The New YAML-Based OAI 5GC Configuration

Demonstration and Design Decisions

Stefan Spettel

28.07.2023

Outline

- Motivation
- Design Decisions
- In-Depth Walkthrough of Configuration
- Demonstration
  - Simple Scenario
  - Slicing Scenario
  - UL CL Scenario
- Reducing Duplicated Code in the OAI 5GC
- Configuration Parsing (Code Examples)
- Q&A

OPEN AIR INTERFACE

Motivation

Old Configuration - Issues

- Three different approaches with old configuration:
  - Bare-Metal: Configure .conf file for your setup
  - Docker-Compose ENV variables
  - Docker-Compose .conf file mounting
- Most people use docker-compose to launch 5GC
- Environment variables have no relation to configuration, no structure
- Rely on “automagic” of entrypoint script

SMF Example

Docker-Compose

```
# Slice 0 (1, 0xFFFFFF)
- DNN_NI0=oai
- TYPE0=IPv4
- DNN_RANGE0=12.1.1.151 - 12.1.1.253
- NSSAI_SST0=1
- SESSION_AMBR_UL0=200Mbps
- SESSION_AMBR_DL0=400Mbps
```

smf.conf

```
DNN_LIST = (
   # PDU_SESSION_TYPE choice in {IPv4, IPv6, IPv4v6}
   # DNN IP ADDRESS RANGE format is for example: "12.2.1.2 - 12.2.1.128"
  {%- if env["DNN_NI0"] is defined %}
  {
    DNN_NI = "{{ env["DNN_NI0"] }}";
    PDU_SESSION_TYPE = "{{ env["TYPE0"] if "TYPE0" in env.keys() else 'IPv4' }}";
    IPV4_RANGE = "{{ env["DNN_RANGE0"] }}";
    IPV6_PREFIX = "2001:1:2::/64"
  }
```

Session Management Subscription Config

OPEN AIR INTERFACE

Issues with ENV Variables

- No clear context of environment variables
- Users need to know the .conf file in any case
- Difficult to integrate structured data
  - Lists with hacks, e.g., TYPE0, TYPE1, ...
- Docker-compose file confusingly long

Issues with old NF config - User

- Missing or ill-defined default values
- Confusing error messages on wrong configuration
- Default values handled via entrypoint automagic
- jinja-based syntax is not user-friendly
- A lot of logic handled in entrypoint
  - Should be done by NF natively

Issues with old NF config - Dev

- Each NF uses similar/same configuration
- Each NF has code to parse and read the same values - duplicated code
- This code itself is also duplicated and not always well structured
- Missing validation of user input
- libconfig is old-school

One Config To Rule Them All

- Use YAML as configuration language
- Use one file to configure all OAI NFs (SMF, AMF, NRF, PCF, ...) - for simple scenarios
- NFs have a reasonable default configuration
- Mounting a YAML configuration file in docker-compose is the way to go

Design Decisions

OPEN AIR INTERFACE

Do Not Repeat Yourself (DRY)

- Duplicated code is easy to write and hard to maintain
- “Small” changes require changing many lines of code
- Config change relevant to all NFs require code changes in all NFs (more than 8 repos)

DRY Example (SMF)

```
smf_cfg.lookupValue(SMF_CONFIG_STRING_DEFAULT_CSCF_IPV6_ADDRESS, astring);
if (inet_pton(AF_INET6, util::trim(astring).c_str(), buf_in6_addr) == 1) {
  memcpy( dest: &default_cscfv6,  src: buf_in6_addr,  n: sizeof(struct in6_addr));
} else {
  Logger::smf_app().error(
      "CONFIG : BAD ADDRESS in " SMF_CONFIG_STRING_DEFAULT_CSCF_IPV6_ADDRESS
      " %s",
      astring.c_str());
  throw(
      "CONFIG : BAD ADDRESS in " SMF_CONFIG_STRING_DEFAULT_CSCF_IPV6_ADDRESS
      " %s",
      astring.c_str());
}
```

OPEN AIR INTERFACE

DRY Example (SMF)

```
smf_cfg.lookupValue(SMF_CONFIG_STRING_DEFAULT_DNS_IPV6_ADDRESS, astring);
if (inet_pton(AF_INET6, util::trim(astring).c_str(), buf_in6_addr) == 1) {
  memcpy( dest: &default_dnsv6,  src: buf_in6_addr,  n: sizeof(struct in6_addr));
} else {
  Logger::smf_app().error(
      "CONFIG : BAD ADDRESS in " SMF_CONFIG_STRING_DEFAULT_DNS_IPV6_ADDRESS
      " %s",
      astring.c_str());
  throw(
      "CONFIG : BAD ADDRESS in " SMF_CONFIG_STRING_DEFAULT_DNS_IPV6_ADDRESS
      " %s",
      astring.c_str());
}
```

