CLOSURE Toolchain User Manual for C Language

Release version 3.0

Peraton Labs

April 29, 2024

Contents

1 CLOSURE Toolchain Overview

1.1 What is CLOSURE?

DARPA's Guaranteed Architecture for Physical Systems (GAPS) is a research program that addresses software and hardware for compartmentalized applications where multiple parties, each with strong physical isolation of their computational environment, have specific constraints on the sharing of data (possibly including redaction requirements) with other parties, and any data exchange between the parties is mediated through a guard that enforces the security requirements.

Peraton Labs' Cross-domain Language extensions for Optimal SecUre Refactoring and Execution (CLOSURE) project is building a toolchain to support the development, refactoring, and correct-by-construction partitioning of applications and configuration of the guards. Using the CLOSURE approach and toolchain, developers will express security intent through annotations applied to the program, which drive the program analysis, partitioning, and code auto-generation required by a GAPS application.

Problem: The machinery required to verifiably and securely establish communication between cross-domain systems (CDS) without jeopardizing data spillage is too complex to implement for many software platforms where such communication would otherwise be desired. To regulate data exchanges between domains, network architects rely on several risk mitigation strategies including human fusion of data, data-diodes, and hypervisors which are insufficient for future commercial and government needs as they are high overhead, customized to specific setups, prone to misconfiguration, and vulnerable

to software/hardware security flaws. To streamline the design, development, and deployment of provably secure CDSs, new hardware and software co-design tools are needed to more effectively build cross-domain support directly into applications and associated hardware early in the development lifecycle.

Solution: Peraton Labs is developing CLOSURE (Cross-domain Language-extensions for Optimal SecUre Refactoring and Execution) to address the challenges associated with building cross-domain applications in software. CLOSURE extends existing programming languages by enabling developers the ability to express security intent through overlay annotations and security policies such that an application can be compiled to separate binaries for concurrent execution on physically isolated platforms.

The CLOSURE compiler toolchain interprets annotation directives and performs program analysis of the annotated program and produces a correct-by-construction partition if feasible. CLOSURE automatically generates and inserts serialization, marshalling, and remote-procedure call code for cross-domain interactions between the program partitions.

1.2 Architecture

CLOSURE has a modular and layered architecture as shown in the figure below. The architecture supports multiple source languages and employs a common LLVM IR format (the thin "waist" of the architecture), where key CLOSURE partitioning and optimization is performed. The architecture simplifies adding source languages, and allows reuse of well-engineered front-ends, linkers, optimizers, and back-ends. Binaries are generated for multiple target hardware platforms.

The developer uses the CLOSURE Visual Interface and associated tools to annotate source code with CLOSURE Language Extensions (CLE). A standard linker and general-purpose program optimizer is invoked to link the GAPS-aware application libraries, the CLOSURE libraries for concurrency and hardware abstraction, and the rewritten legacy libraries into a set of platform specific executables. Shown on the left of the figure is the Global Security Policy Specification (GSPS), which localizes mission security constraints and global security policy, including existing security levels, available hardware systems, allowable cross-level communication, and standard pre-formed cross-domain components including encryption, one-way channels, and downgrade functionality. The GSPS abstracts global security constraints, and allows the user to easily make per-mission or per environmental changes.

The key sub-modules of the toolchain include:

- **CVI**: CLOSURE Visual Interface. The editor built on top of VSCode [1]. Provides the IDE and co-design tools for cross-domain software development.

