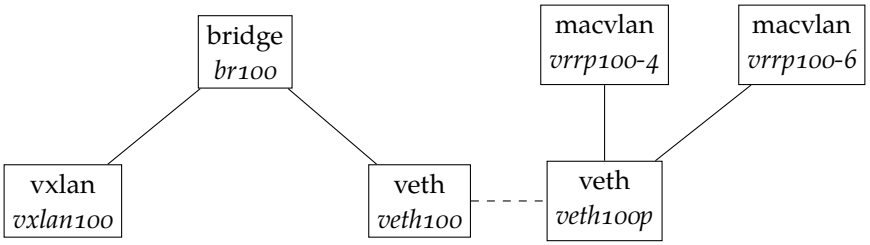


Figure 4.7: Network interface structure for VXLAN ID 100 when using BGP EVPN with VRRP in FRR. Interface types are shown above, names are italic.

Overlay networks	100	141	200	282	400	565	800	1131	1600	2263
Measurements	22	10	16	17	19	13	31	12	22	53

Table 4.3: Number of measurements for different numbers of overlay networks in the VRRP-based implementation.

VXLAN interface is required. However, the *macvlan* interface cannot use the bridge as its master interface. Therefore, I introduced a virtual Ethernet interface that forwards packets from the BGP EVPN bridge to the VRRP *macvlan* device. An example interface structure is shown in Figure 4.7.

As the IGP, I again used OSPF. Unlike the previous implementation, I decided to use semi-intelligent costs as described in Section 3.5.3. Therefore, I also eliminated the runtime configuration of FRR for the ingress gateways.

I used the same way of spacing the numbers of overlay networks on the log scale as before.

4.3.1 Results

I conducted successful measurements in the range between 100 and 2263 overlay networks. In contrast to the previous measurements, the success rate is 100 % for all numbers of overlay networks in this range. The number of measurements for each number of overlay networks is shown in Table 4.3.

For both ingress and egress failover times, I tried using a linear model for the relationship between the number of overlay networks and the failover times. As for wmgwd, the plots indicate the relationship to be superlinear.

I first look at the observed ingress failover times shown in Figure 4.8. The plots indicate that as the number of overlay networks increases, the variance in the failover times increases as well.

For all common numbers of overlay networks $n \in \{100, 141, 200, 282\}$, I want to test if there is a difference between the distributions for n overlay networks using wmgwd and VRRP. As a null hypothesis, I use $P_{n,\text{wmgwd}} = P_{n,\text{vrrp}}$. For all n, the alternatives are $P_{n,\text{wmgwd}} \prec P_{n,\text{vrrp}}$ and $P_{n,\text{vrrp}} \prec P_{n,\text{wmgwd}}$ where $\prec$ refers to one distribution being stochastically less than the other one. I use the Bonferroni correction to correct the p-values for multiple testing. I use a non-parametric Mann-Whitney U test to test for both alternatives. For alternative $P_{\text{wmgwd}} \prec P_{\text{vrrp}}$, the tests are insignificant for all n. For alternative $P_{\text{vrrp}} \prec P_{\text{wmgwd}}$, only the test for $n = 282$ is significant with $p \approx 5.87 \times 10^{-7}$. I conclude that the

