Towalink

Release v1

Jun 10, 2023

Contents:

Towalink is a multi-site connectity solution. It connects sites using point-to-point VPN tunnels via any available access networks. The default topology is full-mesh, but any other topology can be configured easily. It builds on proven technology like BGP routing and WireGuard. Reliable, flexible, straight-forward.

Caution: This is a young project. It is already being used on production sites. But be aware that there are still rough edges and that it thus might not suit your needs and expectations.

CHAPTER 1

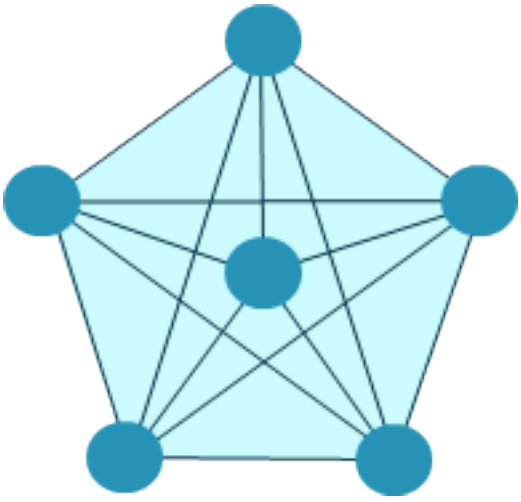
Features & Benefits

Towalink interconnects multiple sites securely and flexibly. It can be considered an "SD-WAN light", a multi-site connectivity solution with central configuration.

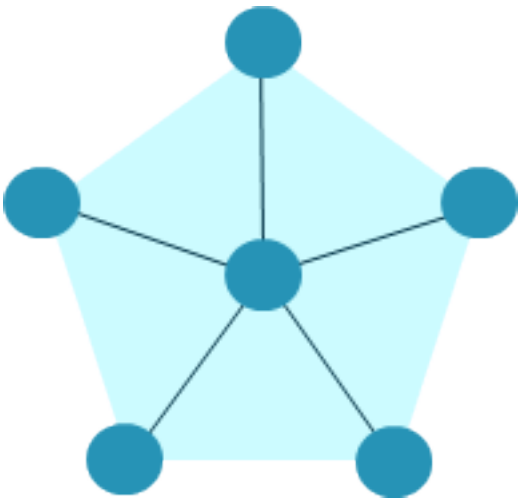
Towalink is built around proven technology like BGP routing (using Bird), VPN tunnels (WireGuard) and Ansible to therewith deliver a powerful but still simple connectivity solution.

1.1 Topologies

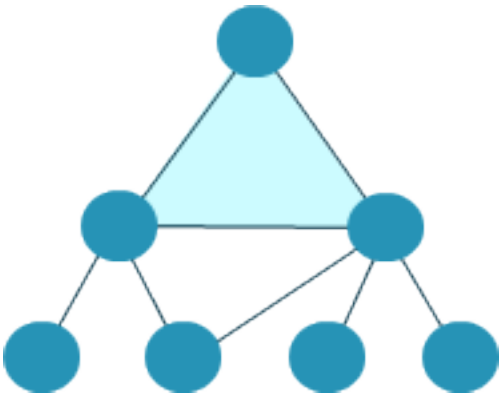
Towalink is completely flexible regarding network topologies.



Full-Mesh Topology: Each site is connected to every other site.



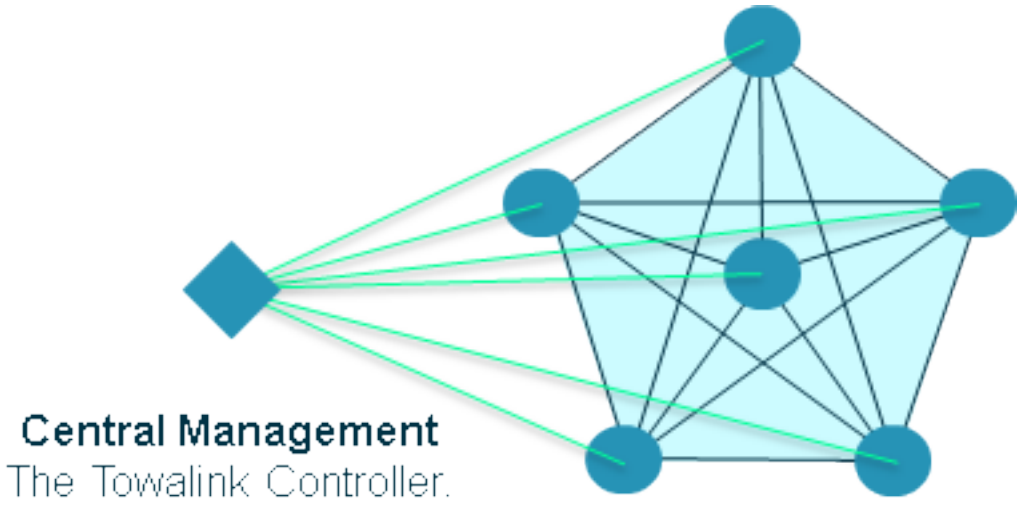
Star Topology: Each site is connected to a central site.



Any Other Topology: Each site is connected to other sites as needed.

1.2 Simple Management

A Towalink installation is managed centrally.