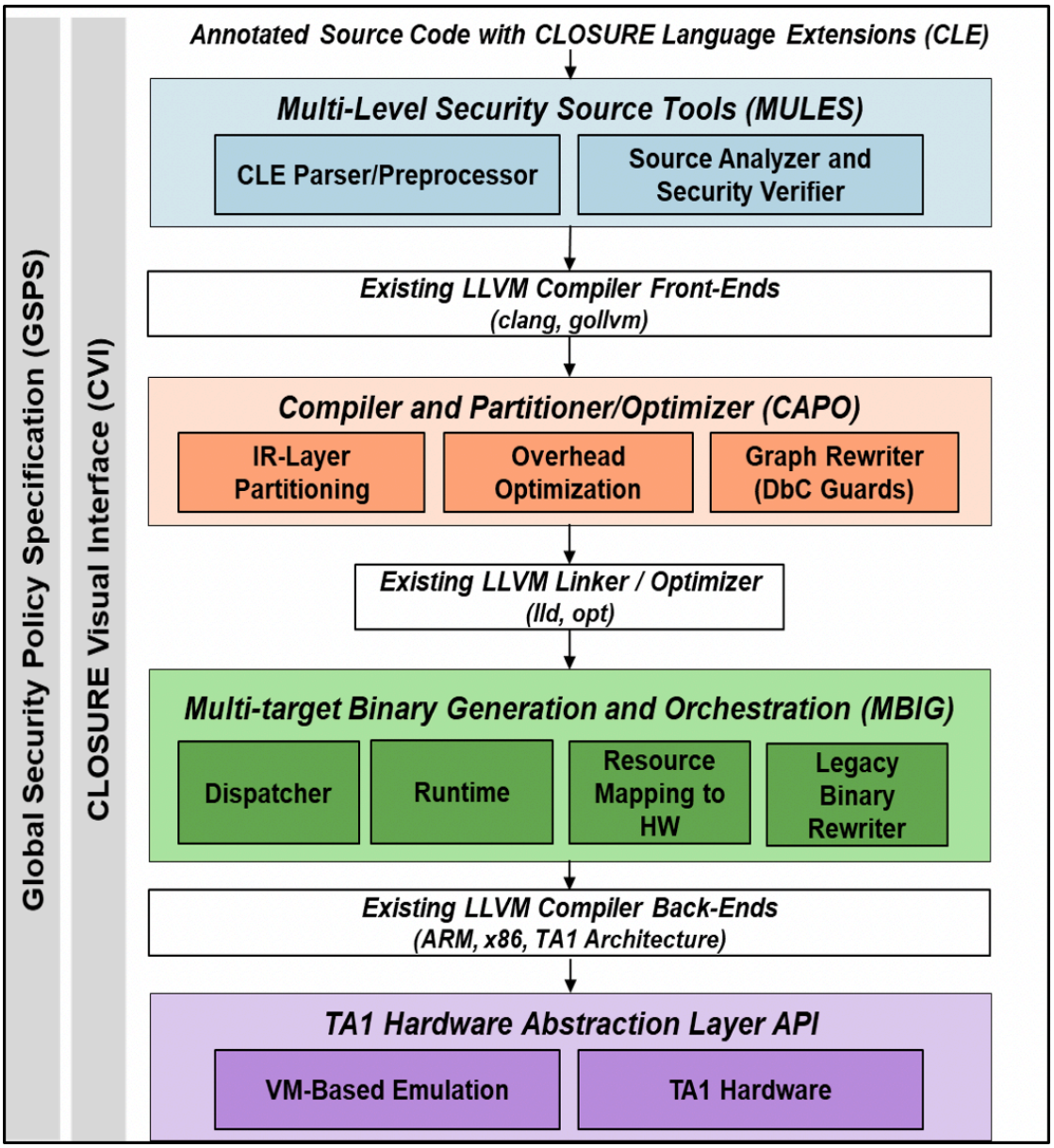


Figure 1: CLOSURE architecture

- **MULES**: Multi-Level Security Source Tools. Source tools containing the CLOSURE Language extensions and CLE schema. Includes a preprocessor which converts CLE annotations to LLVM attributes for clang processing. Code generation from models also resides here.
- **CAPO**: Conflict Analyzer Partition Optimizer. CAPO includes the constraint-based conflict analysis tools to determine if a partitioning is feasible. Additional tools in CAPO auto-generate the additional logic needed to make a program cross-domain enabled (i.e., data type marshalling/serialization, RPCs for cross-domain data exchange, interfacing to the device drivers of cross-domain guards, DFDL [2] and rule generation, among others). CAPO also includes a post-partitioning verifier which checks that the partitioned program including auto-generated code is functionally equivalent to complies with developer security annotations.
- **MBIG**: Multi-Target Binary Generation. Supports compilation to x86 and ARM targets as well as packaging of applications.
- **HAL**: Hardware-Abstraction-layer. Abstracts hardware APIs of different cross-domain hardware devices giving appliacations a single cross-domain communication (XDCOMMS) API. CLOSURE has two HAL implementations:
  - *HAL-Daemon*: Runs HAL as a separate daemon that communicates with applications using 0MQ-based middleware.
  - *HAL-Library*: Linked to applications as a library using the raw XDCOMMS send, receive and control API calls.
- **EMU**: Emulator. Enables test and evaluation of cross-domain applications utilizing QEMU.

1.3 Workflow

The CLOSURE workflow for building cross-domain applications shown in the figure below can be viewed at a high-level as three toolchain stages: 1) Annotation-driven development for for correct-by-construction partitions with interactive feedback for guided refactoring, 2) Automated generation of cross-domain artifacts, compilation, and verification of partitioned program, and 3) seamless support for heterogeneous GAPS hardware architectures and emulation for pre-deployment testing.

In the first stage, the developer either writes a new application or imports existing source which must be tooled for cross-domain operation. The developer must have knowledge of the intended cross-domain policy. CLOSURE provides means to express this policy in code, but it is the requirements analyst/developer who determines the policy in advance. The developer then uses CLE to annotate the program as such. The CLOSURE pre-processor, PDG model, and constraint analysis determine if the partitioning of the annotated program is feasible. If not, feedback and diagnostics are provided back to the developer for guided refactoring towards making the program compliant. Once the

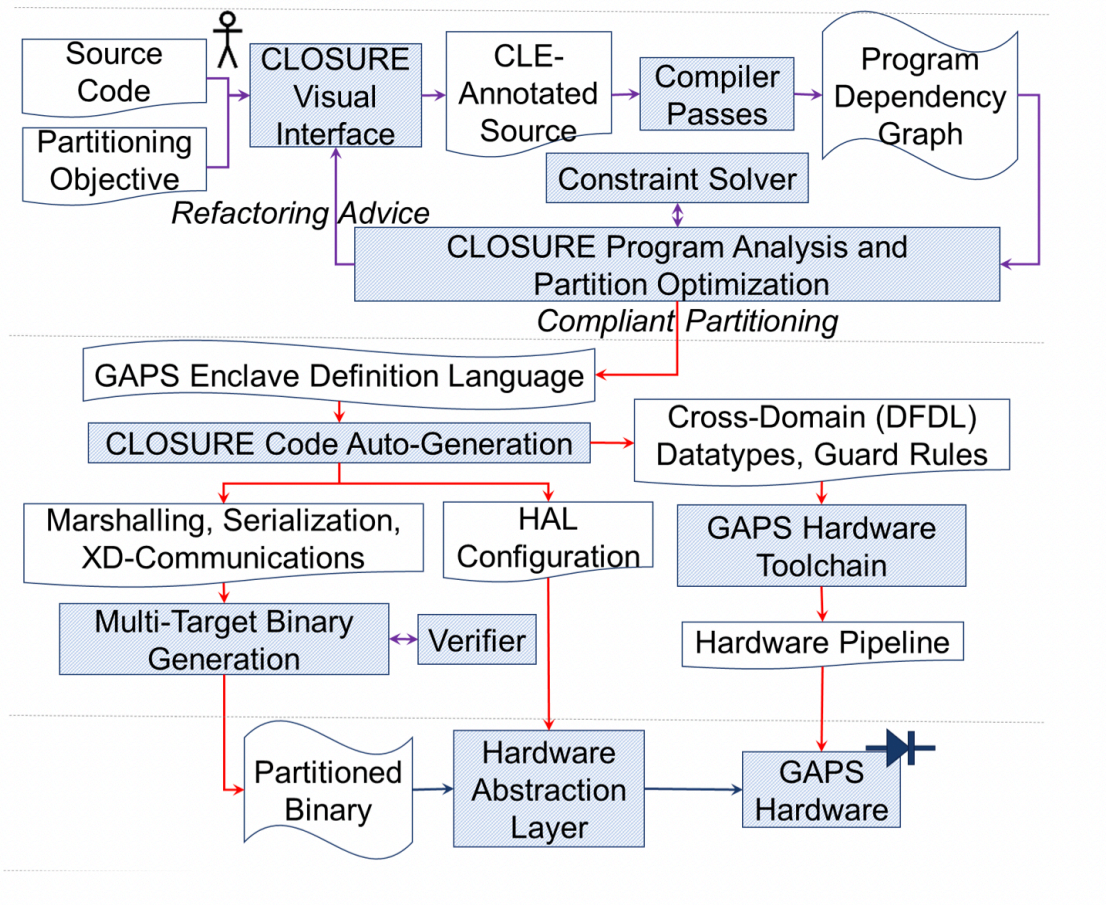


Figure 2: CLOSURE Workflow

program is deemed compliant (via the conflict analyzer), CLOSURE proceeds with automated tooling in which CAPO and associated tools divide the code, generate code for cross-domain remote procedure calls (RPCs), describe the formats of the cross-domain data types via DFDL and codec/serialization code, and generate all required configurations for interfacing to the GAPS hardware via the Hardware Abstraction Layer (HAL). In the final stage, the partitioned source trees are compiled for the target host (e.g., x86 or ARM64) and prepared for deployment.

1.3.1 C generation from Message-flow models

In addition to annotated C programs, CLOSURE can also be used to support partitioning of models of message-flow based applications, that is, applications that have already been partitioned into components, but use a messaging service such as ActiveMQ to exchange messages via publish/subscribe blackboard. The purpose of CLOSURE model-driven design are the following:

- Abstraction is moved up from partitioning control/data flow of a single program to partitioning message flows of a distributed application
- Shifts problem from compilers to design tools
- Considers complex application data structures (nested JSON) and generates flat fixed-formats suitable for simple-but-fast hardware guards
- Increases formatting and marshalling complexity testing flexibility of current GAPS hardware capability

Model driven approach described later in this document.