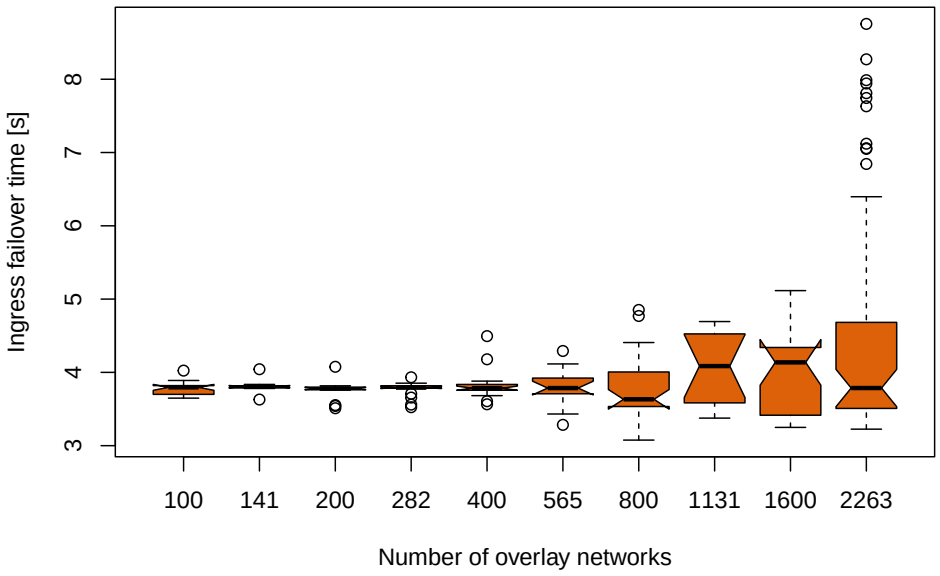


Figure 4.8: Ingress failover times for different numbers of overlay networks in the VRRP-based implementation. 95% confidence intervals for the medians are marked by the notches on the plot.

ingress failover times using VRRP are lower when $n = 282$; I cannot conclude anything for all other n.

I now look at the egress failover times. Figure 4.9 shows boxplots of the observed egress failover times. We see that the ratio of the maximum observation to the minimum observation is much larger than for the ingress failover times. Again, the plot indicates that as the number of overlay networks increases, the variance in the failover times increases as well.

I want to compare the egress failover times with VRRP to the ingress failover times with wmgwd. I use the same hypotheses and tests as for the ingress failover times at $\alpha = 0.95$. Again, I use the Bonferroni correction to correct the p-values for multiple testing. For alternative $P_{\text{wmgwd}} \prec P_{\text{vrrp}}$, the test is insignificant. For alternative $P_{\text{vrrp}} \prec P_{\text{wmgwd}}$, the tests are significant for all values of n with $p_{100} \approx 3.80 \times 10^{-14}$, $p_{141} \approx 8.96 \times 10^{-8}$, $p_{200} \approx 1.29 \times 10^{-8}$ and $p_{282} \approx 5.41 \times 10^{-11}$. I accept this hypothesis and conclude that the egress failover times using VRRP are lower than using wmgwd.

4.3.2 Analysis

The VRRP-based implementation is a considerable improvement over wmgwd since it can handle 2263 overlay networks. Still, for the same reasons as before, I would like to further increase this number. One observation is that with an increasing number of overlay networks, the ingress failover times increase less than the egress failover times. This is since there is no control plane that reconfigures the gateways' OSPF costs at runtime: For

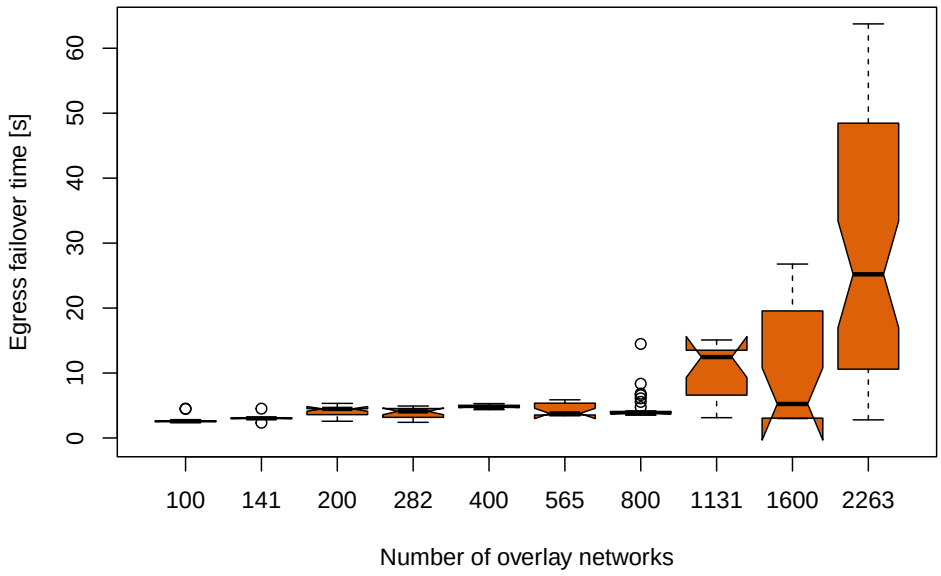


Figure 4.9: Egress failover times for different numbers of overlay networks in the VRRP-based implementation. 95% confidence intervals for the medians are marked by the notches on the plot.

ingress failover, h4 has to notice the failure and recalculate its routes. For egress failover, for each overlay network, the gateways' VRRP routers have to notice the failure, perform leader election and issue one REACH. To be able to handle additional overlay networks, I try to improve on the egress gateway control plane. I will leave the ingress control plane as-is for the next experiment.

