Original scientific paper

Towalink – an Open and Flexible Site Networking Solution

Dirk Henrici, *Senior Member, IEEE,* and Lars Wischhof, *Senior Member, IEEE*

***Abstract* — Securely and reliably connecting multiple operating sites using existing Internet access lines is a common requirement for companies as well as individuals. Towalink is a new open and flexible solution for interconnecting multiple network sites reliably and securely. It is built around proven open-source projects and by itself published open-source. We present in theory and based on a practical setup the central management it provides and show how it is following the infrastructure-as-code paradigm.**

***Keywords* — VPN, SD-WAN, Open Source, WireGuard.**

I. Introduction

Many companies have multiple operating sites and require exchanging data between these sites reliably and securely. A cost-effective approach is to employ existing Internet access lines. Secure tunnels can then be created on top of these to interconnect the different sites while providing confidentiality. With an increasing number of sites, configuring these tunnels becomes a tedious task - especially in case more complex topologies like full mesh are desired that require the setup of many tunnels including managing the needed cryptographic keys. Often, dynamic routing is mandatory and needs to be configured to use alternate paths in case of link failures. On top of these requirements, additional ones add up in practice, e.g., there is a desire to keep the routers' software up to date, to provide monitoring and to aid in troubleshooting.

Towalink is an open-source project that aims at providing an open and flexible solution to these requirements. It is built around and orchestrates several widely used and proven open-source projects to tackle the task. Running on top of the Linux operating system, it uses Ansible [1] for configuration management, secure WireGuard [2] tunnels, widespread tooling like rsync, and Bird [3] or FRR [4] as routing daemon. git is used as version control system to be able to follow an infrastructure-as-code paradigm.

An overview of the project has already been presented in [5]. Since then, Towalink became more mature and now applies the infrastructure-as-code paradigm. It has been improved and extended based on user feedback and the practical experience of operating the solution multiple more months in a practical set-up. Therefore, in this article we describe the solution in more detail - focusing on its structure, its building blocks, and the flexibility provided.

The article is structured as follows: The next section provides an overview of related work and other solutions already available in the networking community. Afterwards, we describe requirements and challenges and present how these are addressed with the architecture and design of Towalink. Finally, we illustrate the benefits of the proposed solution using a practical evaluation setup, including examples for monitoring the system. The article concludes with a brief summary and outlook.

Paper received June 15, 2023; accepted July 12, 2024. Date of publication August 02, 2024. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Milo Tomašević.

This paper is a revised and expanded version of the paper presented at the 30th Telecommunications Forum TELFOR 2022 [5].

Dirk Henrici (corresponding author) and Lars Wischhof are with the Department of Computer Science and Mathematics, Munich University of Applied Sciences HM, Germany (phone: +49 89 1265-3774 and -3766, e-mail: dirk.henrici@hm.edu and lars.wischhof@hm.edu).

II. Related Work

The general idea of SD-WAN is to apply the Software Defined Networking (SDN) concept in Wide Area Networking (WAN) scenarios. Main benefits are high flexibility and low costs since existing Internet connectivity is used. However, several challenges emerge: Underlying connections must be encrypted (usually using VPN technologies), tunnels and topologies must be configured. In the following, we briefly summarize related work in these areas and give an overview of related projects.

Encryption of the data connections between different sites on the network layer is often based on the IPsec protocol [6]. It is natively supported by most modern operating systems, for example current Linux kernels. Since a manual setup of the required keys becomes impractical if more than a few IPsec sessions need to be configured, the Internet Key Exchange (IKE) protocol has been developed. "strongSwan" [7] is one of the most popular open-source implementations. An alternative VPN approach is to perform the encryption on higher layers: "OpenVPN" [8] is a widely used tool to implement VPN servers, using Transport Layer Security (TLS) connections as main building block. "WireGuard" [2] is a comparatively new VPN implementation working on top of UDP. It is supported natively in the Linux kernel since 2020, and it is easier to configure and more lightweight than IPsec.

