CES manual

Author's Name

October 2, 2019

Abstract

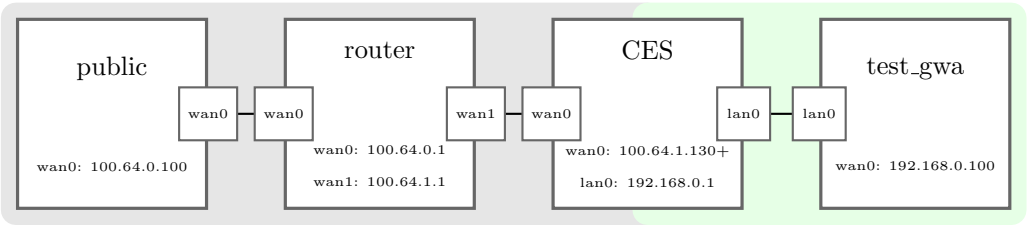
Customer Edge Switching (CES) aims at placing hosts in private networks enabling end-to-end connectivity addressing the reachability issue as well as improved security by using Customer Edge Traversal Protocol(CETP) with Customer Edge Switches. Ultimately, the goal is to replace NAT devices implementing the benefits of NAT switch additional enhanced capabilities such as trust and smart inbound traffic management. CES is a way of moving from the end-to-end principle, popularly employed in the Internet, to the trust-to-trust principle.

1 Simple scenario

1.1 Scenario topology

The scenario consists of 4 network namespaces emulating borderline between the Internet and a local network.

- public: represents a terminal located in a public external network with a global IP address,
- router: a router connecting CES with a public external network,
- CES in a role of NAT and firewall,
- test_gwa: private network.



1.2 Running CES

- Navigate to the `netns` directory as shown in the diagram.

```
└─ orchestration
   └─ netns
```

- Execute the script `doSimpleSetupRGW_v1.sh`

```
$ chmod +x doSimpleSetupRGW_v1.sh
$ ./doSimpleSetupRGW_v1.sh
```

Check if the network namespaces has been created:

```
$ sudo ip netns show
[result]
  public (id: 3)
  router (id: 0)
  gwa (id: 1)
  test_gwa (id: 2)
  default
```

```
$ sudo ip netns exec gwa ./run_gwa.sh
```

1.3 Use case

When the CES has been started successfully, you can test its functioning. Before issuing commands it is recommendable to start a command shell in the respectable network namespaces, i.e.

```
    ip netns exec network_namespace shell
```

where *network_namespace* can be e.g. *public*, *test_gwa* and *shell* can be e.g. *bash*.

1.3.1 Outgoing connectivity: connection initiated by *test_gwa* towards *public* node

`test_gwa$ ping 100.64.0.100`
The response:

```
PING 100.64.0.100 (100.64.0.100) 56(84) bytes of data.
64 bytes from 100.64.0.100: icmp_seq=1 ttl=62 time=0.235 ms
```

1.3.2 Incoming connectivity: connection initiated by *public* towards *gwa* node

`public$ ping gwa.demo`

```
    PING gwa.demo (100.64.1.130) 56(84) bytes of data.
    64 bytes from 100.64.1.130 (100.64.1.130): icmp_seq=1 ttl=63 time=0.134 ms
```

1.3.3 Incoming connectivity: connection initiated by *public* towards *test_gwa* node

[1]. Navigate to the `gwa.demo.policy.d` directory as shown in the diagram. Change the attributes

```
└─ config.d
   └─ gwa.demo.policy.d
```

PBRA_DNS_POLICY_TCP and PBRA_DNS_POLICY_CNAME in `circularpool.policy.yaml` file

[1] Works only with the patch

from `true` to `false`, then issue
`public$ ping test.gwa.demo`

```
PING test.gwa.demo (100.64.1.131) 56(84) bytes of data.
64 bytes from 100.64.1.131 (100.64.1.131): icmp_seq=1 ttl=63 time=0.122 ms
```

Furthermore, on the CES log you should observe something like

```
(...) Policy accepted! (...)
(...) Allocated IP address from Circular Pool: test.gwa.demo. @ 100.64.1.131 (...)
(...) New Circular Pool connection: (test.gwa.demo.) [0] 192.168.0.100 <- 100.64.1.131 (...)
```

With PBRA_DNS_POLICY_TCP set to `true` a connection have to be established using TCP. When PBRA_DNS_POLICY_CNAME establishes that allocation is only allowed via temporary alias names of CNAME responses ... [TODO: more comprehensible description].