One potential bottleneck could be the number of messages sent as part of VRRP. Each virtual router sends a VRRP announcement every 1.5 s. For 2263 overlay networks, this results in 1508 packets per second sent by each physical gateway. Instead, the technique from Section 3.6.2 could be used. Then, each gateway has to send one announcement in each time interval.

4.4 Wmgwd2

For my last implementation evaluation, I hypothesized that VRRP performance could be improved by sending fewer messages.

For that purpose, I wrote a custom daemon wmgwd2.[2] Each gateway runs one wmgwd2 process. The implementation assumes that all gateways are in the same underlay L2 network. On startup, each wmgwd2 process generates a random ID. Every second, each process broadcasts a keepalive message containing that ID. A gateway is assumed dead if

[2] See `https://github.com/rgwohlbold/wmgwd2/tree/cc79e19178afd59442fd86865105d023adae68dc`, last accessed 2023-07-25.

Overlay networks	100	141	200	282	400	565	800	1131	1600
Measurements	83	54	22	64	140	64	60	60	20
Successes	83	43	11	48	123	54	47	52	16
Success rate	1	0.8	0.5	0.75	0.88	0.84	0.78	0.87	0.8

Table 4.4: Number of measurements for different numbers of overlay networks in the wmgwd2-based implementation.

no keepalive messages are received within 3 s. Using HRW hashing with the gateway IDs and the overlay network IDs, assignments of overlay networks to gateways are determined. The daemon reconciles the state of the network interfaces with the assignment results when a keepalive message is received or a dead timer expires.

For the IGP, I used the same control plane as in the last section: I use random costs and configure them at startup.

I use the same numbers of overlay networks as in the previous implementations. This time, I only measure the egress failover time since the ingress control plane is the same as before.

4.4.1 Results

I conducted successful measurements for up to 1600 overlay networks. The number of measurements, successful measurements and the success rate are shown in Table 4.4.

Figure 4.10 shows box plots of the observed egress failover times. I tested whether this implementation achieves lower failover latencies better than the VRRP-based implementation. For $n \in \{100, 141, 200, 282, 400, 565, 800, 1131, 1600\}$, my null hypothesis is that $P_{n,\text{vrrp}} = P_{n,\text{wmgwd2}}$. I tested the hypotheses that $P_{n,\text{wmgwd2}} \prec P_{n,\text{vrrp}}$ and $P_{n,\text{vrrp}} \prec P_{n,\text{wmgwd2}}$ at $\alpha = 0.95$ using a non-parametric Mann-Whitney U test. I use the Bonferroni correction to correct the p-values for multiple testing. None of the tests for $P_{n,\text{wmgwd2}} \prec P_{n,\text{vrrp}}$ are significant. The tests for $P_{n,\text{vrrp}} \prec P_{n,\text{wmgwd2}}$ for $n \in \{100, 141, 282, 400, 565, 800\}$ are significant with $p_{100} \approx 1.81 \times 10^{-10}$, $p_{141} \approx 1.77 \times 10^{-2}$, $p_{282} \approx 6.88 \times 10^{-7}$, $p_{400} \approx 9.64 \times 10^{-14}$,$p_{565} \approx 6.92 \times 10^{-5}$ and $p_{800} \approx 5.64 \times 10^{-4}$. I conclude that the VRRP-based implementation generally achieves lower egress failover times wmgwd2-based implementation.

4.4.2 Analysis

The advantage of wmgwd2 over the VRRP implementation in FRR is that fewer messages are exchanged between the gateways. The disadvantage is that wmgwd2 is an external program, creating overhead as FRR has to be notified when network interfaces are modified. From the results, I cannot infer that the overhead introduced by wmgwd2 is worth the savings from passing fewer messages. Therefore, based on the data, the VRRP-based implementation is the best one I tested.

None of the implementations can consistently handle $\geq$ 3200 overlay networks. The wmgwd2-based implementation showed that the number of messages that are exchanged is not the bottleneck. Therefore, I look at other parts of the system.