There are multiple open-source projects that provide a user interface around the WireGuard configuration for building a central VPN server. To mention just two, "wgfrontend" [9] provides an easy-to-use web user interface to configure road warrior setups; "Algo VPN" [10] simplifies management of WireGuard and also IPsec on the command line. VPN functionality is also part of firewall distributions such as "OPNsense" [11].

However, with all these VPN solutions, configuration of site routers and deployment of key material is up to the

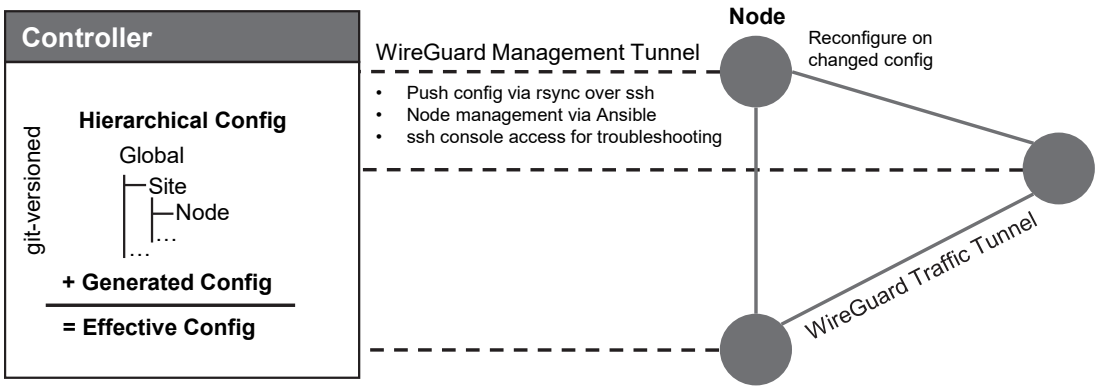

Fig. 1. Applying the hierarchical configuration to the Nodes using the WireGuard management tunnel.

administrator; there is no central key management or even zero-touch installation of routers.

Therefore, several projects extend the VPN concept to a SD-WAN: Many commercial vendors provide proprietary SD-WAN solutions which have a variety of features, central site router management being a very basic one. On open-source side, ”flexiWAN” [12] has had quite some telco press coverage, with the web user interface being commercial software, however. ”Nante-WAN” [13] is already around for some years but appears abandoned since 2018. SD-WAN architectures and tooling are also considered in other recent research projects, e.g. Troia et al. [14] performed measurements in a testlab set-up based on open-source components such as the OpenDaylight SDN controller for managing decentral access nodes (Customer Premises Equipment, CPE). While the basic motivation for these research projects seems similar, Towalink differs in the applied software stack, provides a unified configuration, and is already applied in production settings.

Infrastructure as Code (IaC) [15] [16] is a concept which is motivated by the need to simplify the process of configuring, managing, and provisioning IT infrastructures. It is based on using machine-readable configuration files (e.g. in YAML, JSON or XML), which are often managed using a version control system such as git. As illustrated e.g., in a recent case-study [17], this concept can simplify the setup of Open Networking scenarios. Analogously, IaC has also significant advantages for managing SD-WANs: a central configuration repository can be the single source of truth for all elements in the network and provides transparency on configuration changed performed. These advantages are also the reason why Towalink adopted the IaC approach.

III. Design and Implementation

For the architecture and design of Towalink, which is described in more detail in the following, the focus was on five key aspects:

1. Support for arbitrary network topologies for the links between sites
2. Central point of management for the configuration
3. No central point of failure, i.e. decentralized configuration/local replication of the configuration
4. Zero-touch installation of new routers
5. For configuration errors: fast and convenient roll-back

Regarding the first aspect, different locations shall be able to be interconnected using arbitrary topologies. They could be fully meshed, following a hub and spoke topology, a hierarchical topology, any other basic topology, or any mixture of basic topologies. To model this, we assign each location to one or more groups. Within each group, locations are connected in a full-mesh topology. With this simple approach, we can easily describe and manage arbitrary topologies. See Section IV on the practical setup for an example of this.