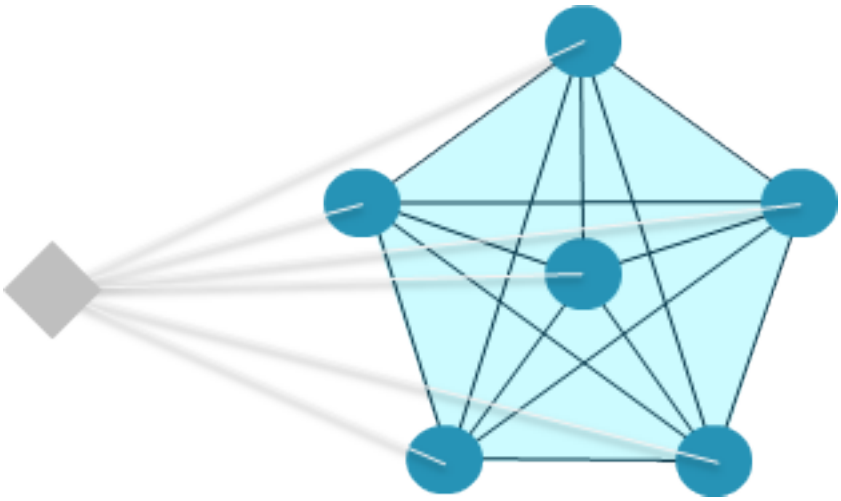


- Fast deployment; the solution is easy to provision and to operate
- Convenient, central configuration and management

The Towalink Controller is the central place of configuration for your Towalink installation. It is also the central place to collect status information and monitoring information from the individual Nodes. Note that user traffic does not run through the Controller.

1.3 Failsafe

Towalink is reliable: site connectivity operates autonomously, i.e. independently of the Controller.



- Connectivity still fully working when Controller not reachable for whatever reasons
- No complex highly-available controller needed like in other solutions

This eases operation considerably compared to other solutions.

1.4 Proven Technology

Towalink builds upon proven, state-of-the art networking technology.

- Supports IPv4 and IPv6 to support any current networking setup
- Builds upon WireGuard: fast & secure VPN tunnels - the state-of-the-art rising star for fast and secure VPN connectivity
- Uses BGP to provide scalable dynamic routing
- Employs BFD (bidirectional forwarding detection) to almost instantaneously detect broken links and foster rerouting

Towalink builds upon proven, state-of-the art open source solutions used worldwide.

- LINUX - Operating Platform
- WireGuard - Secure traffic tunnels
- Bird - BGP-based Dynamic Routing
- Ansible - Configuration Management

. . . and much more great open source software.

1.5 Infrastructure-Agnostic

Towalink runs on arbitrary access technologies and supports high bandwidths even on low-cost hardware.

- Access-Technology-Agnostic: Can be used on top of any Internet access (even ones restricted by DS-Lite if they can reach their direct peers)
- Fast & Efficient: High bandwidths, usually reaching Internet access speeds, even on cost-efficient hardware

The Towalink Controller can be hosted on premise or in the public cloud, as desired.

1.6 Open & Trustworthy

- DSGVO-compliant
- built by network engineers at heart
- no backdoors
- no untrusted vendors
- no closed box

1.7 Flexible

You may use the full power of WireGuard and Bird by managing configuration hierarchically (global scope, site scope, node scope) and adapting configuration flexibly using Jinja2 templates.

CHAPTER 2

Overview & Quickstart

The Towalink multi-site connectivity solution consists of two parts: the Controller and the Nodes (router devices). Towalink is built around proven technology like BGP routing (using Bird) and VPN tunnels (WireGuard) to therewith deliver a powerful but still simple connectivity solution. Node configuration is managed using Ansible (securely via ssh over WireGuard).

2.1 The Towalink Controller

The Towalink Controller is the central place of configuration for your Towalink installation. It is also the central place to collect status information and monitoring information from the individual Nodes.

Apart from that, the Controller is not required in operating your Towalink installation. The connectivity solution itself works fully decentrally so that you don't lose connectivity in case the Controller is not available or cannot be reached for whatever reasons. This eases operation considerably compared to other solutions since the Controller does not need to be highly available. Reliability with simplicity is the mantra here.

The Controller software can either be run on-premises or in the public cloud. It is developed on Debian Linux so that Debian is the preferred distribution to use for the Controller.

The Controller keeps a hierarchical configuration: global-level, Site-level, Node-level. The configuration on each level consists of attribute-value pairs (attribute name and attribute value). There are sensible defaults for almost all values so that you can start right away if desired. Configuration from higher levels is inherited towards lower levels unless explicitly overwritten on lower levels.

- global configuration: configuration data that is used for all Sites and Nodes unless explicitly overwritten at a lower level (i.e. Site level or Node level).
- Site configuration: configuration data that is valid for a given Site, i.e. all Nodes at a given Site. Can be overwritten on Node level.
- Node configuration: configuration data that is valid for a given Node.

The first thing to do for a new Towalink installation is to install the Towalink Controller. See the next subchapter on how to do this.

2.2 The Towalink Nodes

Towalink Nodes are the individual routers located at the different sites. The Node software can even be run on low-cost hardware like the Raspberry Pi (version 4 is recommended due to the Gigabit-capable Ethernet interface) while still providing considerable connection bandwidths.

Nodes connect to the Towalink Controller via a secure tunnel (a WireGuard tunnel) to gather configuration updates and to deliver status information and monitoring data.

Nodes also connect to other Nodes via point-to-point VPN tunnels (WireGuard tunnels) to securely transport and deliver user traffic. The topology can be configured as needed. A full-mesh topology is created by default, i.e. each Node directly connects to every other Node unless something else is configured.

Alpine Linux and Raspbian are fully supported as operating system for the Nodes. Debian and Ubuntu might work, too, but this is not tested.