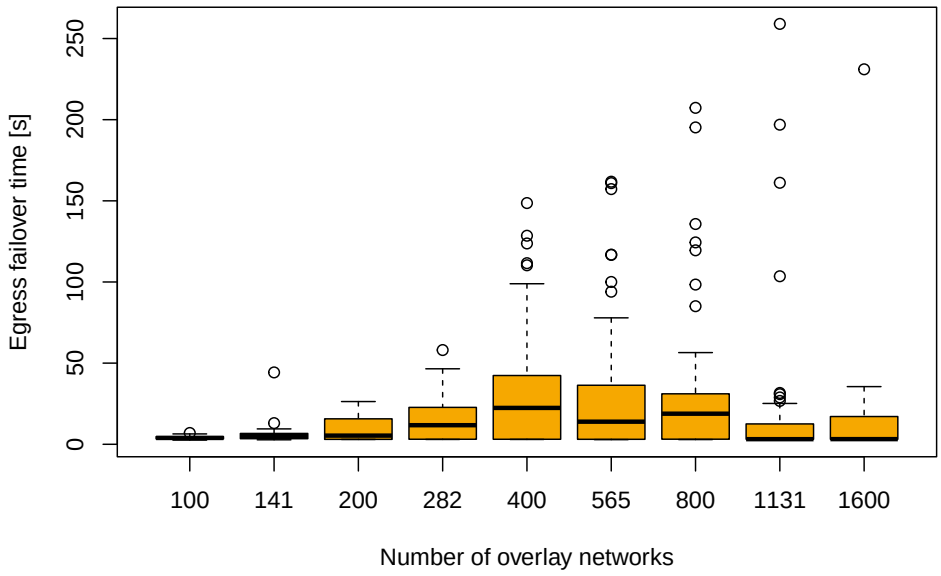


Figure 4.10: Egress failover time for different numbers of overlay networks in the wmgwd2-based implementation. 95% confidence intervals for the medians are marked by the notches on the plot.

I found that with an increasing number of overlay networks, some Linux network operations get slower. For example, with multiple thousand interfaces, wmgwd2 needs much more time to set an interface into the *protodown* state than with multiple hundred interfaces. There have been reports about standard library functions slowing down with increasing numbers of network interfaces [6]. My hypothesis is that my issue is similar, but I did not perform any further analysis. This would also explain why the relationship between the number of overlay networks and the failover time is often superlinear.

All implementations could therefore be improved by being more resourceful with the number of network interfaces. During all tests, I created network interfaces for all overlay networks on all gateways, independent of assignment. Instead, network interfaces could only be created when responsibility is assigned and deleted when unassigned. This would significantly reduce the average number of network interfaces on each gateway. Still, further tests are necessary to determine in which cases the speedup is worth the additional cost to create and delete network interfaces.

Another way in which the egress failover times could be improved is by using VMs instead of Linux namespaces for each gateway. This is due to a Linux-internal lock called the *routing table netlink (RTNL) mutex* [13]. This lock serializes operations in many major network configuration paths. Since Linux namespaces share a kernel, their network interface operations compete for the lock, slowing down each other. If VMs were used, for each VM, there would be no other process competing for the mutex.

One more possibility to improve performance could be to use a different system design that does not change network interfaces at runtime. This could be achieved by integrating my system design directly into FRR. Another option could be to introduce costs into the WiMoVE control plane as described in Section 3.6.1. This would require no runtime reconfiguration of the gateways.

Lastly, the gateways could use another data plane. All the tested implementations require multiple network interfaces for each overlay network. One option is to use an alternative operating-system representation of a tunnel, like Linux lightweight tunnels [37]. However, they, are not yet supported in FRR [14]. Another option could be to use like the Data Plane Development Kit (DPDK) [7]. This could potentially support a constant number of network interfaces, independent of the number of overlay networks. Whether this works for WiMoVE is unclear.

5 Conclusion

In this thesis, I conceptualized a system design that integrates multiple gateways into WiMoVE. This paves the way for highly available WiMoVE deployments, enabling even more use cases. Besides WiMoVE, my design can also be used for general multi-tenant L2 VPN gateways. The theoretical design I proposed is sound; the control plane I conceptualized has little overhead.