1.4 Document Roadmap

In the rest of this document, we first present a quick start guide followed by a detailed usage of the toolchain components. For each component, we describe what it does, provide some insight into how it works, discuss inputs and outputs and provide invocation syntax for usage. We conclude with a discussion of limitations of the current toolchain and a roadmap for future work. We provide samples of significant input and output files in the appendices, and provide a list of bibliographic references at the end.

2 Installation and Quick Start

2.1 Prerequisites

To run the CLOSURE C toolchain [3], you will need the following prerequisites:

1. Ubuntu Linux 20.04 Desktop

2. Docker verion 20.10
3. Visual Studio Code with Remote Containers Extension

The dockerfile [4] we publish provisions all other dependencies needed for the CLOSURE toolchain, such as clang, python, and llvm.

2.2 Installation For CLOSURE Users

The fastest way to get started with the CLOSURE toolchain is to pull the published docker image from dockerhub:

```
docker pull gapsclosure/closuredev:master
```

Then, to get a shell with closure installed, enter the following:

```
docker run -it gapsclosure/closuredev:master
```

2.3 Running example applications

To run the example applications you will need to clone the CLOSURE build repository:

```
git clone https://github.com/gaps-closure/build.git --recurse-submodules
```

This will create a new directory called build within your current directory. Under `apps/examples` there are three examples demonstrating basic usage of the CLOSURE toolchain. For instance, you can open `example1` with vscode as follows:

```
code build/apps/examples/example1
```

VSCode should prompt you with reopening within a container. If not, hit `Ctrl+Shift+P` and type `Reopen in Container` and click the corresponding menu item in the dropdown.

Then you can proceed with the steps in the CLOSURE workflow. If you hit `Ctrl+Shift+B`, you should get a tasks dropdown with a list of build steps which should look like the following:

The workflow begins by annotating the original source, which can be found under `plain`. Hitting `1 Annotate` under the dropdown will copy from `plain` into `annotated`. You can now annotate the program with CLE labels and their definitions. If you are not comfortable yet with CLE annotations, or are stuck, there is a solution provided in `.solution/refactored`

After annotating, you can start `2 Conflicts` under the tasks dropdown which will start the conflict analyzer. If the conflict analyzer finds no issues, it will produce a `topology.json` file. Otherwise, it will print out diagnostics.

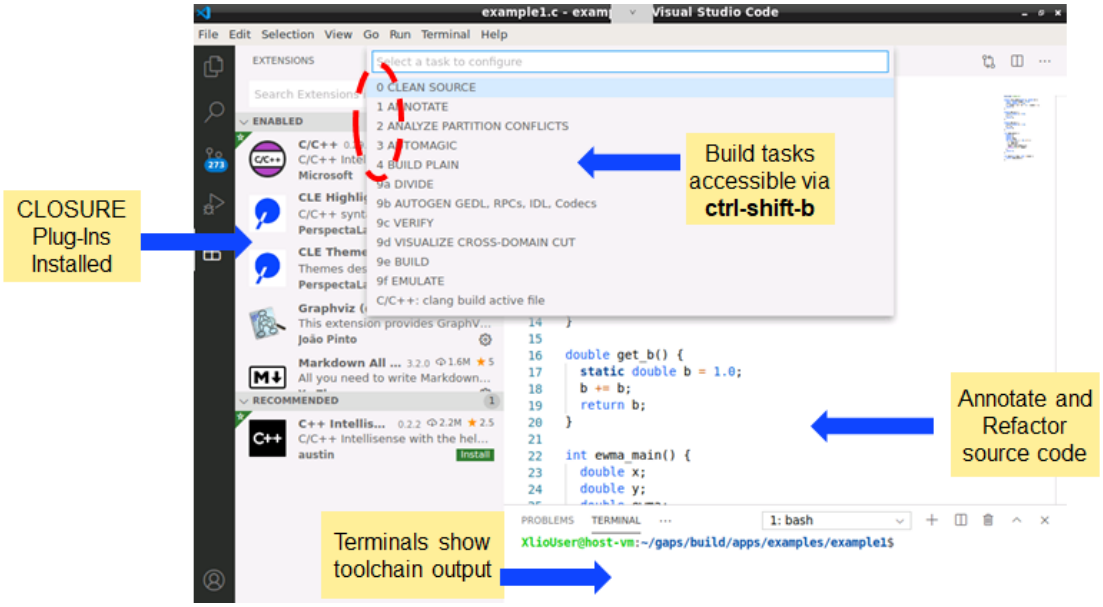


Figure 3: CLOSURE workflow in VSCode

Then, you can start the `3 Automagic` task, which will partition and build the application, as well as start running the application within the emulator.

2.4 Notes For CLOSURE Toolchain Developers

2.4.1 Building the Docker container

For developers of the CLOSURE toolchain, it may be useful to understand how to build the closure container:

If within the build directory, all that needs to be done to build CLOSURE dockerfile is:

```
docker build -f Dockerfile.dev -t gapsclosure/closuredev:master .
```

2.4.2 Switching out environment variables in projects

When testing a new feature of some part of the closure toolchain within a project, it's faster to change environment variables to point to a local version of a given tool.

For example, a CLOSURE toolchain developer may want to test out a new feature of the `rpc_generator.py` script. In `closure_env.sh` the `RPC_GENERATOR` variable can be switched out to point to their own version of the script. Invocation of the script through

the build process will be identical as before. Here's an excerpt of a `closure_env.sh` where `rpc_generator.py` is switched out for a development version.

```
#!/bin/bash

# ...

CLOSURE_TOOLS=/opt/closure
source /opt/closure/etc/closureenv
export CLOSURE_BINS=${CLOSURE_TOOLS}/bin
export CLOSURE_INCLUDES=${CLOSURE_TOOLS}/include
export CLOSURE_LIBS=${CLOSURE_TOOLS}/lib

# ...

# export RPCGENERATOR=rpc_generator
export RPC_GENERATOR=path/to/my/rpc_generator
# ...
```

2.4.3 Dockerfile notes

A list of the dependencies can be found in the Dockerfile in `build`. Most of these dependencies are given by their corresponding apt/python package, and others are installed manually, such as MiniZinc, Haskell and CORE.

The Dockerfile uses a `COPY` command to copy over the `build` directory into the image and builds/installs CLOSURE from within the container.

2.4.4 Installing locally

A build can be made locally if the dependencies specified in the dockerfile are installed on a local machine.

To install locally, enter the following into the shell within the `build` directory:

```
./install.sh -o <install_dir, e.g. /opt/closure>
```

Many of the submodules within `build` repository can be installed similarly, only building and installing the relevant packages for each repository.

Note that you will need to include `<install_prefix>/etc/closureenv` to your path, e.g. in your `.bashrc` in order to have command line access to the closure tools.

2.4.5 Emulator

The emulator (EMU) uses QEMU instances to represent enclave gateways, the nodes designated for cross-domain transactions via a character device. This allows us to model multi-domain, multi-ISA environments on which the partitioned software will execute.

As a prerequisite to executing the emulator, it is necessary to build clean VM instances (referred to as the "golden images") from which EMU will generate runtime snapshots per experiment. The snapshots allow EMU to quickly spawn clean VM instances for each experiment as well as support multiple experiments in parallel without interfering among users.