2.3 Installing the Towalink Controller

The Towalink Controller needs to be installed on Debian Linux. Use a Debian 10 Buster installation that has access to the Internet to be able to download needed software.

Run the following command as user with root privileges to quickly and conveniently install the Towalink Controller:

```
bash <(wget -qO- https://install.towalink.net/controller/)
```

2.4 Configuring your Towalink installation

On the controller, the "tlm" tool (Towalink Manager) is used to configure Sites and Nodes. In the following, a simple installation that uses defaults is shown to get you started quickly. Please see the other parts of the documentation for any advanced features and for customizing your installation.

Note that the "tlm" tool is installed in a Python virtual environment by default. Thus you need to activate that environment to be able to call "tlm" by issuing:

```
source /opt/towalink/venv/bin/activate
```

2.4.1 Adding a Site

To add your first Site (in the example "mysite1"), issue the following command on the Controller:

```
tlm add site mysite1
```

If this is your first command executed, this creates the configuration directory "/etc/towalink" and also creates default template files. Afterwards the Site is created as specified.

2.4.2 Adding a Node

To add your first Node (in the example "primary") on your first Site (in the example "mysite1"), issue the following command:

```
tlm add node primary.mysite1
```

2.4.3 Committing the configuration

Once you added all Sites and Nodes as needed and - if needed - other configuration is done, commit your changes using the following command:

```
tlm commit all
```

This creates a new config version that can be provisioned to the Nodes.

2.4.4 Attaching the Nodes

Now it's time to pair your Nodes with the Controller. This can be done with the following command and only needs to be done once for each Node:

```
tlm attach node primary.mysite1
```

The Node needs to be active and requesting a configuration by running the Node bootstrap script (execute "bash <(wget -qO- https://install.towalink.net/node/) -v -c <hostname/IP of controller>:8000" on the new device). Follow the steps shown to finish the pairing.

After successful attachment, a WireGuard tunnel is established between Controller and Node. The Node can then be managed using Ansible.

2.4.5 Activating the configuration

The latest configuration can be sent to all the Nodes with the following command:

```
tlm activate all
```

2.4.6 Manage your Nodes using Ansible

You can use Ansible and Ansible Playbooks to manage your Nodes like this:

```
tlm ansible node primary.mysite1 all -a whoami
```

CHAPTER 3

Detailed Documentation

3.1 Towalink Overview

The Towalink multi-site connectivity solution consists of two parts: the Controller and the Nodes.

The Nodes are the router devices located at the different sites that shall be interconnected with each other. The interconnection is done securely using WireGuard tunnels via any available access. The interconnection topology can be configured flexibly as needed; full-mesh is the default. The Nodes run the Bird routing daemon to provide dynamic routing using BGP. Broken links are detected quickly using BFD. Built around such proven technology, Towalink delivers a powerful but still simple connectivity solution.

The Controller manages all the Nodes conveniently from a central place. It is securely connected to the Nodes via WireGuard links. Node configuration is managed by the Controller using Ansible and rsync (securely via ssh over WireGuard).

3.1.1 Autonomous Operation

In most SDN solutions, the Controller acts as the "brain" of the installation and the individual devices are dependent on it. In such a set-up, the Controller needs to be highly-available and needs to be implemented in a cluster or other special set-up to avoid single points of failures.

Towalink is different. The Towalink Controller configures and monitors the individual devices (the Nodes). But apart from that, Nodes operate autonomously. The Nodes store the configuration locally and can operate based on this configuration. Since the configuration is stored persistently, it is still available on Node restart. Since the Controller is thus only needed for Node management, especially performing configuration changes, and monitoring, it does not need to be highly available. This makes operation of the installation way simpler compared to a HA set-up needed for other solutions.

3.1.2 Configuration

One of the main purposes of Towalink is to provide a convenient way to centrally manage a fleet of routers, especially WireGuard links and Bird on the routers. Towalink has various means to ease the task.

Basically, there are two kinds of configuration on the Controller: configuration files with attribute-value pairs and service configuration files. * “Configuration files with attribute-value pairs” are files in the “toml” or “yaml” format in which attribute values are assigned to attribute names. Both “toml” and “yaml” are widely used configuration file formats. * “Service configuration files” are the usual configuration files of services used. They can be customized using Jinja templating. For instance, the main service configuration file for the Bird routing daemon is “bird.conf.jinja”, i.e. the naming convention is the usual service configuration filename with the suffix “.jinja”. The attribute-value pairs from the “configuration files with attribute-value pairs” are available for use in the Jinja templates.

Configuration Hierarchy

The configuration is stored under the path “/etc/towalink”. Global configuration is stored directly in that folder. Attribute-value pairs are stored in a file named “config.toml” on the Controller. Other files are service configuration files. The Bird configuration is stored in files with the naming pattern “bird*.conf”.

Using the Jinja templating, one would already be able to configure individual Nodes flexibly using “if” statements and other means. But this would become more and more confusing with an increasing number of Nodes. Therefore, hierarchical configuration is available.

For each Site, there is a subfolder “/etc/towalink/site_<sitename>”. It may contain the same files as the global configuration in “/etc/towalink”. Any files present in the subfolder replace files of the global configuration with the same name. Any attribute-value pairs in “config.toml” within the subfolder extend or replace the attribute-value pairs in the “config.toml” of the global configuration.

Similarly, there is a subfolder within each Site folder for each Node: “/etc/towalink/site_<sitename>/node_<nodename>”. These subfolders contain a “config.toml” and service configuration files that override attribute-value pairs and files of Site level and global level.