It turned out that integrating with existing software is the most challenging part about a multi-gateway architecture. Seemingly, FRR is not designed to be extensible by other programs. Therefore, the interface between my programs and FRR is inefficient and fails to scale to multiple thousand overlay networks. Future work could improve the integration of my design into FRR.

I also encountered scalability issues related to the number of network interfaces inside a single network namespace on Linux. Using multiple network interfaces per overlay network introduces too much overhead when using intelligent costs for the egress gateways. Similar issues occur in datacenter switches which increasingly run on Linux [28]. Features like Linux lightweight tunnels exist, but still unsupported in many user-space programs. Adding support for lightweight tunnels in existing open-source software like FRR presents an area for future work.

Bibliography

[1] Mohammed Alasmar, George Parisis, Richard Clegg, and Nickolay Zakhleniu. "On the Distribution of Traffic Volumes in the Internet and its Implications". In: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. IEEE INFOCOM 2019 - IEEE Conference on Computer Communications. Paris, France: IEEE, Apr. 2019, pages 955–963. ISBN: 978-1-72810-515-4. DOI: 10.1109/INFOCOM.2019.8737483. URL: https://ieeexplore.ieee.org/document/8737483/ (visited on July 3, 2023).

[2] Mina Tahmasbi Arashloo, Pavel Shirshov, Rohan Gandhi, Guohan Lu, Lihua Yuan, and Jennifer Rexford. "A Scalable VPN Gateway for Multi-Tenant Cloud Services". In: *ACM SIGCOMM Computer Communication Review* 48.1 (Apr. 2018), pages 49–55. DOI: 10.1145/3211852.3211860.

[3] Randall Atkinson and Stephen Kent. *Security Architecture for the Internet Protocol*. RFC 2401. Nov. 1998. DOI: 10.17487/RFC2401. URL: https://www.rfc-editor.org/info/rfc2401.

[4] etcd Authors. *etcd: A distributed, reliable key-value store for the most critical data of a distributed system*. 2023. URL: https://etcd.io.

[5] "Bin-Packing". In: *Combinatorial Optimization*. Volume 21. Berlin/Heidelberg: Springer-Verlag, 2006, pages 426–441. ISBN: 978-3-540-25684-7. DOI: 10.1007/3-540-29297-7_18. URL: http://link.springer.com/10.1007/3-540-29297-7_18 (visited on July 10, 2023).

[6] Bryan Quigley. *Extremely Slow Sudo with Many Network Interfaces Due to Slow Getifaddrs() Syscall Perf*. Ubuntu Bug Tracker. Jan. 24, 2014. URL: https://bugs.launchpad.net/ubuntu/+source/sudo/+bug/1272414.

[7] DPDK contributors. *Data Plane Development Kit*. Version 23.03. Mar. 31, 2023. URL: https://www.dpdk.org/.

[8] OpenBSD contributors. *OpenBSD PF - Firewall Redundancy (CARP and pfsync)*. OpenBSD Frequently Asked Questions. URL: https://www.openbsd.org/faq/pf/index.html.

[9] David Thaler and Chinya Ravishankar. *A Name-Based Mapping Scheme for Rendezvous*. Technical Report CSE-TR-316-96. University of Michigan, 1996.

[10] Ralph Droms. *Dynamic Host Configuration Protocol*. RFC 2131. Mar. 1997. DOI: 10.17487/RFC2131. URL: https://www.rfc-editor.org/info/rfc2131.

[11] Agnieszka Dutkowska-Żuk, Austin Hounsel, Amy Morrill, Andre Xiong, Marshini Chetty, and Nick Feamster. "How and Why People Use Virtual Private Networks". In: *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pages 3451–3465. ISBN: 978-1-939133-31-1. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/dutkowska-zuk.

[12] Even Zhang. *Is DHCP FORCERENEW (RFC 3203) message supported in Windows DHCP client?* Microsoft Learn Questions. July 10, 2020. URL: https://learn.microsoft.com/en-us/answers/questions/118689/is-dhcp-forcerenew-(rfc-3203-)-message-supported-i.

[13] Florian Westphal. "RTNL mutex, the network stack big kernel lock". In: Netdev 2.2. Seoul, Korea: Red Hat, Nov. 2017. URL: https://netdevconf.info/2.2/papers/westphal-rtnlmutex-talk.pdf.