1.3.4 Incoming connectivity via TCP: connection initiated by *public* towards *test_gwa* node

Issue the following commands `test_gwa$ nc -l 1234`
`public$ nc test.gwa.demo 1234`
Observe, if the connection is established. Try to type on the console in `test_gwa` something. If the connection is established, the typed text should appear in the console of `public`.

2 For a developer

So, you want to learn how to extend or better understand CES functionality? The goal of this chapter is to teach you some concepts and standards used in CES which allows you to have a smooth start with the CES development.

2.1 Key modules

CES heavily depends on several Python modules whose knowledge contributes to better understating the operation of the software as a whole. The modules can be divided into internal, related to functioning of the systems (e.g. asyncio), and external, adding new functionalities to the software (e.g. python-openvswitch.).

Internal modules

- asyncio: a library to write concurrent code using the async/await syntax. asyncio is used as a foundation for multiple Python asynchronous frameworks that provide high-performance network and web-servers, database connection libraries, distributed task queues, etc. asyncio is often a perfect fit for IO-bound and high-level structured network code.
- functools: a module is for higher-order functions: functions that act on or return other functions. In general, any callable object can be treated as a function for the purposes of this module.

External modules

- iptables: a command line utility for configuring Linux kernel firewall

– ipset: a companion application for the iptables Linux firewall. It allows for efficient setup of firewall rules.
– libipset3
– ebtables
– bridge-utils
– dns: CES communications rely heavily on domain name queries to create a well-defined state with host information and the chosen virtual anchor (proxy-address) for the subsequent forwarding of datapackets. A domain resolution operation must be always placed first in order to create a valid state in CES , obtain a proxy-address and subsequently forward the data packets.

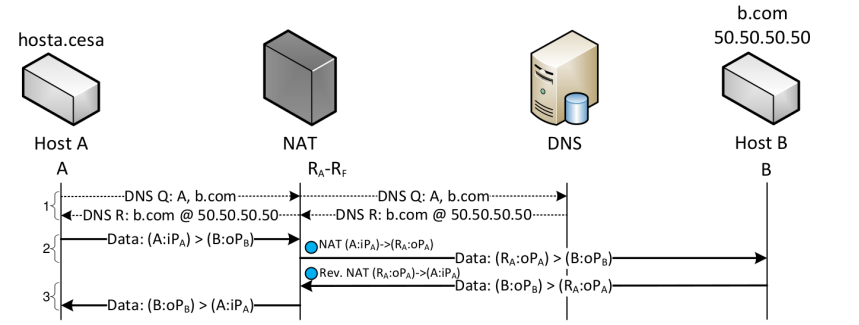
2.2 Components

CES consist of several modules, each of which contains one or more classes.

– datarepository.py: responsible for storage of application configuration parameters, among them configuration file and policy data
– host.py: Consist of 2 classes: a) *HostEntry* a generic class for network nodes. Stores information related to configuration and services (e.g. dns server) running at the hosts. b) *HostTable*: an array which stores objects of class *HostEntry*. The module is dedicated to containerisation.
– callbacks.py: responsible for handling of received DNS queries. Separate functions for processing queries from different network zones. Some functions unfinished.

2.3 Use cases

2.3.1 Outgoing connectivity: connection initiated by *test_gwa* towards *public* node



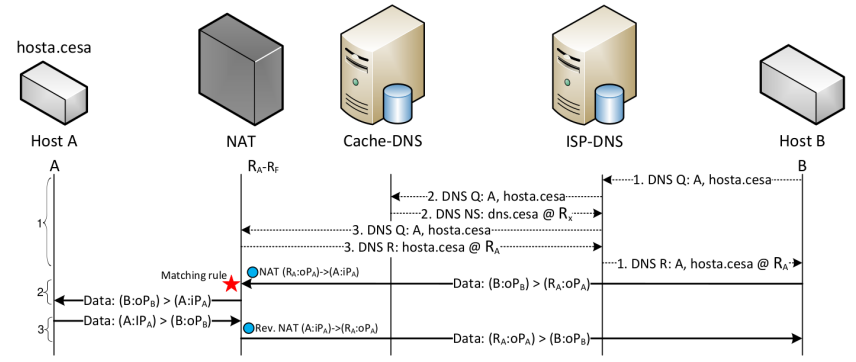
A: Private IP of Host-A	iP_A: Local port of Host-A	oP_A: Public port of Host-A
B: Public IP of Host-B	oP_B: Local/public port of Host-B	R_A: Public IP for Host-A