Instead of looking into the “config.toml” files to see the attribute-value pairs you can use the “show” command of the tlm tool on the Controller to display them:

```
tlm show global
tlm show site <sitename>
tlm show node <nodeid>|<nodename.sitename>
```

The “complete configuration” of a Site or Node that is obtained by evaluating the configuration hierarchy can be displayed using the tlm tool, too. Use the “show_all” command to see the complete attribute-value pairs to be applied on the respective level:

```
tlm show_all site <sitename>
tlm show_all node <nodeid>|<nodename.sitename>
```

See the *Command Reference* for details.

Generated Configuration

A “generated configuration” is created and updated automatically and stored in “/etc/towalink/generated/config.toml”. It only contains attribute-value pairs, no service configuration files. For instance, it stores encryption keys for the VPN links to be created between Nodes. You can modify this configuration, but there is usually no need to do so.

Effective Configuration

The “effective configuration” is created for each node based on the “complete configuration” (obtained by evaluating the configuration hierarchy) and the “generated configuration”. It is stored below “/etc/towalink/effective”. This is the configuration that gets provisioned towards the individual Nodes.

For each Node, a yaml file with attribute-value pairs is stored. By convention, it contains some generic attributes and the needed attributes for each service. For instance, attributes for the Bird routing daemon are prefixed by "bird_". Additionally, the service configuration files (from the complete configuration) are available. All service configuration files that are ".jinja" template files are processed using the yaml file with the attribute-value pairs.

The "effective configuration" is versioned, i.e. there can be multiple config versions for each Node. Each time, "tlm commit..." is executed for a Node and there are configuration changes, a new config version is created.

```
tlm commit all
tlm commit site <sitename>
tlm commit node <nodeid>|<nodename.sitename>
```

All config versions of a Node are mirrored to the respective Node. Note that committing the configuration and mirroring it to the Node does not activate the new configuration on the Node. This is done separately.

Using the "tlm activate..." command, one of the available configuration versions can be activated on one or many Nodes. For instance, the following command activated the latest configuration on each Node:

```
tlm activate all
```

See the *Command Reference* for details and information on more fine-grained control of configuration activation.

3.1.3 Routing Configuration

Each Node runs the Bird routing daemon to implement dynamic routing via BGP and optionally also Babel. It is proven software that is in use by ISPs and other companies all over the world. You have full flexibility to configure Bird as needed. The Bird configuration can be modularized into several files. Using Towalink, each file can either be used globally for all Nodes or provide configuration for specific Sites or specific Nodes. This way you can implement whatever needed while minimizing effort and avoiding copy & paste errors.

Default Routing Configuration

Towalink comes with a default configuration that is providing fault-tolerant dynamic routing for IPv4 and IPv6. Adapt it as desired in case it does not suit your needs.

The default Bird configuration is designed and implemented as follows:

- Each Site represents an own autonomous system.
- Only BGP is used, there is no IGP (this is an unusual configuration but it has some benefits like simplicity).
- Direct eBGP peering is used across WireGuard link between Sites.
- The WireGuard links are supervised using BFD (Bidirectional Forwarding Detection) to detect and act upon broken links.
- BGP peerings are done using a Node-individual IPv4 loopback address (also used as BGP router id) that is also configured on WireGuard point-to-point links.
- BGP peerings between Sites are done directly over the WireGuard links, no BGP multihop.
- All addresses of local interfaces are announced via BGP.
- This already ensures IPv4 reachability. Note that certain IPv4 loopback address become unreachable on link failures. You thus should not do reachability tests with the loopback address as the source address or destination address.
- For IPv6, the WireGuard links only have link local addresses.

- Bird has static IPv6 routes configured to make Node-individual IPv6 loopback addresses reachable via the direct WireGuard links. Note that certain IPv6 loopback address become unreachable on link failures. You thus should not do reachability tests with the loopback address as the source address or destination address.
- "bird.conf" includes multiple other "bird_*.conf" configuration files to make the configuration more modular.
- You may add additional include files as needed (the naming convention "bird_*.conf" needs to be used, however).
- You may overwrite global "bird_*.conf" configuration files on Site level or on Node level as needed.
- You may modify times or any other configuration as needed.
- You may create a completely different configuration if desired (the naming convention "bird_*.conf" needs to be used, however).

3.2 Towalink Controller Installation

The Towalink Installer is used to install the Towalink Controller. Doing so is the first step in creating a new Towalink installation.

3.2.1 Environment

The Towalink Installer can be used on Debian Linux. Use a Debian 10 Buster installation. Installation on Ubuntu might work as well but this is not tested so that issues can be expected.

The Towalink Controller can be run on top of any virtual machine (or physical server), be it within an on-premise installation or in the public cloud (using AWS EC2, Google Compute Engine, Azure Virtual Machines, or other vendors). Running it in a container is currently not supported since WireGuard support on the host would be required which cannot be installed automatically.

The machine needs to have Internet access to be able to download needed software. Later on, the Internet access is needed to connect with the Towalink Nodes on the sites to be connected with each other.

3.2.2 Installation

Run the following command as user with root privileges to quickly and conveniently install the Towalink Controller. It downloads and executes a bash script. The script is a wrapper around an Ansible playbook used to install the Controller. Before executing the playbook, Ansible and git are installed as prerequisites, and the environment is prepared.

```
bash <(wget -qO- https://install.towalink.net/controller/)
```

Optionally, you may add parameters to control what's happening and to already provide answers to questions that would be asked later (to thus avoid interactive install).

```
bash <(wget -qO- https://install.towalink.net/controller/) [<parameters...>]
```

The mentioned parameters can be any command line arguments that are available for an Ansible playbook. The following variables can be provided with the "-e" parameter:

- "-e input_welcomeprompt=skip"
  Do not show the welcome prompt.
- "-e input_monitoring=y" or "-e input_monitoring=n"

Specifies whether monitoring shall be installed or not. If not specified, it is asked interactively whether to install or not.