[14] FRR contributors. *FRRouting Project*. 2023. URL: https://frrouting.org.

[15] Tony L. Hain. *Architectural Implications of NAT*. RFC 2993. Nov. 2000. DOI: 10.17487/RFC2993. URL: https://www.rfc-editor.org/info/rfc2993.

[16] Matt Holdrege and Pyda Srisuresh. *IP Network Address Translator (NAT) Terminology and Considerations*. RFC 2663. Aug. 1999. DOI: 10.17487/RFC2663. URL: https://www.rfc-editor.org/info/rfc2663.

[17] IEEE Computer Society. *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. IEEE Standard 802.11-2020. Dec. 2020, page 4379.

[18] Jason A. Donenfeld. "WireGuard: Next Generation Kernel Network Tunnel". In: NDSS. 2017. URL: https://www.wireguard.com.

[19] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web". In: *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing - STOC '97*. the twenty-ninth annual ACM symposium. El Paso, Texas, United States: ACM Press, 1997, pages 654–663. ISBN: 978-0-89791-888-6. DOI: 10.1145/258533.258660. URL: http://portal.acm.org/citation.cfm?doid=258533.258660 (visited on June 13, 2023).

[20] Kevin Parrish. *Wi-Fi 6 vs. Wi-Fi 6E: What's the Difference?* May 24, 2023. URL: https://www.highspeedinternet.com/resources/wi-fi-6-vs-wifi-6e.

[21] Marek Majkowski. *Cloudflare servers don't own IPs anymore – so how do they connect to the Internet?* The Cloudflare Blog. Nov. 25, 2022. URL: https://blog.cloudflare.com/cloudflare-servers-dont-own-ips-anymore/.

[22] Mininet Project Contributors. *Mininet*. Version 2.3.0. URL: https://mininet.org/.

[23] M. Mitzenmacher. "The power of two choices in randomized load balancing". In: *IEEE Transactions on Parallel and Distributed Systems* 12.10 (Oct. 2001), pages 1094–1104. ISSN: 10459219. DOI: 10.1109/71.963420. URL: http://ieeexplore.ieee.org/document/963420/ (visited on June 28, 2023).

[24] Moritz Frenzel and Nicola von Thadden. "Infrastructure Review" (Chaos Communication Camp). Aug. 25, 2019. URL: https://media.ccc.de/v/Camp2019-10390-infrastructure_review.

[25] John Moy. *OSPF Database Overflow*. RFC 1765. Mar. 1995. DOI: 10.17487/RFC1765. URL: https://www.rfc-editor.org/info/rfc1765.

[26] John Moy. *OSPF Version 2*. RFC 2328. Apr. 1998. DOI: 10.17487/RFC2328. URL: https://www.rfc-editor.org/info/rfc2328.

[27] Stephen Nadas. *Virtual Router Redundancy Protocol (VRRP) Version 3 for IPv4 and IPv6*. RFC 5798. Mar. 2010. DOI: 10.17487/RFC5798. URL: https://www.rfc-editor.org/info/rfc5798.

[28] Nikolay Aleksandrov, David Ahern, and Roopa Prabhu. "Scaling Network Interfaces on Linux". In: *Scaling Network Interfaces on Linux*. NetDev 1.1. Seville, Spain, Feb. 10, 2016. URL: https://netdevconf.info/1.1/proceedings/slides/ahern-aleksandrov-prabhu-scaling-network-cumulus.pdf.

[29] OpenVPN, Inc. *OpenVPN*. URL: https://openvpn.net/.

[30] OpenVPN, Inc. *Setting up high-availability failover mode*. OpenVPN Access Server Knowledge Base. URL: https://openvpn.net/vpn-server-resources/setting-up-high-availability-failover-mode/.

[31] Celia Paulsen and Robert Byers. *Glossary of key information security terms*. NIST IR 7298r3. Gaithersburg, MD: National Institute of Standards and Technology, July 2019, NIST IR 7298r3. DOI: 10.6028/NIST.IR.7298r3. URL: https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.7298r3.pdf (visited on July 9, 2023).

[32] Ashok Singh Sairam and Gautam Barua. "Distributed route control schemes to load balance incoming traffic in multihomed stub networks". In: *2010 National Conference On Communications (NCC)*. IEEE. 2010, pages 1–5.