FIGURE 5.1 INTERNET: OUTGOING CONNECTION

2.3.2 Incoming connectivity: connection initiated by *public* towards *test_gwa* node

For incoming connections, state information is stored each time a DNS query is received. The state comprises different fields such as the originator's IP, the allocated public IP address, the private IP address of the host, status of the entry and a timeout. The public IP addresses are given following a simple circular mechanism starting from the beginning of the pool, selecting the next one each time and coming back to the first one upon reaching the end of the pool. When a DNS query is received, the

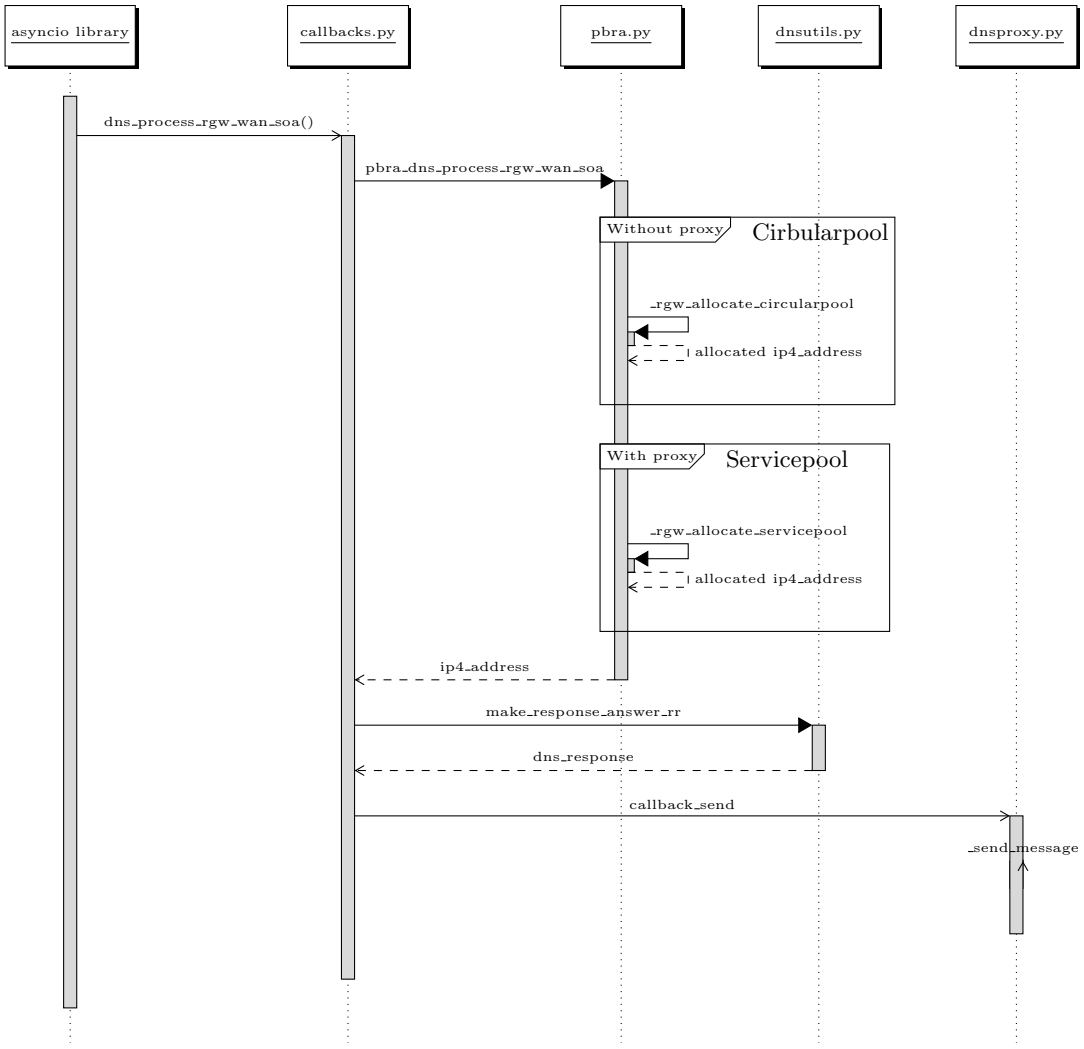
next available address is chosen and marked as in "waiting" status. Because the originator is unknown, the tuple of state information is created with the next information: (unknown, public_IP_in_CES, private_host_IP, waiting, timeout). The actual state information stored in the forwarding table for an active connection includes additional fields regarding the port numbers and the protocol in use. For a better understanding we have decided to use the previously defined tuple of information throughout the rest of the figures