OPEN AIR INTERFACE

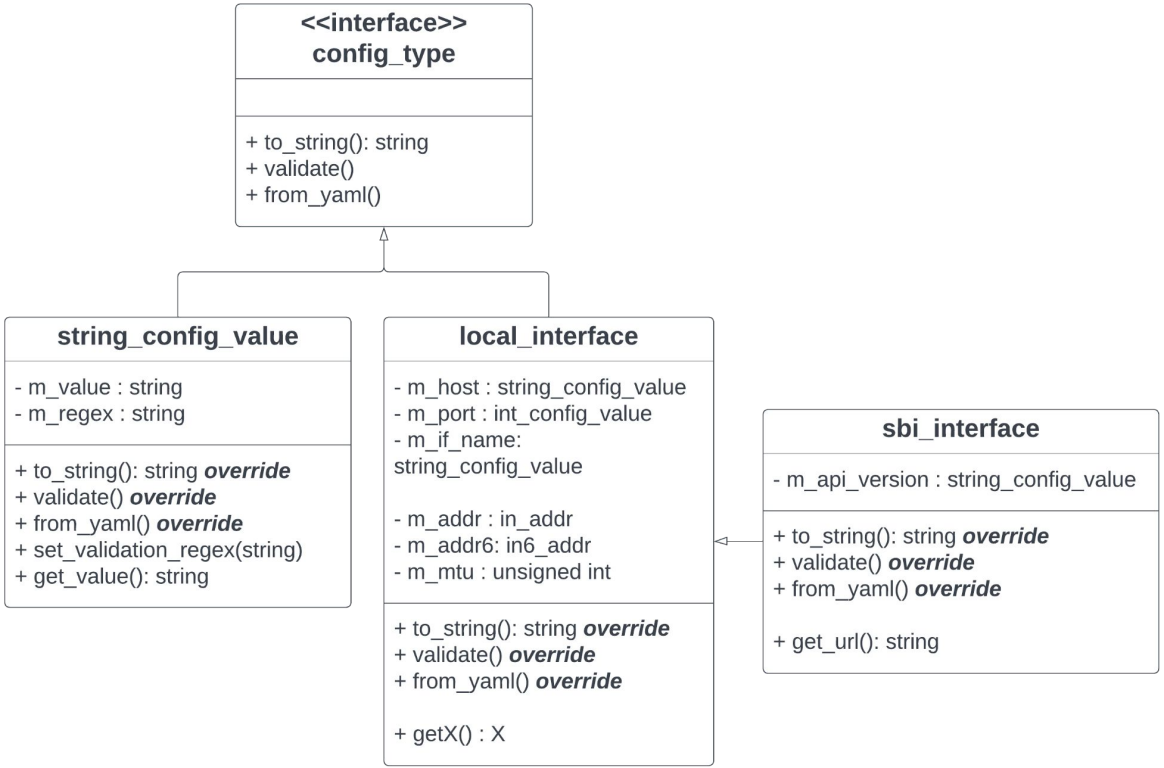
How to follow DRY

- Use a common source repository for all the code used by more than one NF
- Two approaches within NF code
  - Make helper functions (procedural style)
  - Use Object-oriented-programming (OOP) paradigm

Object-Oriented-Programming

- Create a base class “config_type” as interface
- Defines pure virtual functions
  - from_yaml -> To read YAML configuration
  - to_string -> To generate nice config output
  - validate -> To validate user input
- Each configuration type inherits from “config_type”
  - Must implement functions (dynamic polymorphism)

Config Class Diagram



DRY & OOP - Class Definition

```
class ue_dns : public config_type {
 private:
  string_config_value m_primary_dns_v4;
  string_config_value m_secondary_dns_v4;
  string_config_value m_primary_dns_v6;
  string_config_value m_secondary_dns_v6;

  // generated values
  in_addr m_primary_dns_v4_ip{};
  in_addr m_secondary_dns_v4_ip{};
  in6_addr m_primary_dns_v6_ip{};
  in6_addr m_secondary_dns_v6_ip{};
```

DRY & OOP - Reading IPs

```
void ue_dns::validate() {
  m_primary_dns_v4.validate();
  m_secondary_dns_v4.validate();
  m_primary_dns_v6.validate();
  m_secondary_dns_v6.validate();

  m_primary_dns_v4_ip = safe_convert_ip( ipv4_string: m_primary_dns_v4.get_value());
  if (m_secondary_dns_v4.is_set()) {
    m_secondary_dns_v4_ip = safe_convert_ip( ipv4_string: m_secondary_dns_v4.get_value());
  }
  if (m_primary_dns_v6.is_set()) {
    m_primary_dns_v6_ip = safe_convert_ip6( ipv6_string: m_primary_dns_v6.get_value());
  }
  if (m_secondary_dns_v6.is_set()) {
    m_secondary_dns_v6_ip = safe_convert_ip6( ipv6_string: m_secondary_dns_v6.get_value());
  }
```