- "-e python_venv="
  Do not install within a Python virtual environment. The installation is done in "/opt/towalink/venv" by default.

Parameters used for development and testing purposes:

- "-e pip_test=yes"
  Do install the Towalink module from the PyPi test repository and not from the main repository.
- "-e tlm_path=/home/towalink/controller_tlm"
  Do not install the tlm tool from PyPi but install it locally from the path specified.

3.3 Using the Towalink Controller

The Towalink Controller is used for managing your Towalink installation conveniently from a central place.

3.3.1 The tlm Tool

The "tlm" tool is used to manage your Towalink installation. It was installed during the installation of the Towalink Controller as described above.

In case you installed using the default parameters, the "tlm" tool was installed in a Python virtual environment. Therefore you need to enter this environment before using the "tlm" tool. This is done using the following command:

```
source /opt/towalink/venv/bin/activate
```

Afterwards, you can call the "tlm" tool with the parameters needed. Here is the generic usage - but just see the next sections on practical examples:

```
tlm [-?|--help] [-l|--loglevel debug|info|error] <operation> <entity> [<arguments...>]
```

3.3.2 Creating Site and Node Configuration

Use the "tlm" tool to create new Sites as needed using the "add site" operation:

```
tlm add site mysite1
```

Similarly, you can create Nodes within the Sites as needed using the "add node" operation:

```
tlm add node primary.mysite1
```

Each Node gets a Node identifier assigned. Use either this Node identifier (<nodeid>) or the notation "<nodename>.<sitename>" to identify a Node when using the "tlm" tool.

3.3.3 Attaching Nodes

New Nodes need to be "paired" with the Towalink Controller. This is called "Node attachment". This is a one-time process for each new Node. Once the attachment is done, the Node maintains a WireGuard tunnel to the Controller. Via this tunnel, the Node is managed by the controller in a secure manner.

Firewall

The "tlm" tool starts a web server on port 8000 to receive configuration requests from the unconfigured Nodes. This web server needs to be reachable for the Nodes. The port 8000/tcp only needs to be open during the Node attachment process.

Server Certificate

The web server of the "tlm" tool that is used for attaching the nodes requires an SSL certificate. The private key is expected in "/etc/towalink/certs/key.pem", the public key in "/etc/towalink/certs/server.pem".

A certificate can be created using openssl, for instance:

Note that the common name of the certificate needs to match the hostname of the Towalink Controller.

Be aware that the Nodes need to be able to verify the certificate chain for a successful attach. In case of a self-signed certificate, you thus need to provide the CA certificate or the server certificate to the Nodes.

Name Resolution

You need to make sure that the DNS name resolution is working properly on your Towalink Controller. This means that the Controller's hostname needs to resolve to the Controller's IP address (entry in "/etc/hosts" for the hostname is not just 127.0.0.1 but the interface IP address). In addition, the Controller needs to be able to resolve the hostnames of the Nodes to be attached to the Nodes' IP addresses using which the Nodes can be reached.

> **Attention:** If name resolution is not working properly, you won't be able to attach Nodes successfully!

Attachment Process

The attachment process is initiated by the following command:

```
tlm attach node <nodeid>|<nodename.sitename>
```

Once started, the "tlm" tool starts the web server and collects Nodes' connection attempts for 20 seconds. Afterwards you select which of these Nodes to configure or interrupt the process.

The next connection attempt of the selected Node provides data on the Node and is answered with needed configuration information to establish the WireGuard management tunnel for that Node. Once the data has been exchanged, the "tlm" tool configures the WireGuard management tunnel endpoint on the Controller and waits for the management tunnel to come up.

Once the management tunnel is working, the "tlm" tool prepares the new Node using an Ansible playbook. Ansible connects securely via ssh over the WireGuard management tunnel. The "tlm" tool prints the Ansible playbook command that is used to prepare the Node so that you can reissue the command in the case of problems.

After the Node has been prepared using Ansible, the attachment process is finished. You now can manage the Node using the "tlm" tool.

3.3.4 Managing Nodes

The Node configuration is managed directly using the "tlm". Any other management tasks regarding the Nodes are done using Ansible. You can use any Ansible functionality (Ansible and Ansible Playbooks) to do this.

Depending on the "tlm" command, either a single Node, all Nodes of a Site, or the whole installation can be targeted. See the *Command Reference* for details.

Using Ansible

Ansible and Ansible Playbooks can be called using the following commands:

```
tlm ansible node <nodeid>|<nodename.sitename> <ansible arguments...>
```

```
tlm ansible_playbook node <nodeid>|<nodename.sitename> <ansible_playbook arguments...>
```

"tlm ansible" and "tlm ansible_playbook" are just wrappers around Ansible and Ansible Playbooks, respectively. These wrappers add inventory information so that the Node(s)/Site(s) as specified are targeted.

Preparing and Mirroring Node Configuration

The "tlm commit" updated the effective configuration. In case the configuration changed compared to the previous one, a new configuration version is created. All available configuration versions are then mirrored to the respective Node so that the configuration becomes available there. Note that a changed configuration is not applied with this command.

```
tlm commit node <nodeid>|<nodename.sitename>
```

Activating Node Configuration

Use the "tlm activate" operation to apply a certain configuration version:

```
tlm activate node <nodeid>|<nodename.sitename> [<version>]
```

If no version is specified, the latest configuration version is activated.