A: Private IP of Host-A	iP_A: Local port of Host-A	oP_A: Public port of Host-A
B: Public IP of Host-B	oP_B: Local port of Host-B	R_A: Public IP for Host-A

Perquisites

- proxy_required = no



3 Code development guide

The first 90 percent of the code accounts for the first 90 percent of the development time. The remaining 10 percent of the code accounts for the other 90 percent of the development time. – Tom Cargill, Bell Labs

Writing software is more than simply creating an application that fulfills its requirements; it has to be created to be maintainable, robust, verifiable, and readable, among many other things. Because of this, we as developers need to constantly strive to learn and find new ways that can help create better code, and ultimately, a better software application.. . .
Write as if someone else will maintain your code. If you are coding for a job, someone else probably will maintain your code some day, and if you're coding for yourself, you'll feel like someone else when you revisit code you wrote 6 months ago and haven't thought about since. In either case, the code should be readily understandable; complexity should be minimized and documented where it's present; changes to one place shouldn't cause inscrutable errors in other places; etc. The following chapters describe how to accomplish these things.

3.1 Readability

Before starting to write, one should understand thoroughly what is the purpose of the program. Furthermore, the you should made a draft of the program architecture, i.e. how, what it will use, how modules, services will work with each other, what structure will it have, how it will be tested and debugged, and how it will be updated.

Try to write code that is simple to read and which will be understandable for other developers. ... Do not try to optimize at the cost of readability because time and resources that will be dedicated to decipher a hard readable code will be much higher than what you get from optimization. If you need to make optimization, then make it like independent module with DI, with 100% test coverage and which will not be touched for at least one year.

3.2 Comments

Comments should be an integral part of a program. They can be introduced on different levels of code: starting from a module-level, through classes, functions or methods, down to inline explanations. Using comments throughout code shed light on its functionality and can help other developers gain an understanding of how it all works very quickly. The comments should follow the conventions described in the following documents:

- PEP 8 – Style Guide for Python Code – coding conventions for the Python code comprising the standard library in the main Python distribution,
- PEP 257 – Docstring Conventions – semantics and conventions associated with Python docstrings,
- Documenting Python Code: A Complete Guide – a practical guide with examples.
- Writing Comments in Python (Guide) – another practical guide

3.3 Static typing

Without type annotations, basic reasoning such as figuring out the valid arguments to a function, or the possible return value types, becomes a hard problem. Examples of a few common questions that are often not trivial to answer without proper typing comments

- Can this function return None?
- What is this items argument supposed to be?
- What is the type of the id attribute: is it int, str, or maybe other?
- What is the type of this argument: a list, a tuple or a set?

```
def fib(n):
    a, b = 0, 1
    while a < n:
        yield a
        a, b = b, a+b
```

Without the static typing

```
def fib(n: int) -> Iterator[int]:
    a, b = 0, 1
    while a < n:
        yield a
        a, b = b, a+b
```

With the static typing

The recommended static type checker for Python is mypy

3.4 Automatic tests

3.5 Understating code

Once the project reaches certain level of complexity, measured e.g. in lines of code, and several programmers contribute to it work on it, understanding the code becomes quite important. The ability to comprehend is considered as an essential part of the software maintenance process. It is also one of the most critical and time-consuming task during software maintenance process.

[Source code comprehension analysis in software maintenance]

- Read the Code and Documentation (if exists)
- Execute White-Box And/or Black-Box
- Extract Block
- Write a Meaningful Comments
- Analyze the Internal Structure Dependencies
- Generate Program Graphs
- Refactoring

```
2Verbatim line.3
\ces
```