At each location (called “Site” in Towalink), one can have one or more routers (called “Nodes” in Towalink). These routers need to be managed and configured. We want to do this from a central point of management that we call the Towalink Controller. Each router (i.e. Node) establishes a secure WireGuard tunnel to the Controller. This tunnel always remains active and is used for managing the router. Router management can be done in three ways as illustrated in Figure 1: First, the configuration management tool Ansible can be used to install and update software (incl. patching the operating system) and to perform any other desired tasks on the router. Second, a more specialized approach for network configuration is used: Configuration files can be mirrored to the router using the rsync tool. Changed files automatically trigger reloads of the associated service(s), e.g. a routing daemon. For this, we use a feature of rsync to provide the path to the remote rsync executable to place a hook that is executed after successful file mirroring. Third, ssh can be used over the management tunnel to gain console access to the router for advanced troubleshooting.

New routers can be added using a zero-touch installation process. The routers ship with a generic bootstrap script that attempts to contact the Controller. After confirmation by the administrator on the Controller, the router gets the needed configuration to setup the management tunnel. Using Ansible, the needed services are installed on the router. Afterwards, the router can be managed and used regularly. Router replacement follows the same procedure and is therewith easy, too.

With the described approach, we get central router management incl. the ability to keep routers up-to-date, and we also get zero-touch installation. The Controller is only needed to manage the routers and push new configuration to the routers. There is no need for it to be highly available. Due to the decentral configuration, connectivity is still fully

working when the Controller is not reachable for whatever reasons. Such a setup is way easier to operate and maintain than with attempting to centralize the control plane to a highly available cluster acting as a "central brain".

This leaves us with two quite independent questions: How do we create and maintain the configuration for many routers conveniently at the Controller in a state-of-the-art manner? And what network design and configuration could be used to get a suitable site connectivity solution?

Configuration Handling

Towalink manages router configs based on two pieces of configuration: key-value pairs of configuration data and configuration file templates. The latter use Jinja2 [18] templating that allows to create the actual configuration files flexibly based on evaluating the key-value pairs of data. Jinja2 is widely adopted and used in many Python-based tools.

Since different routers have very similar configuration in practice, Towalink provides hierarchical configuration (Figure 2). This means that key-value pairs and configuration files can be defined on global level, on Site level, and on Node level. Higher-level configuration gets inherited to lower levels unless being overwritten there. Together with the Jinja2 templating engine [18], this allows for flexible configuration without error-prone duplication of repeating configuration elements.

Based on this user-provided configuration, additional key-value pairs are generated automatically. Especially, any key material needed to configure the secure tunnels needed for the defined topology is automatically generated and stored as so-called "generated configuration". The hierarchical user-defined configuration and this generated configuration together are used to evaluate the Jinja template files and therewith form the configuration files to be pushed to the routers.

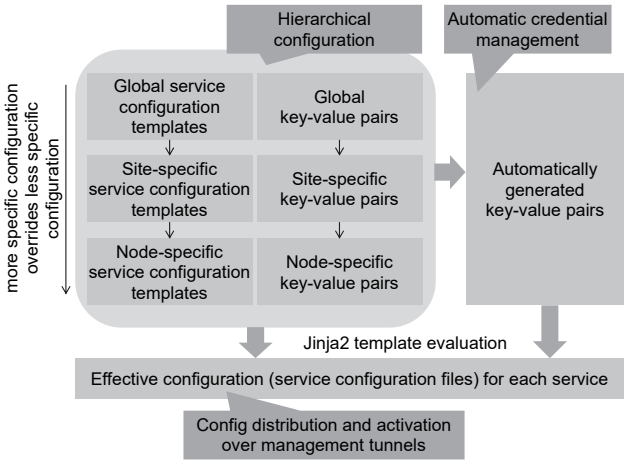

Fig. 2. Configuration handling.

Using version control is a best practice in software engineering. Due to its benefits like transparency on what was done and what was changed, one nowadays uses version control also to describe the intended target state for infrastructure like VMs, containers, and the networking between systems (Infrastructure-as-Code, IaC, see Sec. II). Towalink adapts this paradigm and versions the router (i.e. Node) configuration with git version control. This, e.g., allows for a fast and convenient rollback in case of configuration errors.