OPEN AIR INTERFACE

Reusability

- The new configuration is generalized
- Can be used, adapted and enhanced by all NFs
- Configuration has some aspects of a library
  - High abstraction
  - More code reuse
- Use well-known “yaml-cpp” library to read YAML files

Design Decisions - Pros/Cons

Pros

- Reusable code
- Maintainable
- Easily extendable
- Modern C++
- YAML is user-friendly

Cons

- May not be easy for C-style procedural programmers
- “Over-engineered”

How to extend configuration?

- Start with any NF (e.g., SMF)
- Check local configuration in “smf_config.hpp”
- See “config.hpp” and “config_types.hpp” in the common-src repository
- Inherit from “config_type” and you are ready to go
- Use your own configuration in the NF-specific part and parse YAML there

In-Depth Walkthrough of Configuration

OPEN AIR INTERFACE

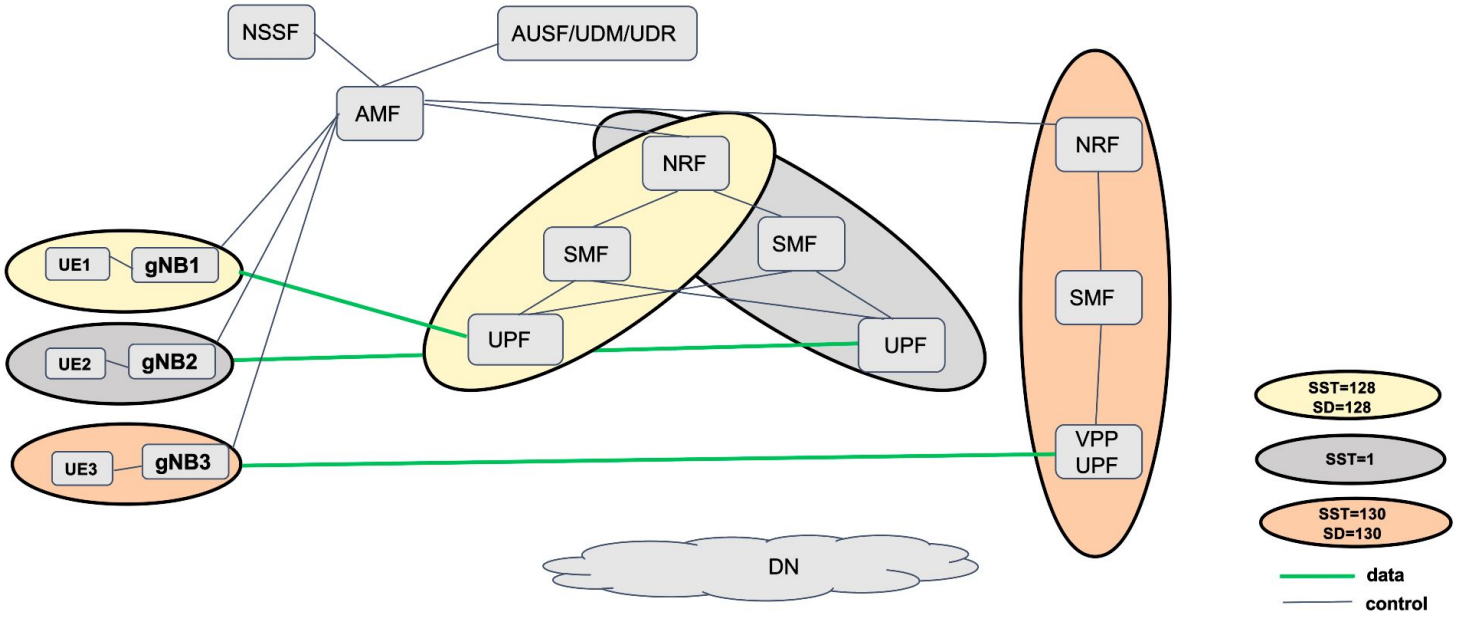
Discussed Configuration File

https://gitlab.eurecom.fr/oai/cn5g/oai-cn5g-fed/-/blob/develop/docker-compose/conf/basic_nrf_config.yaml

Demonstration Basic Scenario

- Scenario
  - NFs are registered to NRF
  - SMF uses local subscription data
  - No PCF, no NSSF
  - Registration + PDU Session Establishment using gnbsim