A symbolic link to the requested config version is set on the Node. Afterwards the changed configuration files are copied to their needed location. Finally, all services with changed configuration files are reloaded or restarted. Then the requested config version is active on the Node.

3.4 Preparing Towalink Nodes

New Towalink Nodes can be easily prepared for central management by your Towalink Controller.

3.4.1 Environment

There are currently two operating systems that are tested and known to work:

- Alpine Linux (at time of writing v3.12 is current)
- Raspberry Pi OS (previously called "Raspbian") (based on Debian v10 Buster)

Alpine Linux uses the apk packet manager, Raspberry Pi OS the apt packet manager. Distributions based on the yum package manager (e.g. CentOS) are not yet supported.

3.4.2 Preparation

You need to make sure that the operating system is properly installed and that you have working Internet access on the primary network interface.

Check that the Node's hostname is properly set:

```
hostname -f
```

This command must return the fully qualified domain name of the Node. If this is not yet the case in your installation, you usually need to correct the "/etc/hosts" file.

It is important that name resolution works properly. This means that DNS hostnames need to be resolvable towards their corresponding IP address. Make sure that the hostname of the Node itself resolves to the Node's (public) IP address and not to localhost (127.0.0.1).

3.4.3 Bootstrap Script

The bootstrap scripts installs basic required packages and regularly attempts to download a Node config from the Controller. Once the Node could be configured and the management connection to the Controller is up and running, the bootstrap script exits. The bootstrap script installs itself on the Node to that it is executed on each boot.

Run the following command with root permissions to install and execute the bootstrap script:

```
bash <(wget -qO- https://install.towalink.net/node/) -v -c <hostname/IP of controller>
↪:8000
```

In case you do not have a shell that supports this notation (like ash on Alpine), wrap it into a bash command like this:

```
apk add bash
bash -c "bash <(wget -qO- https://install.towalink.net/node/) -v -c <hostname/IP of␣
↪controller>:8000"
```

The script parameters are optional. Additional information on them:

- "-v" enables verbose logging
- "-c <hostname/IP of controller>:8000" denotes the Controller to connect to. If omitted, the Node attempts to find its Controller via "towalink.net"

Certificate Chain

The bootstrap script attempts to download the Node configuration. For security reasons, the server SSL certificate is validated. In case of an invalid certificate, the config download is not done. You will then see the message "Bootstrap config download failed with http response 00060. Ignoring and continuing" in the log.

In case your Controller uses a self-signed server certificate, you need to provide the server certificate or the certificate of a trusted certification authority in the following file:

Restart the Node after providing or changing the certificate file.

Recovery mechanism

There exists an additional mechanism to recover Nodes that no longer have a Controller or that have other serious issues that make the management connection fail. On each boot, the bootstrap script attempts to download a recovery script

from the “towalink.net” infrastructure. In case this download succeeds and the script has been validated successfully, the recovery script is executed.

Once the Node gets attached to a Controller, a security token (recovery token) is generated and stored in the Controller. Only recovery scripts that contain the correct security token are executed. Due to this security measure, the “towalink.net” infrastructure can’t execute any code on your Nodes so that this feature cannot be abused as a backdoor. Only in the case that a management connection could not be established and the machine has been restarted five times in a row without a successful management connection, the token security check is disabled. This feature is present to consider the case of a lost recovery token.

3.5 Frequently Asked Questions

The following sections provide answers to questions that have been raised towards the Towalink project community.

3.5.1 Why is this project called “Towalink”?

“towa” is Japanese for “forever”, “permanent”. “link” relates to the English network term: “having a link”, “being connected”. The “Towalink” therewith relates to its target: providing reliable connectivity.

CHAPTER 4

Command Reference

4.1 "tlm" tool on the Towalink Controller

"tlm" is the interface for configuring your Towalink installation. You can use it to modify the configuration files and to trigger actions.

4.1.1 Listing entities

Listing Sites

Command:

```
tlm list sites
```

Parameters:

no parameters

Example:

```
tlm list sites
```

Listing Nodes

Command:

```
tlm list nodes all|<sitename>
```

Parameters:

<sitename>: name of the Site whose Nodes shall be shown; "all" for Nodes of all Sites

Examples:

```
tlm list nodes all
tlm list nodes mysite1
```

4.1.2 Showing the configuration

Showing the global configuration

Command:

```
tlm show global
```

Parameters:

no parameters

Example:

```
tlm show global
```

Showing the Site configuration

Command:

```
tlm show site <sitename>
```

Parameters:

<sitename>: name of the Site whose configuration shall be displayed

Example:

```
tlm show site mysite1
```

Showing the Node configuration

Command:

```
tlm show node <nodeid>|<nodename.sitename>
```

Parameters:

- <nodeid>: numeric node identifier
- <nodename.sitename>: name of the Node within the given Site (identified by its name)

Examples:

```
tlm show node 15
tlm show node primary.mysite1
```

Showing the complete Site configuration

Command:

```
tlm show_all site <sitename>
```

Parameters:

<sitename>: name of the Site whose complete configuration shall be displayed

Example:

```
tlm show_all site mysite1
```

Showing the complete Node configuration

Command:

```
tlm show_all node <nodeid>|<nodename.sitename>
```

Parameters:

- <nodeid>: numeric node identifier
- <nodename.sitename>: name of the Node within the given Site (identified by its name)