Usually these QEMU images are mounted in the docker container, so that they can be used across containers without rebuilds. By default it mounted from and to `/IMAGES` but can be changed in each project's `.devcontainer/devcontainer.json`

VM images can be automatically built using `qemu-build-vm-images.sh` which is mounted in `/opt/closure/emu` by default in the `.devcontainer/devcontainer.json`. The script fetches the kernel, builds and minimally configures the VM disk images, and saves a golden copy of the kernels and images. This script is run by default if needed in the supported applications during the VSCode task 9E.

3 Detailed Usage and Reference Manual

3.1 Annotations

The CLOSURE toolchain relies on source level annotations to specify the cross domain constraints. Developers annotate programs using CLOSURE Language Extensions (CLE) to specify cross-domain security constraints. Each CLE annotation definition associates a *CLE label* (a symbol) with a *CLE JSON* which provides detailed specification of cross-domain data sharing and function invocation constraints.

These source level annotations determine the following:

1. The assignments of functions and global variables to enclaves
2. The confidentiality of data between enclaves
3. Which functions can be called cross domain
4. Guard rules which transform data as it crosses domains

Typically, these annotations are only applied to a subset of the program and a separate tool called the *conflict analyzer* is able infer the CLE labels of the rest of the program elements given this subset.

3.1.1 A Simple Annotation

We can associate a *label* with some JSON which describes constraints on the cross domain properties of program elements, forming a CLE definition. This is done using a custom `#pragma cle` within the C source:

```
#pragma cle def FOO { "level": "orange" }
```

The example above provides a minimal cle definition which can constrain a global variable. In order to apply such a definition to some global variable you can use `#pragma cle begin/end`:

```
#pragma cle begin FOO
int bar = 0;
#pragma cle end FOO
```

or more simply

```
#pragma cle FOO
int bar = 0;
```

Now look at the JSON in the label definition more closely:

```
#pragma cle def FOO { "level": "orange" }
```

The JSON specifies a `"level"` field set to `"orange"`. The level is like security level, but there is no requirement for an ordering among the levels. A single level may correspond to many enclaves, but in most cases they will be in a bijection with the enclaves. The level names can be any string.

Enclaves are isolated compartments with computation and memory reources. Each enclave operates at a specified level. Enclaves at the same level may be connected by a network, but enclaves at different levels must be connected through a cross-domain guard (also known as SDH or TA-1 hardware within the GAPS program).

When applying the `FOO` label to `int bar`, we effectively pin `bar` to a single level `"orange"`.

3.1.2 An Annotation with Cross-Domain Flow (cdf) Definition

The next example shows another label definition, this time with a new field called `cdf`, standing for cross-domain flow.

```
#pragma cle def ORANGE_SHAREABLE {"level":"orange",\
  "cdf": [\
    {"remotelevel":"purple", \
     "direction": "egress", \
     "guarddirective": { "operation": "allow"}}\
  ] }
```

Here, the `"remotelevel"` field specifies that the program element the label is applied to can be shared with an enclave so long as its level is `"purple"`. The `"guarddirective": { "operation": "allow"}}` defines how data gets transformed as it goes across enclaves. In this case, `{ "operation": "allow" }` simply allows the data to pass uninhibited. The `"direction"` field is currently not used and is ignored by the CLOSURE toolchain (may be removed in future release).

The `cdf` is an array, and data can be released into more than one enclave. Each object within the `cdf` array is called a `cdf`.

3.1.3 Function Annotations

Broadly there are two types of annotations: 1) data annotations and 2) function annotations. The previous examples showcased data annotations, but function annotations allow for functions to be called cross domain, and data to be passed between them.

Function annotations look similar to data annotations, but contain three extra fields within each cdf, `argtaints`, `codtaints` and `rettaints`.

```
#pragma cle def XDLINKAGE_GET_A {"level":"orange",\
  "cdf": [\
    {"remotelevel":"purple", \
     "direction": "bidirectional", \
     "guarddirective": { "operation": "allow"}, \
     "argtaints": [], \
     "codtaints": ["ORANGE"], \
     "rettaints": ["TAG_RESPONSE_GET_A"], \
     "idempotent": true, \
     "num_tries": 30, \
     "timeout": 1000 \
    }, \
    {"remotelevel":"orange", \
     "direction": "bidirectional", \
     "guarddirective": { "operation": "allow"}, \
     "argtaints": [], \
     "codtaints": ["ORANGE"], \
     "rettaints": ["TAG_RESPONSE_GET_A"] \
```

```
    } \
  ] }
```

In a function annotation, the `cdf` field specifies remote levels permitted to call the annotated function. Function annotations are also different from data annotations as they contain
`taints` fields.

A taint refers to a label or an assigned label by the conflict analyzer for a given data element. There are three different taint types to describe the inputs, body, and outputs of a function: `argtaints`, `codtaints` and `rettaints` respectively. Each portion of the function may only touch data tainted with the label(s) specified by the function annotation: `rettaints` constrains which labels the return value of a function may be assigned. Similarly, `argtaints` constrains the assigned labels for each argument. This field is a 2D-array, mapping each argument of the function to a list of assignable labels. `codtaints` includes any other additional labels that may appear in the body. Function annotations can coerce between labels of the same level, so it is expected that these functions are to be audited by the developer. Often, the developer will add a cdf where the remotelevel is the same as the level of the annotation, just to perform some coercion.

Optional fields in function annotations include

- idempotent: indicates function can be called repeatedly (e.g., when messages are lost in the network, you RPC logic can reissue the request)
- num_tries: Upon failure, how many attempts to call a function cross domain before giving up
- timeout: controls the timeout for the cross domain read function (see `xdc_recv` function).

3.1.4 Label Coercion

Only an annotated function can accept data of one or more label and produce data with other labels as allowed by the annotation constraints on the function's arguments, return, and body. We call this label or taint coercion. See label coercion for detailed discussion. Label coercion can happen within a single level when a function annotation is given a cdf with a remotelevel the same as its level.

3.1.5 TAGs

In the `XDLINKAGE_GET_A` label, there is `TAG_RESPONSE_GET_A` label. This is a reserved label name which does not require a user-provided definition. The definitions for these `TAG_` labels are generated automatically by the toolchain; for every cross domain call there are two `TAG_` labels generated for receiving and transmitting data, called `TAG_REQUEST_` and `TAG_RESPONSE_`. Each generated tag label has a suffix which is

the name of the function it is being applied to in capital letters. The label indicates associated data is the result if incoming or outgoing data specific to the RPC logic. This supports verification of data types involved in the cross-domain cut and that only intended data crosses the associated RPC.

