A Cost-Effective And Simplified Method for Network Failover

White Paper 2

Version 1

Special Start-Up Service

by Gospodin Bodurov

Executive summary

The growing demand for self-hosted infrastructure underscores the need for reliable and resilient network connectivity. Ensuring continuous access and minimizing downtime requires a dual uplink strategy that provides redundancy and load balancing. In this paper, we introduce a cost-effective approach to implementing self-hosted environments with two different uplinks, maximizing uptime and optimizing network performance.

RATE THIS PAPER ★★★★★

Introduction

Recent global events have led to rising power consumption and increased labor costs, forcing hosting and cloud providers to raise prices for their customers. Over the past two years, prices for the same set of services have roughly doubled. These challenges are expected to reshape the IT industry, pushing more companies to consider self-hosting for non-critical systems. A key component of any server infrastructure is high-availability for uplinks. Although power and uplink outages are rare today, they do still occur. To mitigate these risks, any reputable hosting or cloud provider will implement a backup uplink solution. This paper outlines the approach we have adopted in our self-hosting environment to address these challenges.

When discussing efficient multiple uplinks solutions, three major considerations come to mind - high availability, load balancing, and expected speed of the used configurations. Different data center providers may have varying industry guidelines for the acceptable configurations of these factors. To avoid confusion, we will define and use the following constraints in the subsequent sections:

- High availability should be configured using an active-backup approach, where data is transmitted and received through the primary available interface. The secondary interface remains on standby and is only activated if the primary interface experiences a failure.
- To simplify the setup and minimize equipment costs, load-balancing will not be implemented.
- Speed will be determined by the primary uplink, as the secondary is typically only used in rare circumstances and has little impact. However, as a general guideline, both service providers should offer matching speeds.

Comparison of Network High-Availability Approaches

When addressing uplink high availability, the industry typically relies on two main strategies: BGP multihoming and network bonding.

BGP multihoming refers to a network setup where an organization connects to multiple ISPs using BGP to ensure redundancy and improved reliability. This configuration helps maintain internet connectivity even if one ISP experiences downtime, by routing traffic through alternate connections.

Network bonding is a technique that combines multiple network interfaces into a single logical interface to increase bandwidth and provide redundancy. It improves network performance and reliability by distributing traffic across multiple links and maintaining connectivity if one of the links fails.

Table 1 highlights the key differences between these two approaches.

Table 1

Breakdown of the differences between BGP multihoming and network bonding in terms of uplink high availability

Aspect	BGP Multihoming	Network Bonding
Purpose	Provides redundancy by connecting to multiple ISPs	Combines multiple physical links into a single logical interface
Layer of Operation	Operates at Layer 3 (Network layer)	Operates at Layer 2 (Data Link layer)
Failover Mechanism	Automatically reroutes traffic through a different ISP if one fails	Distributes traffic across available links and shifts traffic in case of failure
Scalability	Scales well across geographic regions, multiple ISPs	Typically limited to connections within the same physical network
Complexity	Requires configuration of BGP routing and coordination with ISPs	Simpler to implement, mainly handled on the local network
Cost	May involve higher costs due to a need of more advanced equipment	Generally lower cost as it involves only local network infrastructure
Load Balancing	Supports intelligent routing and load balancing based on policies and traffic types	Performs basic load balancing, splitting traffic evenly across available links
Configuration Effort	Requires BGP expertise and ISP coordination	Easier to configure, typically done at the network device level

The table above clearly shows that use of both approaches is necessary for achieving a pure high availability in a network setup. However, it's also evident that they introduce significant complexity and dependencies into the network, violating a key engineering principle: keeping the solution as simple as possible.