Network Design for Site Connectivity

With the configuration management described above, any services on the routers can be installed, configured, and restarted. With this capability, Towalink can be considered a generic SD-WAN framework that can be extended easily to support new services. To have a site connectivity solution that can easily be deployed in practical setups but that is still flexible and secure, we have a focus on using state-of-the-art open-source technologies for the network design of Towalink. It is described in the following.

Since we use the Linux network stack for routing, we fully support both IPv4 and IPv6 and configure both in the shipped example configuration. Point-to-point WireGuard [2] VPN links are used to implement secure traffic tunnels between routers. It is a well-documented, state-of-the-art solution to provide confidentiality. It has good performance, is less complex to configure than IPsec, and can even be used in combination with low-cost Internet routers that have trouble forwarding IPsec traffic with reasonable performance. However, using IPsec with Strongswan [7] as an IKE daemon would have been a viable alternative, as already outlined in Sec. II.

Depending on the topology configured, there can be multiple paths interconnecting the locations. In case of an outage, alternate paths shall be used automatically without user interaction. Dynamic routing is used to achieve this. We use Bird [3] as a routing daemon. Each Site gets configured as a private autonomous system and routers (i.e. Nodes) in different Sites exchange routes via the external Border Gateway Protocol (eBGP). An additional routing protocol as interior gateway protocol (IGP) is not needed with the default configuration provided. This provides for simplicity in operation and for faster failover.

To achieve low failover times in case of link failure, bi-directional forwarding detection (BFD) is run on top of the point-to-point WireGuard links to detect any outages quickly and to then foster rerouting. Alternatively, FRR [4] could have been used as routing daemon.

Note that only standardized protocols and widely used and proven open-source software like Linux, WireGuard, and Bird are used in this network design. This allows for easy troubleshooting since any experienced network engineer can do it and suitable tools such as protocol dissectors are widely available. Also note that we describe a typical example configuration here – since Towalink is an open-source project, individual parts such as the technology used for the secure traffic tunnels can be configured and even exchanged as desired.

IV. Evaluation in a Practical Setup

In order to demonstrate the feasibility and advantages of using the concept proposed with Towalink for practical SD-WAN scenarios, it was applied in a real-world scenario which includes typical link configurations for SD-WANs for small or medium enterprises. A focus was on using non-standard network topology, consumer-grade Internet access lines, and low-cost hardware. At the time of writing of this

article, Towalink has been in active use in this setup for approx. 24 months.

The setup is comprised of eight sites interconnected in a mixed topology as shown in Figure 3. The four sites denoted with A to D are fully meshed; E is only connected to two of these sites; F is a stub site that is only connected to site A without any redundant links. Sites G and H are even completely separated from the others to therewith show that a single controller can also manage such scenarios. This mixed topology provides different scenarios and behavior in the event of link outages or complete site outages.

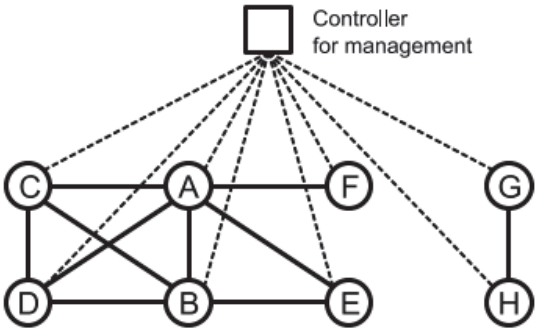

Fig. 3. Evaluation topology with eight sites within different parts of Germany.

Table I shows that the setup in this topology can be represented by forming four groups where in each group the sites are fully meshed. This results in 11 links needed. Due to the link between A and B being present within two groups, only 10 links need to be configured effectively.

TABLE 1: FULL MESH WITHIN EACH SITE GROUP.