3.1.6 Example cross domain function

Consider `double bar(double x, double y);` which resides in level `purple` and `void foo()` which resides in level `orange`, and the intent is that `foo` will call `bar` cross domain. A full specification of this interaction using CLE annotations is presented as follows:

```
#pragma cle def ORANGE {"level":"orange",\
  "cdf": [\
    {"remotelevel":"purple", \
     "direction": "egress", \
     "guarddirective": { "operation": "allow"}}\
  ] }

#pragma cle def PURPLE { "level": "purple" }

#pragma cle def FOO {"level":"orange",\
  "cdf": [\
    {"remotelevel":"orange", \
     "direction": "bidirectional", \
     "guarddirective": { "operation": "allow"}, \
     "argtaints": [], \
     "codtaints": ["ORANGE", "TAG_RESPONSE_BAR", "TAG_REQUEST_BAR"], \
     "rettaints": [] \
    } \
  ] }

#pragma cle def BAR {"level":"purple",\
  "cdf": [\
    {"remotelevel":"purple", \
     "direction": "bidirectional", \
     "guarddirective": { "operation": "allow"}, \
     "argtaints": [["TAG_REQUEST_BAR"], ["TAG_REQUEST_BAR"]], \
     "codtaints": ["PURPLE"], \
     "rettaints": ["TAG_RESPONSE_BAR"] \
    }, \
    {"remotelevel":"orange", \
     "direction": "bidirectional", \
     "guarddirective": { "operation": "allow"}, \
```

```
     "argtaints": [["TAG_REQUEST_BAR"], ["TAG_REQUEST_BAR"]], \
     "codtaints": ["PURPLE"], \
     "rettaints": ["TAG_RESPONSE_BAR"] \
    } \
  ] }

#pragma cle begin FOO
void foo() {
#pragma cle end FOO
  double result = bar(0, 1);
  // ...
}


#pragma cle begin BAR
double bar(double x, double y) {
#pragma cle end BAR
  return (x + y) * (x * y);
}
```

In this example there are two label coercions. One from the caller side from `ORANGE` to the tags of `foo` and the other on the callee side from the tags of `bar` to `PURPLE` and back. These coercions are needed because the autogenerated code only works on data cross domain with the tag labels. Future releases may permit users to specify these tag labels, so that less coercion is needed.

3.2 Conflict Analyzer Using MiniZinc Constraint Solver

The role of the conflict analyzer is to evaluate a user annotated program and decide if the annotated program respects the allowable information flows specified in the annotations. As input, the conflict analyzer requires the user annotated C source code. Based on this, if it is properly annotated and a partition is possible it will provide an assignment for every global variable and function to an enclave (topology.json). If the conflict analyzer detects a conflict, it produces a report guiding users to problematic program points that may need to be refactored or additional annotations applied.

The conflict analyzer uses a constraint solver called MiniZinc [5] to perform program analysis and determine a correct-by-construction partition that satisfies the constraints specified by the developer using CLE annotations. MiniZinc provides a high level language abstraction to express constraint solving problems in an intuitive manner. MiniZinc compiles a MiniZinc language specification of a problem for lower level solver such as Gecode. We use an Integer Logic Program (ILP) formulation with MiniZinc. MiniZinc

also includes a tool that computes the minimum unsatisfiable subset (MUS) of constraints if a problem instance is unsatisfiable. The output of this tool can be used to provide diagnostic feedback to the user to help refactor the program.

In addition to outputing a `topology.json` providing the assignments of global and function variables, the conflict analyzer can also provide a more detailed version of this file `artifact.json`. This file provides the label, level, and enclave assignments to every program element.

Downstream tools in the CLOSURE toolchain will use the output of the solver to physically partition the code, and after further analysis (for example, to determine whether each parameter is an input, output, or both, and the size of the parameter), the downstream tools will autogenerate code for the marshaling and serialization of input and output/return data for the cross-domain call, as well as code for invocation and handling of cross-domain remote-procedure calls that wrap the function invocations in the cross-domain cut.

3.2.1 Introduction to the Conflict Analyzer

3.2.2 Usage

The usage of the conflict analyzer is as follows:

```
usage: conflict_analyzer [-h] [--temp-dir TEMP_DIR] [--clang-args
 ↪  CLANG_ARGS] [--schema [SCHEMA]] --pdg-lib PDG_LIB [--source-path
 ↪  SOURCE_PATH] [--constraint-files [CONSTRAINT_FILES [CONSTRAINT_FILES
 ↪  ...]]] [--output OUTPUT]
                          [--artifact ARTIFACT] [--conflicts CONFLICTS]
 ↪  [--output-json] [--log-level {DEBUG,INFO,ERROR}]
                          sources [sources ...]

positional arguments:
  sources               .c or .h to run through conflict analyzer

optional arguments:
  -h, --help            show this help message and exit
  --temp-dir TEMP_DIR   Temporary directory.
  --clang-args CLANG_ARGS
                        Arguments to pass to clang (paths should be
 ↪  absolute)
  --schema [SCHEMA]     CLE schema
  --pdg-lib PDG_LIB     Path to pdg lib
  --source-path SOURCE_PATH
```

```
                        Source path for output topology. Defaults to
↪  current directory
 --constraint-files [CONSTRAINT_FILES [CONSTRAINT_FILES ...]]
                        Path to constraint files
 --output OUTPUT        Output path for topology json
 --artifact ARTIFACT    artifact json path
 --conflicts CONFLICTS
                        conflicts json path
 --output-json          whether to output json
 --log-level {DEBUG,INFO,ERROR}, -v {DEBUG,INFO,ERROR}
```

As inputs the conflict analyzer takes in C source and header files and outputs a topology.json and possibly an `artifact.json`.

The pdglib can be found at `/opt/closure/lib/libpdg.so`. The source path controls the `source_path` field in the topology. The `output-json` option will output diagnostics and results in a JSON form readable by CVI.

Note: the `--clang-args` is a comma separated list of arguments to pass to clang, and any paths given (e.g., includes) must be absolute.

3.2.3 The CLOSURE `preprocessor`

The CLOSURE preprocessor is a source transformer that will take a given source or header file with CLE annotations, and produce

1. A modified source or header file with LLVM `__attribute__` annotations
2. A cle-json file which contains a mapping from each label to its corresponding definition in JSON form. See appendix for an example cle-json file.

The output C code of the preprocessor will go to a minimally modified LLVM clang that will support the CLOSURE-specific LLVM `__attribute__` annotations and pass them down to the LLVM IR level.

In addition, the preprocessor performs several well-formedness checks on the cle labels and definitions, using a json schema.

For example:

With an initial source C file containing the following:

```
int *secretvar = 0;
```

The developer annotates the C file as follows:

```
#pragma cle def ORANGE { /* CLE-JSON, possibly \-escaped multi-line with
 ↪  whole bunch of constraints*/ }
#pragma cle ORANGE
int *secretvar = 0;
```

After running the preprocessor, we get a C file with pragmas removed but with `__attribute__` inserted (in all applicable places), e.g.,:

```
#pragma clang attribute push (__attribute__((annotate("ORANGE"))),
 ↪  apply_to =
 ↪  any(function,type_alias,record,enum,variable(unless(is_parameter)),field))
int *secretvar = 0;
#pragma clang attribute pop
```

3.2.4 `opt` pass for the Program Dependence Graph (PDG)

The Program Dependence Graph (PDG)[6] [7] [8] is an abstraction over a C/C++ program which specifies its control and data dependencies in a graph data structure. The PDG is computed by adding an analysis pass to clang. The resulting graph is then parsed into a format that MiniZinc can reason about.

During the invocation of the conflict analyzer, a subprocess is spawned in python to retrieve this `minizinc` (problem instance in MiniZinc format) representation of the PDG.