Choosing The Right Approach

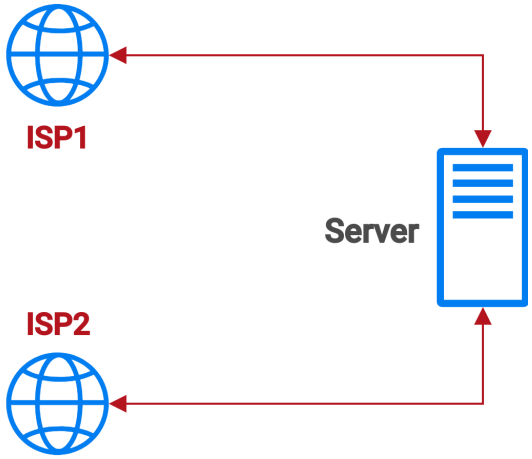
A significant number of SMEs lack the resources or expertise to configure advanced network setups like BGP multihoming and network bonding. This is especially true for most tech start-ups, leaving them with two main options: rely on cloud providers for networking infrastructure or forego network failover altogether. When setting up our self-hosted infrastructure, our goal was to achieve a reasonable level of reliability using only off-the-shelf equipment. Additionally, we aimed to keep the system as simple as possible for easy maintenance and updates.

Foundation

Every scientific paper should begin with a basic example and progressively introduce complexity until the main objective is achieved. In our case, we start with the most straightforward networking setup: two uplinks connected to a single computer. **Figure 1** illustrates this configuration. Furthermore, this article will focus on implementing our approach specifically on Linux server environments. If you wish to replicate a similar setup on Windows or macOS, consider it a practical exercise.

Figure 1

Two ISPs connected to a different LAN each

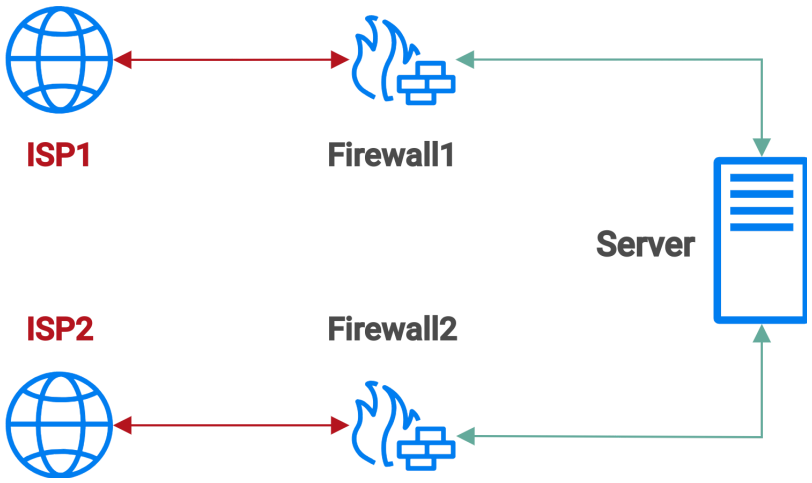


The approach described transforms our single server into a router, requiring all necessary configurations, such as BGP multihoming and network bonding, to be handled directly on the server. Additionally, the server relies on its own firewall software, which is often not optimized for uplink protection.

A more effective solution is to use dedicated firewall routers for each uplink and connect the server to them. **Figure 2** illustrates this improved setup, with two routers—one for each uplink.

Figure 2

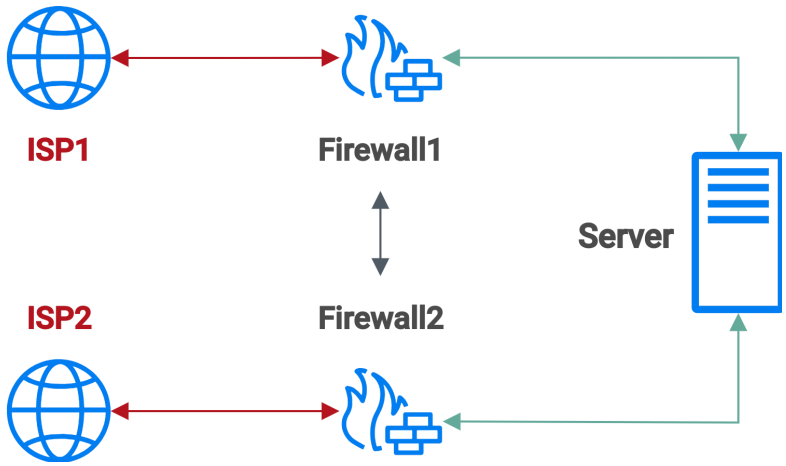
Now each ISP has a firewall protecting its internal network



In this scenario, we should choose one of the two approaches mentioned earlier. For BGP multihoming, the typical solution is to connect the two routers using CARP, creating a single logical router for the setup. For network bonding, the interfaces should be bonded at the server level. While some specialized router vendors may support bonding at the logical router level, this would add unnecessary complexity to the configuration. Both approaches are illustrated in **Figures 3 and 4**.