Sites in Group	Included Sites	Resulting Links
A, B, C, D	4 sites	6 links
A, B, E	3 sites	3 links
A, F	2 sites	1 link
G, H	2 sites (separated)	1 link
		11 links total

For the different sites, hardware and connectivity vary to get a quite inhomogeneous setup. The sites are located in different parts of Germany and are mostly connected cost-effectively via consumer DSL access lines provided by Deutsche Telekom, Inexio, and Telefónica o2. These lines have different link speeds and use consumer DSL routers with built-in DSL modems for access line termination. Site A is a virtualized environment in a Hetzner data center. Sites A, B, G, and H run Towalink Nodes on Alpine Linux on x86 virtual machines. Site C runs its Node on Debian Linux on dated low-end x86 hardware. The three other sites use ARM-based Raspberry Pis with Pi OS installed. The Controller is hosted on Debian Linux on another x86 virtual machine in the data center of site A.

Since we run most of the Towalink Nodes behind DSL routers on consumer DSL lines, we need to cope with the properties of such a setup. First, we want to avoid the need to configure the DSL routers, e.g. to setup a port forwarding as this would counteract the target of simple, zero-touch installation. Second, public IP addresses are dynamically assigned by Internet service providers (ISPs) and change either daily or at least on every new Internet connection establishment. Third, some ISPs no longer assign customer-unique global IPv4 addresses dynamically but use DS-Lite where they only assign private IPv4 addresses, tunnel the IPv4 traffic via IPv6, and do a carrier-grade NAT to a customer-shared pool of public IPv4 addresses on ISP side.

To cope with the dynamically changing public IP addresses, we use DynDNS to assign hostnames that resolve to the currently assigned addresses. As WireGuard only resolves hostnames once (when the WireGuard interface is configured), we use the wgtrack utility [19] to monitor the operation of WireGuard links and re-resolve the hostnames after a link ceased to function. wgtrack implements a flexibly configurable retry mechanism with exponential backoff capability.

We experimented with UDP hole punching [20] [21] to traverse the network address translation done by the DSL routers for IPv4. However, the complexity like rendezvous servers and the increasing use of DS-Lite by the ISPs made it much more attractive to rely on IPv6 instead. Connectivity with dynamic IPv6 prefix assignment is offered by all ISPs in our setup.

With IPv6, we just need to pass the stateful firewalls of the DSL routers since there is no NAT involved. The WireGuard endpoints on both sides of a to-be-established link send WireGuard UDP packets to the IPv6 address of the peer. This address was resolved and configured by wgtrack. As both peers do this, the firewalls of the DSL routers allow the flow of UDP datagrams bi-directionally. One could call this UDP firewall hole punching. This approach has proven to work well and reliably with the DSL routers of all the ISPs involved. Site A has a statically assigned public IPv6 address and is therewith always reachable easily.

All this provides for WireGuard links working reliably over consumer-grade DSL lines. Only in case of ISPs that force a daily interruption of the DSL link, this of course also affects the site connectivity via Towalink. As we do not have redundant Internet connectivity, such events reduce overall uptime.

Fig. 4. Screenshot of Grafana monitoring dashboard.

Monitoring is implemented using open-source tooling, too. The Telegraf data collection agent collects data from the routers. The already mentioned wgtrack utility [19] provides data on the network traffic of the WireGuard links to Telegraf. The collected data gets ingested into an InfluxDB time series database that is well suited for storing metrics data. Grafana dashboards are used for visualization of the collected data, see Figure 4.

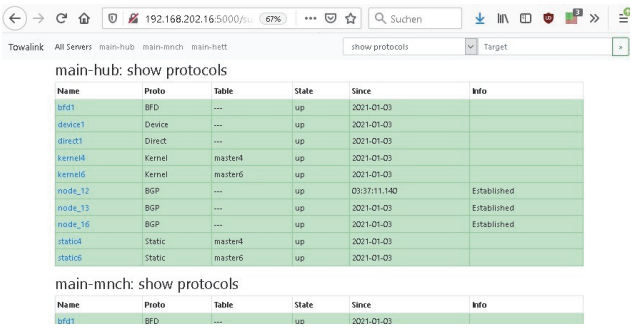

Fig. 5. Screenshot of BGP Looking Glass.