The relevant PDG node types for conflict analysis are Inst (instructions), VarNode (global, static, or module static variables), FunctionEntry (function entry points), Param (nodes denoting actual and formal parameters for input and output), and Annotation (LLVM IR annotation nodes, a subset of which will be CLE labels). The PDG edge types include ControlDep (control dependency edges), DataDep (data dependency edges), Parameter (parameter edges relating to input params, output params, or parameter field edges to encode parameter trees), and Annot (edges that connect a non-annotation PDG node to an LLVM annotation node). Each of these node and edge types are further divided into subtypes.

More documentation about the specific nodes and edges in the PDG can be found here.

3.2.5 input generation and constraint solving using Minizinc

From the cle json outputted by the preprocessor, the conflict analyzer generates a `minizinc` representation of the cle json, which describes the annotations and enclaves for a given program. The `minizinc` produced from the cle json is designed to be used with the PDG `minizinc` representation, and together provide a full representation of the

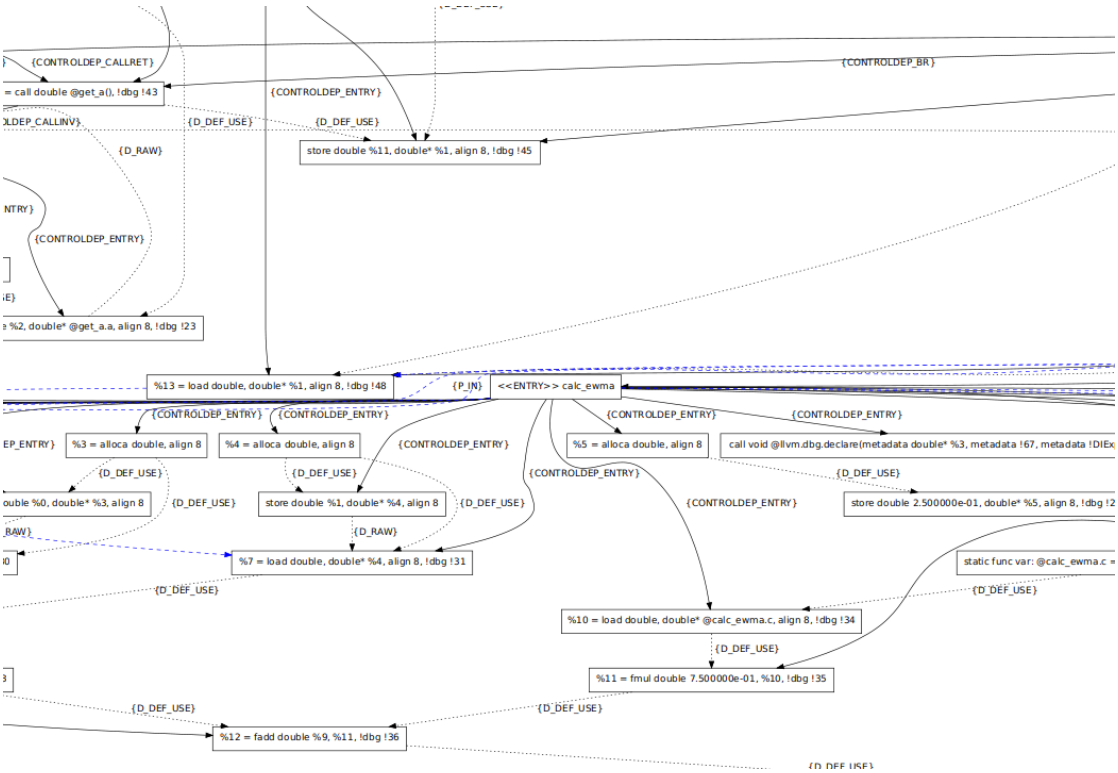


Figure 4: Visualization of the PDG for example 1

program and the constraints provided by the annotations in `minizinc` form. A sample minizinc representation for example 1 can be found in the appendix.

Together, this can be fed, along with a set of constraints described in section constraints, to `minizinc` producing an assignment of every node in the PDG to a label, or a minimally unsatisfied set of constraints. Understanding these constraints is crucial to understanding why a certain program cannot pass the conflict analyzer.

From these assignments, the `topology.json` and `artifact.json` can be generated.

3.2.6 Diagnostics using findMUS

Diagnostic generation produces command line output containing source and dest node and grouped by the constraints violated in MiniZinc. When given `--output-json` it should also produce a machine readable `conflicts.json` which can be ingested by CVI to show these errors in VSCode.

3.2.7 Detailed MiniZinc constraint model

The following assumes some familiarity with MiniZinc syntax. More about MiniZinc, it's usage and syntax can be found here.

In the model below, the `nodeEnclave` decision variable stores the enclave assignment for each node, the `taint` decision variable stores the label assignment for each node, and the `xdedge` decision variable stores whether a given edge is in the enclave cut (i.e., the source and destination nodes of the edge are in different enclaves. Several other auxiliary decision variables are used in the constraint model to express the constraints or for efficient compilation. They are described later in the model.

The solver will attempt to assign a node annotation label to all nodes except a user annotated function. Only user annotated functions may have a function annotation. Functions lacking a function annotation cannot be invoked cross-domain and can only have exactly one taint across all invocations. This ensures that the arguments, return and function body only touch the same taint.

3.2.7.1 General Constraints on Output and Setup of Auxiliary Decision Variables

Every global variable and function entry must be assigned to a valid enclave. Instructions and parameters are assigned the same enclave as their containing functions. Annotations can not assigned to a valid enclave and they must be assigned to `nullEnclave`.

```
constraint :: "VarNodeHasEnclave"                forall (n in VarNode)
 ↪  (nodeEnclave[n]!=nullEnclave);
constraint :: "FunctionHasEnclave"               forall (n in
 ↪  FunctionEntry)      (nodeEnclave[n]!=nullEnclave);
```

```
constraint :: "InstHasEnclave"                  forall (n in Inst)
 ↪  (nodeEnclave[n]==nodeEnclave[hasFunction[n]]);
constraint :: "ParamHasEnclave"                 forall (n in Param)
 ↪  (nodeEnclave[n]==nodeEnclave[hasFunction[n]]);
constraint :: "AnnotationHasNoEnclave"          forall (n in Annotation)
 ↪  (nodeEnclave[n]==nullEnclave);
```

The level of every node that is not an annotation stored in the `nodeLevel` decision variable must match:

- the level of the label (taint) assigned to the node
- the level of the enclave the node is assigned to

```
constraint :: "NodeLevelAtTaintLevel"           forall (n in
 ↪  NonAnnotation)      (nodeLevel[n]==hasLabelLevel[taint[n]]);
constraint :: "NodeLevelAtEnclaveLevel"         forall (n in
 ↪  NonAnnotation)      (nodeLevel[n]==hasEnclaveLevel[nodeEnclave[n]]);
```

Only function entry nodes can be assigned a function annotation label. Furthermore, only the user can bless a function with a function annotation (that gets be passed to the solver through the input).

```
constraint :: "FnAnnotationForFnOnly"           forall (n in
 ↪  NonAnnotation)      (isFunctionAnnotation[taint[n]] ->
 ↪  isFunctionEntry(n));
constraint :: "FnAnnotationByUserOnly"          forall (n in
 ↪  FunctionEntry)      (isFunctionAnnotation[taint[n]] ->
 ↪  userAnnotatedFunction[n]);
```