Figure 3

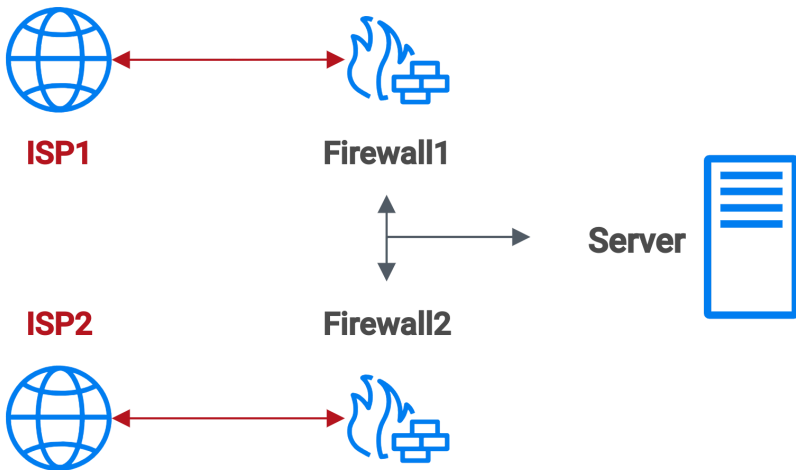
Both routers act as one logical router using multicast (CARP) to sync state between them. ISP1 and ISP2 are bond together using BGP multihoming



This method is widely implemented in data centers for redundancy and load balancing. However, the main challenge with this approach is the need for routers that support CARP and more advanced switches to accommodate the internal network.

Figure 4

Networking switching is done on the server level



The primary disadvantage of this setup is that it requires enabling a kernel module, configuring each server in the cluster individually, and modifying the router settings to accommodate this scenario, including ensuring that both DHCP servers use the same configuration settings, among other tasks.

Simple Approach

Considering the previous paragraphs, both standard BGP multihoming and network bonding provide significantly more functionality than just basic uplink failover, which would complicate our setup beyond our intended scope.

Let's revisit our basic example featuring two routers and a single computer connected to them. By conducting tests in Linux, continuously pinging a specified domain while repeatedly disconnecting and reconnecting the cables, we can observe the following behavior:

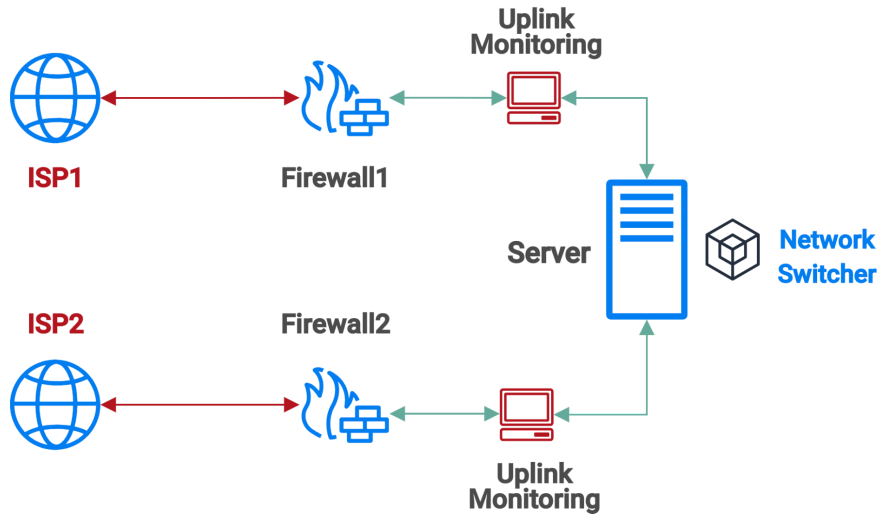
- Once all cables are connected and there are no faulty uplink, then Linux will automatically determine which network route should take priority and utilize it accordingly
- If the uplinks (indicated by red cables in **Figure 2**) are connected and one of the green cables fails (meaning one of the routers is down), Linux will automatically switch to the other connection
- If one of the uplinks is disconnected, but both green cables shown in **Figure 2** (indicating the router's internal networks) are functioning, Linux will not be able to switch to the correct network and will lose Internet connectivity.