A BGP looking glass application was set up to visually interact with the site routers' Bird BGP daemons, see a screenshot in Figure 5. We selected a lean, modern implementation called "bird-lg-go" [22] that got its name from using the Go programming language. We contributed to that project to better match Towalink's needs by implementing display name support and support for clients connected via IPv6 link local addresses. The tool provides a good overview and allows showing route tables and other data conveniently via web browser.

Towalink inherits the performance of the WireGuard links that it is built upon. In our real-life setup, throughput is only limited by the DSL access line speeds, i.e. the achievable download rates and especially the lower upload rates. This is due to WireGuard's elliptic curve cryptography being that efficient that even on low-end hardware like RaspberryPis the crypto performance is no limitation. Copying large video files over the WireGuard WAN links works smoothly and reliably, see Figure 6. Also, interactive applications like remote desktop access work flawlessly in tests as well as in regular use.

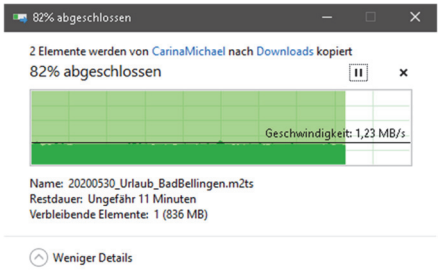

Fig. 6. Stable throughput while copying large video files from remote Samba share.

Bidirectional Forwarding Detection (BFD) was configured with the default multiplier of 5 and an interval of 500 ms for checking the liveliness of the link neighbors. With these settings, link failures are detected and rerouting over alternative paths takes place in under 3 seconds. This is tolerable for many applications but could be reduced as needed by lowering the BFD interval towards to the much lower Bird default. However, lowering the interval comes at the cost of increased overhead traffic on the links. Once a previously failed link comes active again, rerouting takes place without traffic interruption.

In our practical setup, the Internet access lines have shown to be the limiting factor in terms of availability. Especially the forced daily interruptions by some ISPs result in short outage times (up to approx. 2 minutes) of the affected sites. After such an event, the WireGuard links towards the data center site come back up first. Only after registration of a changed public IP address in DynDNS and expiry of the records time-to-live (60s with the DynDNS service used), the direct connectivity between sites that rely on firewall hole punching comes back up and a rerouting towards the direct route takes place.

The solution worked reliably during the approx. 24 months of use. Where redundant links are in place, outages were compensated automatically by using alternative paths once BFD detected the outages. Outages seen were caused by power interruptions, by planned maintenance work, by ISP-forced regular daily interruption of some of the consumer DSL lines, and by unplanned interruption of DSL lines. Hardware failures did not occur so that no replacements had to be done.

From operational point of view, adding additional sites was a simple task. In such a case, the hierarchical configuration and the Jinja2 templating (see Sec. III) proved powerful. However, the practical tests showed that it is helpful to have a very modular Bird BGP daemon configuration right from the start so that no later rework is needed to be able to have special configurations. Configuration parts can then be overridden easily on lower levels if needed.

In summary, the evaluation in the practical setup illustrates that Towalink can provide a reliable SD-WAN solution that achieves the five key aspects as specified in Sec. III. Detailed performance measurements and studies for more complex scenarios are future work.

V. Conclusion

This paper presents Towalink, a novel open-source solution for lightweight SD-WANs. It implements a simple, modern site connectivity solution based on well-established open-source tools and networking protocols such as WireGuard and BGP. Towalink is publicly available on GitHub [22] and well documented on Read-the-Docs [23]. To not reinvent the wheel, Towalink is architected based upon proven, state-of-the-art building blocks. For management and configuration, an Infrastructure-as-Code (IaC) approach has recently been implemented which allows detailed version control and simplifies a rollback in case of configuration errors. The solution has been in use in a real-world setup for many months and has proven reliable and practical. Many design decisions have been confirmed. Future work will consider automated monitoring solutions in more detail, add traffic prioritization/de-prioritization on application-level, and include extensive performance measurements.

ACKNOWLEDGMENT

The authors thank all users who tested the solution and helped to improve it. They also thank the reviewers for their valuable feedback.