Examples:

```
tlm show_all node 15
tlm show_all node primary.mysite1
```

4.1.3 Creating and removing entities

Creating a Site

Command:

```
tlm add|create site <sitename>
```

Parameters:

<sitename>: name of the Site that shall be added

Example:

```
tlm add site mysite1
```

Creating a Node

Command:

```
tlm add|create node <nodename.sitename>
```

Parameters:

- <nodeid>: numeric node identifier

- <nodename.sitename>: name of the Node within the given Site (identified by its name)

Example:

```
tlm add node primary.mysite1
```

Removing a Site

Command:

```
tlm del|remove site <sitename>
```

Parameters:

<sitename>: name of the Site that shall be added

Example:

```
tlm del site mysite1
```

Removing a Node

Command:

```
tlm remove node <nodeid>|<nodename.sitename>
```

Parameters:

- <nodeid>: numeric node identifier
- <nodename.sitename>: name of the Node within the given Site (identified by its name)

Examples:

```
tlm del node 15
tlm del node primary.mysite1
```

4.1.4 Modifying the configuration

Setting the global configuration

Command:

```
tlm set global <attr> <value>
```

Parameters:

- <attr>: name of the attribute that shall be set
- <value>: new value for the attribute; special value “empty” to remove the attribute

Example:

```
tlm set global wg_keepalive 25
```

Setting the Site configuration

Command:

```
tlm set site <sitename> <attr> <value>
```

Parameters:

- <sitename>: name of the Site whose configuration shall be changed
- <attr>: name of the attribute that shall be set
- <value>: new value for the attribute; special value “empty” to remove the attribute

Example:

```
tlm set site mysite1 wg_keepalive 25
```

Setting the Node configuration

Command:

```
tlm set node <nodeid>|<nodename.sitename> <attr> <value>
```

Parameters:

- <nodeid>: numeric node identifier
- <nodename.sitename>: name of the Node within the given Site (identified by its name)
- <attr>: name of the attribute that shall be set
- <value>: new value for the attribute; special value “empty” to remove the attribute

Example:

```
tlm set node 15 wg_keepalive 25
tlm set node primary.mysite1 wg_keepalive 25
tlm set node main.hubsite groups '[ "hubs" ]'
```

4.1.5 Config change management

List Nodes with changed configuration

Command:

```
tlm list changed
```

Parameters:

no parameters

Example:

```
tlm list changed
```

Commit configuration of all Nodes

Command:

```
tlm [-m <message>] commit all
```

Parameters:

<message>: commit message

Example:

```
tlm -m "Change debug level" commit all
```

Commit the configuration of a Site's Nodes

Command:

```
tlm [-m <message>] commit site <sitename>
```

Parameters:

- <message>: commit message
- <sitename>: name of the Site whose configuration shall be committed

Example:

```
tlm -m "Change debug level" commit site mysite1
```

Commit the configuration of a single Node

Command:

```
tlm [-m <message>] commit node <nodeid>|<nodename.sitename>
```

Parameters:

- <message>: commit message
- <nodeid>: numeric node identifier
- <nodename.sitename>: name of the Node within the given Site (identified by its name)

Example:

```
tlm -m "Change debug level" commit node 15
tlm -m "Change debug level" commit node primary.mysite1
```

4.1.6 Activate Node configuration

Activate the latest configuration on all Nodes

Command:

```
tlm activate all
```

Parameters:

no parameters

Example:

```
tlm activate all
```

Activate the configuration on all Nodes of a Site

Command:

```
tlm activate site <sitename> [<version>]
```

Parameters:

- <sitename>: name of the Site whose configuration shall be activated
- <version>: version number of the configuration; "latest" is default

Examples:

```
tlm activate site mysite1
tlm activate site mysite1 v5
```

Activate the configuration of a single Node

Command:

```
tlm activate node <nodeid>|<nodename.sitename> <version>
```

Parameters:

- <nodeid>: numeric node identifier
- <nodename.sitename>: name of the Node within the given Site (identified by its name)
- <version>: version number of the configuration; "latest" is default

Example:

```
tlm activate node 15
tlm activate node 15 v5
tlm activate node 15 latest
tlm activate node primary.mysite1
tlm activate node primary.mysite1 v5
tlm activate node primary.mysite1 latest
```

4.1.7 Pairing Nodes

Attach a Node configuration to a physical device

Command:

```
tlm attach node <nodeid>|<nodename.sitename>
```

Parameters:

- <nodeid>: numeric node identifier
- <nodename.sitename>: name of the Node within the given Site (identified by its name)

Example:

```
tlm attach node primary.mysite1
```

4.1.8 Executing Ansible for Node(s)

Executing Ansible for all Nodes

Command:

```
tlm ansible all <ansible arguments...>
```

Parameters:

<ansible arguments...>: arguments for Ansible

Example:

```
tlm ansible all all -a whoami
```

Executing Ansible for all Nodes of a Site

Command:

```
tlm ansible site <sitename> <ansible arguments...>
```

Parameters:

- <sitename>: name of the Site for whose Nodes Ansible shall be called
- <ansible arguments...>: arguments for Ansible

Examples:

```
tlm ansible site mysite1 all -a whoami
```

Executing Ansible for a single Node

Command:

```
tlm ansible node <nodeid>|<nodename.sitename> <ansible arguments...>
```

Parameters:

- <nodeid>: numeric node identifier
- <nodename.sitename>: name of the Node within the given Site (identified by its name)
- <ansible arguments...>: arguments for Ansible