Now that we have this knowledge, we need to implement a method for our Linux system to switch routes to the correct network. Fortunately, this can be easily accomplished using standard Linux command-line tools, particularly the **nmcli** command. We will leave the specifics of this implementation as a practical exercise for the reader.

Additionally, we chose a more advanced setup involving two agents (single board computers) for each network. These agents continuously monitor their respective uplink for outages, and in the event of an outage, a software package installed on the server automatically switches the network. You can view the new setup in **Figure 5**.

Figure 5

Following setup gives a good failover performance



Now the following approach has some downsides:

- There are two public IP addresses instead of just one. An advanced DNS server should be utilized to manage incoming traffic and ensure stability in the event of a failure
- The state of the connections is not maintained, which means that during network switching, a reconnection may be necessary, potentially leading to the failure of long-running operations
- For advanced deployments like Kubernetes and Docker Swarm, network bonding or advanced routing and switching will still be essential to ensure that the internal network for orchestration operates in failover mode

Building On Top Of The Simpler Approach

This straightforward approach would have been unsuitable for a standard environment 10-15 years ago, where each server hosted a copy of a software application or multiple VMs were deployed. Implementing such a solution back then would have resulted in a highly complex internal and external DNS configuration, significantly raising the overall cost and maintenance of the setup.

Two things changed during the past years, helping this approach to work:

- The emergence of orchestration engines like Kubernetes and Docker Swarm has significantly addressed the internal DNS issue with our approach, as their built-in network management automatically handles load balancing and pods failover
- The introduction of encrypted reverse TCP tunnels, such as Cloudflare's Zero Trust Tunnels, has largely resolved external DNS and load balancing challenges, effectively addressing most issues related to multiple uplinks.

Improved Setup

And indeed the following setup will work almost without issue:

- Multiple servers, each equipped with an orchestration engine, are connected to both routers as shown in **Figure 2**
- Two of the servers will host pods with reverse TCP tunnels, connecting to an infrastructure similar to Cloudflare DNS
- A software module to restart the tunnel pods from one of the manager nodes during network switching. Since orchestration engines rely on the underlying Linux network stack, this streamlined approach will switch the tunnel uplink upon restarting the tunnel

This setup presents two problems that need to be addressed: first, a network switching event will terminate all active connections, and second, if the internal network of the orchestration engine fails, everything collapses. Fortunately, uplink, router or network switch failures are extremely rare events these days. With these considerations in mind, we can implement the following policy, which effectively addresses these challenges:

- Long-running operations should be divided into smaller batches
- Regular backups of internal router should be maintained (for example, PFSense provides this capability)

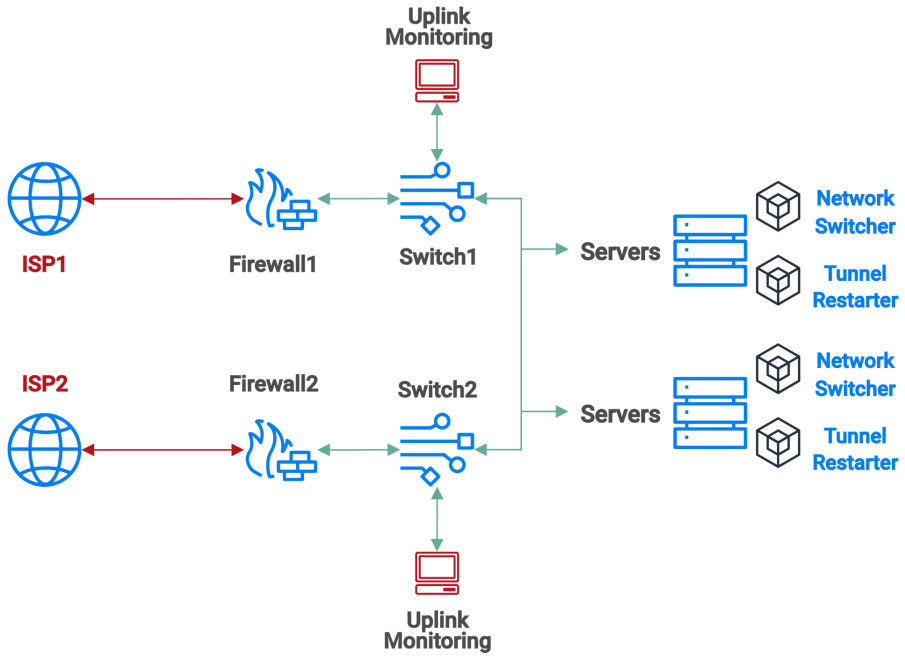
With this straightforward policy, we can effectively address both issues. Long-running operations can restart from the point of failure, and in the event of an internal router or switch failure, we can quickly replace the faulty device with a spare and resume operations.