REFERENCES

[1] Red Hat, Inc., *Ansible Website,* https://www.ansible.com/ (accessed Jun. 5, 2023).
[2] J. A. Donenfeld, „WireGuard: Next Generation Kernel Network Tunnel,“ in *Proc. of 24th Annual Network and Distributed System Security Symposium*, 2017.
[3] CZ NIC Labs, *BIRD Internet Routing Daemon,* https://bird.network.cz/ (accessed Jun. 5, 2023).
[4] *FRRouting Project,* https://frrouting.org/ (accessed Jun. 5, 2023).
[5] D. Henrici and L. Wischhof, „Site Connectivity with Towalink – Implementing an Open Source "SD-WAN Light",“ in *2022 30th Telecommunications Forum (TELFOR)*, 2022.
[6] S. Kent and K. Seo, „RFC 4301: Security Architecture for the Internet Protocol,“ RFC Editor, 2005.
[7] The strongSwan Team, *strongSwan – the OpenSource IPsec-based VPN Solution,* https://www.strongswan.org/ (accessed Jun. 5, 2023).
[8] OpenVPN, Inc., *OpenVPN website,* https://openvpn.net/ (accessed Jun. 5, 2023).
[9] D. Henrici, *wgfrontend,* https://pypi.org/project/wgfrontend/ (accessed Jun. 5, 2023).
[10] *Algo VPN,* https://github.com/trailofbits/algo (accessed Jun. 5, 2023).
[11] *OPNsense a true open source security platform and more,* https://opnsense.org (accessed Sep. 2, 2022).
[12] flexiWAN Ltd., *flexiWAN,* https://flexiwan.com/ and https://gitlab.com/flexiwangroup (accessed Jun. 5, 2023).
[13] *Nante-WAN,* https://github.com/upa/nante-wan (accessed Jun. 5, 2023).
[14] S. Troia, L. Zorello, A. Maralit und G. Maier, „SD-WAN: An Open-Source Implementation for Enterprise Networking Services,“ in *2020 22nd International Conference on Transparent Optical Networks (ICTON)*, IEEE, 2020, p. 1–4.
[15] M. Artac, T. Borovssak, E. D. Nitto, M. Guerriero und D. A. Tamburri, „DevOps: Introducing Infrastructure-as-Code,“ in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, 2017.
[16] M. Guerriero, M. Garriga, D. A. Tamburri und F. Palomba, „Adoption, Support, and Challenges of Infrastructure-as-Code: Insights from Industry,“ in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2019.
[17] G. Salazar-Chacón und D. M. Parra, „Infrastructure-as-Code in Open-Networking: Git, Ansible, and Cumulus-Linux Case Study,“ in *2023 IEEE 13th Annual Computing and Communication Workshop and Conference (CCWC)*, 2023.
[18] *Jinja2 Template Engine,* https://palletsprojects.com/p/jinja/ (accessed Jun. 5, 2023).
[19] D. Henrici, *wgtrack,* https://pypi.org/project/wgtrack/ (accessed Jun. 5, 2023).
[20] B. Ford, D. Kegel and P. Srisuresh, *State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs),* RFC Editor, 2008.
[21] B. Ford, P. Srisuresh and D. Kegel, „Peer-to-Peer Communication Across Network Address Translators,“ in *Proceedings of the 2005 USENIX Annual Technical Conference, April 10-15, 2005, Anaheim, CA, USA*, 2005.
[22] D. Henrici, ``*Towalink'' Code Repository on Github,* https://www.github.com/towalink/ (accessed Jun. 3, 2023).
[23] D. Henrici, ``*Towalink'' Documentation on Read-the-Docs,* https://towalink.readthedocs.io/ (accessed Jun. 5, 2023).
[24] Y. X. et al., *bird-lg-go,* https://github.com/xddxdd/bird-lg-go (accessed Jun. 5, 2023).
[25] G. Mine, J. Hai, L. Jin und Z. Huiying, „A design of SD-WAN-oriented wide area network access,“ in *2020 International Conference on Computer Communication and Network Security (CCNS)*, 2020.