Set up a number of auxiliary decision variables:

- `ftaint[n]`: CLE label taint of the function containing node `n`
- `esEnclave[e]`: enclave assigned to the source node of edge `e`
- `edEnclave[e]`: enclave assigned to the destination node of edge `e`
- `xdedge[e]`: source and destination nodes of `e` are in different enclaves
- `esTaint[e]`: CLE label taint of the source node of edge `e`
- `edTaint[e]`: CLE label taint of the destination node of edge `e`
- `tcedge[e]`: source and destination nodes of `e` have different CLE label taints
- `esFunTaint[e]`: CLE label taint of the function containing source node of edge `e`, `nullCleLabel` if not applicable
- `edFunTaint[e]`: CLE label taint of the function containing destination node of edge `e`, `nullCleLabel` if not applicable
- `esFunCdf[e]`: if the source node of the edge `e` is an annotated function, then this variable stores the CDF with the remotelevel equal to the level of the taint of the destination node; `nullCdf` if a valid CDF does not exist

- `edFunCdf[e]`: if the destination node of the edge `e` is an annotated function, then this variable stores the CDF with the remotelevel equal to the level of the taint of the source node; `nullCdf` if a valid CDF does not exist

```
constraint :: "MyFunctionTaint"                 forall (n in PDGNodeIdx)
 ↪  (ftaint[n] == (if hasFunction[n]!=0 then taint[hasFunction[n]] else
 ↪  nullCleLabel endif));
constraint :: "EdgeSourceEnclave"               forall (e in PDGEdgeIdx)
 ↪  (esEnclave[e]==nodeEnclave[hasSource[e]]);
constraint :: "EdgeDestEnclave"                 forall (e in PDGEdgeIdx)
 ↪  (edEnclave[e]==nodeEnclave[hasDest[e]]);
constraint :: "EdgeInEnclaveCut"                forall (e in PDGEdgeIdx)
 ↪  (xdedge[e]==(esEnclave[e]!=edEnclave[e]));
constraint :: "EdgeSourceTaint"                 forall (e in PDGEdgeIdx)
 ↪  (esTaint[e]==taint[hasSource[e]]);
constraint :: "EdgeDestTaint"                   forall (e in PDGEdgeIdx)
 ↪  (edTaint[e]==taint[hasDest[e]]);
constraint :: "EdgeTaintMismatch"               forall (e in PDGEdgeIdx)
 ↪  (tcedge[e]==(esTaint[e]!=edTaint[e]));
constraint :: "SourceFunctionAnnotation"        forall (e in PDGEdgeIdx)
 ↪  (esFunTaint[e] == (if sourceAnnotFun(e) then
 ↪  taint[hasFunction[hasSource[e]]] else nullCleLabel endif));
constraint :: "DestFunctionAnnotation"          forall (e in PDGEdgeIdx)
 ↪  (edFunTaint[e] == (if destAnnotFun(e) then
 ↪  taint[hasFunction[hasDest[e]]] else nullCleLabel endif));
constraint :: "SourceCdfForDestLevel"           forall (e in PDGEdgeIdx)
 ↪  (esFunCdf[e] == (if sourceAnnotFun(e) then
 ↪  cdfForRemoteLevel[esFunTaint[e], hasLabelLevel[edTaint[e]]] else
 ↪  nullCdf endif));
constraint :: "DestCdfForSourceLevel"           forall (e in PDGEdgeIdx)
 ↪  (edFunCdf[e] == (if destAnnotFun(e) then
 ↪  cdfForRemoteLevel[edFunTaint[e], hasLabelLevel[esTaint[e]]] else
 ↪  nullCdf endif));
```

If a node `n` is contained in an unannotated function then the CLE label taint assigned to the node must match that of the containing function. In other words, since unannotated functions must be singly tainted, all noded contained within the function must have the same taint as the function.

```
constraint :: "UnannotatedFunContentTaintMatch" forall (n in
 ↪  NonAnnotation where hasFunction[n]!=0)
 ↪  (userAnnotatedFunction[hasFunction[n]]==false ->
 ↪  taint[n]==ftaint[n]);
```

If the node `n` is contained in an user annotated function, then the CLE label taint assigned to the node must be allowed by the CLE JSON of the function annotation in the argument taints, return taints, or code body taints. In other words, any node contained within a function blessed with a function-annotation by the user can only contain nodes with taints that are explicitly permitted (to be coerced) by the function annotation.

```
constraint :: "AnnotatedFunContentCoercible"    forall (n in
 ↪  NonAnnotation where hasFunction[n]!=0 /\ isFunctionEntry(n)==false)
 ↪  (userAnnotatedFunction[hasFunction[n]] -> isInArctaint(ftaint[n],
 ↪  taint[n], hasLabelLevel[taint[n]]));
```

3.2.7.2 Constraints on the Cross-Domain Control Flow The control flow can never leave an enclave, unless it is done through an approved cross-domain call, as expressed in the following three constraints. The only control edges allowed in the cross-domain cut are either call invocations or returns. For any call invocation edge in the cut, the function annotation of the function entry being called must have a CDF that allows (with or without redaction) the level of the label assigned to the callsite. The label assigned to the callsite must have a node annotation with a CDF that allows the data to be shared with the level of the (taint of the) function entry being called.

```
constraint :: "NonCallControlEnclaveSafe"      forall (e in
 ↪  ControlDep_NonCall where isAnnotation(hasDest[e])==false)
 ↪  (xdedge[e]==false);
constraint :: "XDCallBlest"                    forall (e in
 ↪  ControlDep_CallInv) (xdedge[e] ->
 ↪  userAnnotatedFunction[hasDest[e]]);
constraint :: "XDCallAllowed"                  forall (e in
 ↪  ControlDep_CallInv) (xdedge[e] ->
 ↪  allowOrRedact(cdfForRemoteLevel[edTaint[e],
 ↪  hasLabelLevel[esTaint[e]]]));
```

Notes:

1. No additional constraint is needed for control call return edges; checking the corresponding call invocation suffices, however, later on we will check the data return edge when checking label coercion.

2. The conflict analyzer is working with the annotated unpartitioned code and not the fully partitioned code which will includes autogenerated code. The actual cut in the partitioned code with autogenerated code to handle cross-domain communications will be between the cross-domain send and receive functions that are several steps removed from the cut in the `xdedge` variable at this stage of analysis. The autogenerated code will apply annotations to cross-domain data annotations

that contain GAPS tags, and they will have a different label. So we cannot check whether the label of the arguments passed from the caller matches the argument taints allowed by the called function, or if the return taints match the value to which the return value is assigned. A downstream verification tool will check this.

3.2.7.3 Constraints on the Cross-Domain Data Flow Data can only leave an enclave through parameters or return of valid cross-domain call invocations, as expressed in the following three constraints.