Final Setup

Considering all this research, we have chosen to use the following devices: two routers for uplink firewalls, one internal router for the orchestration network, three standard switches for server networking, two single-board computers for uplink monitoring, and a few custom software packages to automate network monitoring and switching mechanisms. The resulting configuration is illustrated in **Figure 6**.

Figure 6

Final setup of how the new approach should work



Our approach offers an alternative method for ensuring network failover. While it has some downsides, it is suitable for non-critical systems, such as demos or internal applications. The level of high availability is not comparable to that provided by production-grade data centers; however, the operational costs are significantly lower. For instance, a sample setup with five servers using the cooling solution outlined in Whitepaper 1 incurs recurring costs of approximately **50-60 euros** per month, with networking equipment costs not exceeding **500 euros**. In contrast, achieving the same setup in the cloud or through a dedicated server rack in a data center would be nearly **20 times** more expensive.

Conclusion

Creating software products is becoming increasingly easier each year, but this convenience comes at a cost. More systems will require cloud environments, and using production-grade clouds and data centers for demos, development, and internal systems can lead to unnecessary waste of natural resources. To stay competitive in today's technological landscape, we need more flexible approaches, such as utilizing cost-effective networking stacks or repurposing old hardware. Our solution provides small and medium-sized businesses (SMBs) with the opportunity to implement adaptable infrastructure in server rooms with less stringent requirements for internal or demo systems. Moreover, our approach can be extended to remote and home offices, offering backup uplinks for those in need.

And of course, we have several improvements in mind for future iterations of the approach:

- Adjust the TCP tunnels to automatically detect problematic connections and reconnect, rather than restarting the entire pod
- Set up a round-robin configuration with four servers, each equipped with tunnel pods. Two servers will utilize the primary uplink, while the other two will connect through the secondary uplink. An external DNS will then distribute traffic evenly among the tunnels deployed across these four servers.

✎ About the author

Gospodin Bodurov is a seasoned software engineer, entrepreneur, investor, and mentor with a rich career spanning multiple domains in technology and business. Growing up in Burgas, Bulgaria, Bodurov's deep engagement with computers began at a young age, leading him to self-teach C++ and become a dedicated Linux and Unix user. Holding degrees in Computer Science and IT Project Management, he has held various roles from intern to executive director in startups across Bulgaria, the UK, the US, and Switzerland. As the CEO of S³, a family-owned business incubator and investment fund, Bodurov focuses on cultivating an entrepreneurial ecosystem grounded in integrity and flexibility. His extensive experience includes lecturing at prestigious institutions, coaching national informatics teams, and developing a broad spectrum of systems and applications.

RATE THIS PAPER ★★★★★

Resources

- RFC 4271 - A Border Gateway Protocol 4 (BGP-4) - https://datatracker.ietf.org/doc/html/rfc4271
- Linux Ethernet Bonding Driver - https://www.kernel.org/doc/Documentation/networking/bonding.txt
- Cloudflare Tunnels
  TCP-based reverse tunnels
- Docker Swarm
  Docker containers orchestration engine

Note: Internet links can become obsolete over time. The referenced links were available at the time this paper was written but may no longer be available now.

Contact us

For feedback and comments about the content of this white paper:

Special Start-Up Service
info@s-3.team

If you have start-up or have questions specific to your setup:

Contact your S³ Team representative at
https://www.s-3.team/#contact-index

If you are interested in more content coming from Gospodin Bodurov:

Visit
https://bodurov.net