Example:

```
tlm ansible node 15 all -a whoami
tlm ansible node primary.mysite1 all -a whoami
```

4.1.9 Executing Ansible Playbook for Node(s)

Executing Ansible Playbook for all Nodes

Command:

```
tlm ansible_playbook all <ansible_playbook arguments...>
```

Parameters:

<ansible_playbook arguments...>: arguments for Ansible Playbook

Example:

```
tlm ansible_playbook all myplaybook.yml -e test_variable="test"
```

Executing Ansible Playbook for all Nodes of a Site

Command:

```
tlm ansible_playbook site <sitename> <ansible_playbook arguments...>
```

Parameters:

- <sitename>: name of the Site for whose Nodes the Ansible Playbook shall be executed
- <ansible_playbook arguments...>: arguments for Ansible Playbook

Examples:

```
tlm ansible_playbook site mysite1 myplaybook.yml -e test_variable="test"
```

Executing Ansible Playbook for a single Node

Command:

```
tlm ansible_playbook node <nodeid>|<nodename.sitename> <ansible_playbook arguments...>
```

Parameters:

- <nodeid>: numeric node identifier
- <nodename.sitename>: name of the Node within the given Site (identified by its name)
- <ansible_playbook arguments...>: arguments for Ansible Playbook

Example:

```
tlm ansible_playbook node 15 myplaybook.yml -e test_variable="test"
tlm ansible_playbook node primary.mysite1 myplaybook.yml -e test_variable="test"
```

4.1.10 Executing git commands

Run git executable

You may execute any git command for the local repository that contains Towalink's configuration files.

Command:

```
tlm git <git arguments...>
```

Parameters:

<git arguments...>: arguments for git command

Example:

```
tlm git branch new-branch
```

4.1.11 Setting the debug level

You may configure the verbosity of the command by setting the log level parameter to the desired value.

```
tlm --loglevel <loglevel> ...
```

Parameters:

<loglevel>: valid values are "debug", "info", "warning", and "error"

Example:

```
tlm --loglevel debug list sites
```

CHAPTER 5

Developers

Towalink is an open source solution. It intends to become a powerful, open alternative to proprietary, closed solutions. We started with two developers/engineers and aim at creating a vibrant community of users and contributors.

5.1 Links to Important Resources

The following public resources are very relevant to the Towalink project:

- Website: https://www.towalink.net
- Documentation: https://towalink.readthedocs.io
- Github: https://github.com/towalink/

5.2 Help Wanted

The following topics are considered the most desirable areas for your contributions:

- REST API
  The Towalink Controller can be configured using the "tlm" command line interface as of now. The existing functionality should also be exposed via a REST API. Proposed framework: Flask.
- web frontend
  A user-friendly web frontend for configuring the Towalink installation would be nice. It should access the REST API mentioned.
- monitoring improvements, alerting
  Monitoring and alerting in a pre-configured and user-friendly way is considered very important. Help to improve on that.
- documentation improvements
  Any means to improve the project documentation are welcome (corrections, enhancements, tutorials/howtos, success stories, etc.).

- spread the word
  Towalink is not well known to the public yet. Help to spread the word about its existence, e.g. on websites dealing with related topics.

We are also looking for companies that are willing to become reference customers. This shall be for mutual benefit. It should be "friendly customers" that are open to help improve on Towalink. We imagine a feedback loop in which we jointly monitor the experience with Towalink in the company environment and aim at iteratively improving Towalink to suit the practical needs.

5.3 Contributing

When intending to contribute to the Towalink code repositories, please discuss the change you wish to make via issue, email, or any other method with the owners of the repository before making a change.

We use a meritocratic community model. The more you get involved and contribute, the more can you influence the direction of the project.

Please follow our code of conduct in all your interactions with the project.

5.3.1 How to report a bug

Write an email for now; we will switch to a bug tracker in future.

Please include expected behavior, experienced behavior, and any information needed to reproduce the issue. In case you solved the issue for you, please tell what you did or what code you changed.

5.3.2 Style Guide / Coding conventions

For Python: If in doubt, follow the PEP8 style guide. Note that we use a few exceptions, e.g. we do not enforce line length restrictions any more.

5.3.3 Testing

There are no automated tests yet that cover the end-to-end solution. Please ensure that any code changes you propose are thoroughly tested.

Some modules have unit tests. Please make sure that any code changes are tested with these. Please also extend the test coverage to cover new features/etc. of your changes.

5.3.4 How to submit changes

You may merge a pull request in once you have the sign-off of another developer, or if you do not have permission to do that, you may request the reviewer to merge it for you.

5.3.5 How to request an "enhancement"

Write an email if you like to suggest a feature to a project but that is not a bug/problem with the existing code.

In case you request an enhancement for business needs, consider donating to support the development.

5.3.6 Recognition model

Contributions will be listed.

5.3.7 Copyright/license

By your contribution you provide the rights to use your code according to the projects license. This is mostly the AGPL but you also grant permission to the project owner to release the code under different licenses in the future.

5.4 Contributor Covenant Code of Conduct

5.4.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

5.4.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

5.4.3 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

5.4.4 Enforcement

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

See https://www.contributor-covenant.org/version/2/0/code_of_conduct.html for information on enforcement.

5.4.5 Attribution

This Code of Conduct is adapted from the [Contributor Covenant][homepage], version 2.0, available at https://www.contributor-covenant.org/version/2/0/code_of_conduct.html. Community Impact Guidelines were inspired by [Mozilla's code of conduct enforcement ladder](https://github.com/mozilla/diversity). [homepage]: https://www.contributor-covenant.org For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

search