Demonstration Slicing Scenario



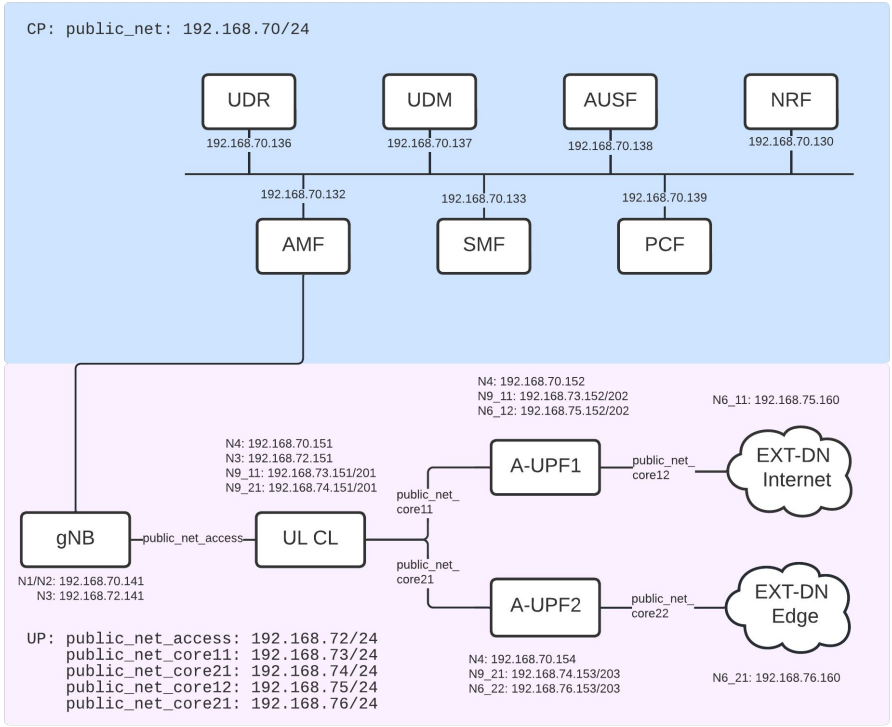
Discussed Configuration Files

- https://gitlab.eurecom.fr/oai/cn5g/oai-cn5g-fed/-/blob/develop/docker-compose/conf/slicing_base_config.yaml
- https://gitlab.eurecom.fr/oai/cn5g/oai-cn5g-fed/-/blob/develop/docker-compose/conf/slicing_slice1_config.yaml
- https://gitlab.eurecom.fr/oai/cn5g/oai-cn5g-fed/-/blob/develop/docker-compose/conf/slicing_slice2_config.yaml
- https://gitlab.eurecom.fr/oai/cn5g/oai-cn5g-fed/-/blob/develop/docker-compose/conf/slicing_slice3_config.yaml

Demonstration UL CL Scenario

- Scenario
  - NFs are registered to NRF
  - SMF uses local subscription data
  - 3 VPP-UPFs
  - PCF with traffic routing policies
  - Registration + PDU Session Establishment using gnbsim

Architecture UL CL Scenario



Discussed Configuration File

https://gitlab.eurecom.fr/oai/cn5g/oai-cn5g-fed/-/blob/develop/docker-compose/conf/ulcl_config.yaml

Reducing Duplicated Code in the OAI

OPEN AIR INTERFACE

Current Situation 1/2

- Many NFs use the same code for certain functionalities
  - Protocols (PFCP, NAS, NGAP)
  - HTTP (client and server code)
  - Utils
  - Common models, definitions, etc

Current Situation 2/2

- Code began to diverge, hard to maintain same code in several repositories
- Need to move code to new repository: common-src
- Use common-src as git submodule
- In the process of migrating copied code
- Done for config and logger

Config Example

	New Config LoC	Old Config LoC
common-src	1696	0
SMF	1544*	1360
NRF	20	204
PCF	152	373
NSSF	288	441
Sum	**3700**	**2378**

OPEN AIR INTERFACE

Config Example Discussion

- More LoC, but measured only 4 NFs
- More features:
  - Validation for all configuration types
  - Better string representation of configuration
- Some LoC overhead through OOP
- No code duplications
- Potential for removing more LoC

Next Steps

- Refactor HTTP client and use the same HTTP client for all NFs
- Refactor NRF registration/lookup procedures and use the same for all NFs
- Move PFCP protocol layer to common-src (used by SMF and UPF)

Configuration Parsing Code Examples

OPEN AIR INTERFACE

Feedback

- New YAML configuration is still under development
- Please use it and give us feedback
  - Right Now
  - Mailing List
  - GitLab Issues
  - Slack: openairinterface.slack.com

Q & A

OPEN AIR INTERFACE

Thank you!

Contact: stefan.spettel@phine.tech

OPEN AIR INTERFACE