[33] Aaron Schlitt, Alexander Sohn, Lina Wilske, and Richard Wohlbold. "WiMoVE: An Architecture for Large Wi-Fi Systems". In: *SKILL 2023*. Gesellschaft für Informatik. 2023.

[34] SoftEther VPN Project. *SoftEther VPN*. URL: https://www.softether.org/.

[35] Jeongseok Son, Yongqiang Xiong, Kun Tan, Paul Wang, Ze Gan, and Sue Moon. "Protego: Cloud-Scale Multitenant IPsec Gateway". In: *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. Santa Clara, CA: USENIX Association, July 2017, pages 473–485. ISBN: 978-1-931971-38-6. URL: https://www.usenix.org/conference/atc17/technical-sessions/presentation/son.

[36] Yves T'Joens, Peter De Schrijver, and Christian Hublet. *DHCP reconfigure extension*. RFC 3203. Dec. 2001. DOI: 10.17487/RFC3203. URL: https://www.rfc-editor.org/info/rfc3203.

[37] Thomas Graf. *Lightweight & Flow Based Tunneling*. E-mail. July 10, 2015. URL: https://lwn.net/Articles/650778/.

[38] Andrew J. Valencia, Glen Zorn, William Palter, Gurdeep-Singh Pall, Mark Townsley, and Allan Rubens. *Layer Two Tunneling Protocol "L2TP"*. RFC 2661. Aug. 1999. DOI: 10.17487/RFC2661. URL: https://www.rfc-editor.org/info/rfc2661.

A Zusammenfassung

In großen Wi-Fi-Netzen kann Broadcastverkehr viel Übertragungszeit beanspruchen. Gängige Wi-Fi-Systeme filtern Broadcastverkehr. Dabei werden Anwendungen beeinträchtigt, die auf Layer-2-Funktionalität angewiesen sind. WiMoVE ist eine Wi-Fi-Architektur, die einen anderen Ansatz verfolgt. Durch die Verwendung von Overlaynetzen auf existierender IP-Infrastruktur reduziert WiMoVE den Broadcastverkehr und stellt Nutzern trotzdem reguläre Layer-2-Funktionalität bereit. In WiMoVE gibt es ein Gateway, das Pakete zwischen den Overlaynetzen und dem restlichem Netz weiterleitet. Bislang wurde nur ein Gateway in WiMoVE betrachtet. Dies führt zu Verlässlichkeits- und Durchsatzproblemen. In dieser Bachelorarbeit integriere ich mehrere Gateways in WiMoVE. Zunächst stelle ich eine Netzwerkarchitektur auf, in der Gateways als Layer-3-Router fungieren. Die Gateways geben für jedes Overlaynetz eine Route über ein interior gateway protocol (IGP) bekannt. Danach führe ich das Konzept der Verantwortlichkeit von Gateways für Overlaynetze ein. Ich komme zu dem Schluss, dass es für jedes Overlaynetz genau ein verantwortliches Gateway geben sollte. Anschließend diskutiere ich mögliche Strategien zur Lastbalancierung und entscheide, einen randomisierten Ansatz zu verwenden. Ich zeige, wie dieser mit dem IGP und der WiMoVE-Steuerebene implementiert werden kann. Anschließend präsentiere ich drei Implementierungen, die verschiedene Featurekombinationen verwenden. Ich vergleiche die Implementierungen basierend auf gemessenen Failoverzeiten für unterschiedliche Anzahlen von Overlaynetzen. Die Ergebnisse zeigen, dass die Implementierung basierend auf Virtual Router Redundancy Protocol (VRRP) am vielversprechendsten ist. Auch wenn die Implementierung über 2000 Overlaynetze verarbeiten kann, benötigen große Installationen zehntausende Overlaynetze. Ich analysiere die Ursache der Skalierungsprobleme und diskutiere mögliche Lösungen.

Eidesstattliche Erklärung

Hiermit versichere ich, dass meine Bachelorarbeit "Load Balancing and Failover for Isolated, Multi-Tenant Layer 2 Virtual Private Networks" selbstständig verfasst wurde und dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt wurden. Diese Aussage trifft auch für alle Implementierungen und Dokumentationen im Rahmen dieses Projekts zu.

Potsdam, den 27. Juli 2023,

(Richard Wohlbold)