Any data dependency edge that is not a data return cannot be in the cross-domain cut. For any data return edge in the cut, the taint of the source node (the returned value in the callee) must have a CDF that allows the data to be shared with the level of the taint of the destination node (the return site in the caller). For any parameter passing edge in the cut, the taint of the source node (what is passed by the callee) must have a CDF that allows the data to be shared with the level of the taint of the destination node (the corresponding actual parameter node of the callee function).

```
constraint :: "NonRetNonParmDataEnclaveSafe"   forall (e in
 ↪ DataEdgeNoRet)      (xdedge[e]==false);
constraint :: "XDCDataReturnAllowed"           forall (e in
 ↪ DataDepEdge_Ret)    (xdedge[e] ->
 ↪ allowOrRedact(cdfForRemoteLevel[esTaint[e],
 ↪ hasLabelLevel[edTaint[e]]]));
constraint :: "XDCParmAllowed"                 forall (e in Parameter)
 ↪ (xdedge[e] -> allowOrRedact(cdfForRemoteLevel[esTaint[e],
 ↪ hasLabelLevel[edTaint[e]]]));
```

3.2.7.4 Constraints on Taint Coercion Within Each Enclave While the constraints on the control dependency and data dependency that we discussed governed data sharing at the cross-domain cut, we still need to perform taint checking to ensure that data annotated with different labels inside each enclave are managed correctly and only when the mixing of the taints is explicitly allowed by the user.

Labels can be cooerced (i.e., nodes of a given PDG edge can be permitted to have different label assigments) inside an enclave only through user annotated functions. To track valid label coercion across a PDG edge `e`, the model uses an additional auxiliary decision variable called `coerced[e]`.

Any data dependency or parameter edge that is intra-enclave (not in the cross-domain cut) and with different CLE label taints assigned to the source and destination nodes must be coerced (through an annotated function).

Note: one may wonder whether a similar constraint must be added for control dependency edges at the entry block for completeness. Such a constraint is not

necessary given our inclusion of the `UnannotatedFunContentTaintMatch` and `AnnotatedFunContentCoercible` constraints discussed earlier.

```
constraint :: "TaintsSafeOrCoerced"            forall (e in
 ↪  DataEdgeParam)      ((tcedge[e] /\ (xdedge[e]==false)) ->
 ↪  coerced[e]);
```

If the edge is a parameter in or parameter out edge, then it can be coerced if and only if the associated function annotation has the taint of the other node in the argument taints for the corresponding parameter index. In other words, what is passed in through this parameter has a taint allowed by the function annotation.

```
constraint :: "ArgumentTaintCoerced"
 forall (e in Parameter_In union Parameter_Out)
  (if     destAnnotFun(e)   /\ isParam_ActualIn(hasDest[e])    /\
 ↪  (hasParamIdx[hasDest[e]]>0)
   then coerced[e] == hasArgtaints[edFunCdf[e], hasParamIdx[hasDest[e]],
 ↪  esTaint[e]]
   elseif sourceAnnotFun(e) /\ isParam_ActualOut(hasSource[e]) /\
 ↪  (hasParamIdx[hasSource[e]]>0)
   then coerced[e] == hasArgtaints[esFunCdf[e],
 ↪  hasParamIdx[hasSource[e]], edTaint[e]]
   else true
   endif);
```

If the edge is a data return edge, then it can be coerced if and only if the associated function annotation has the taint of the other node in the return taints.

```
constraint :: "ReturnTaintCoerced"            forall (e in
 ↪  DataDepEdge_Ret)     (coerced[e] == (if sourceAnnotFun(e) then
 ↪  hasRettaints[esFunCdf[e], edTaint[e]] else false endif));
```

If the edge is a data dependency edge (and not a return or parameter edge), then it can be coerced if and only if the associated function annotation allows the taint of the other node in the argument taints of any parameter,

Note that this constraint might appear seem redundant given the `AnnotatedFunContentCoercible` constraint discussed earlier. On closer inspection we can see that the following constraint also includes edges between nodes in the function and global/static variables; the earlier constraint does not. There is overlap between the constraints, so some refinement is possible, which may make the model a little harder to understand.

```
constraint :: "DataTaintCoerced"
 forall (e in DataEdgeNoRetParam)
  (if (hasFunction[hasSource[e]]!=0 /\ hasFunction[hasDest[e]]!=0 /\
 ↪  hasFunction[hasSource[e]]==hasFunction[hasDest[e]])
```

```
  then coerced[e] == (isInArctaint(esFunTaint[e], edTaint[e],
↪  hasLabelLevel[edTaint[e]]) /\
                     isInArctaint(esFunTaint[e], esTaint[e],
↪  hasLabelLevel[esTaint[e]]))     % source and dest taints okay
  elseif (isVarNode(hasDest[e]) /\ hasFunction[hasSource[e]]!=0)
  then coerced[e] == (isInArctaint(esFunTaint[e], edTaint[e],
↪  hasLabelLevel[edTaint[e]]) /\
                     isInArctaint(esFunTaint[e], esTaint[e],
↪  hasLabelLevel[esTaint[e]]))
  elseif (isVarNode(hasSource[e]) /\ hasFunction[hasDest[e]]!=0)
  then coerced[e] == (isInArctaint(edFunTaint[e], esTaint[e],
↪  hasLabelLevel[esTaint[e]]) /\
                     isInArctaint(edFunTaint[e], edTaint[e],
↪  hasLabelLevel[edTaint[e]]))
  else coerced[e] == false
  endif);
```

3.2.7.5 Solution Objective In this model, we require the solver to provide a satisfying assignment that minimizes the total number of call invocation that are in the cross-domain cut. Other objectives could be used instead.

```
var int: objective = sum(e in ControlDep_CallInv where xdedge[e])(1);
solve minimize objective;
```

3.3 Code Dividing and Refactoring

Once the CAPO partitioning conflict analyzer has analyzed the CLE-annotated application code, and determined that all remaining conflicts are resolvable by RPC-wrapping to result in a security compliant cross-domain partitioned code, the conflict analyzer will save the code in the refactored directory along with a a topology file (JSON) containing the assignment of every function and global variable to an enclave/level. A sample topology JSON is provided below.

```
{
   "levels": ["orange", "purple"],
   "enclaves": ["purple_E", "orange_E"],
   "source_path": ["./refactored"],
   "functions": [
      {"name": "get_a", "level": "orange", "enclave": "orange_E",
↪       "file": "test1_refactored.c", "line": 29},
      {"name": "main",  "level": "purple", "enclave": "orange_E",
↪       "file": "test1_refactored.c", "line": 35},
      // ...
```

```
    ],
    "global_scoped_vars": [
      {"name": "globalScopeVarNotFunctionStatic", "level": "purple",
       ↪  "file": "test1_refactored.c", "line": 5},
      // ...
    ],
}
```

Given the refactored, annotated application, and the topology, the divider creates a `divvied` directory, divides the code into files in separate subdirectories (one per enclave), such that the source code for each function or global variable is placed in its respective enclave. Furthermore, all related code like type, variable, and function declarations, macro definitions, header includes, and pragmas are handled, so that the source in each directory has all the relevant code, ready for automated partitioning and code generation for RPC-wrapping of functions, and marshaling, tagging, serialization, and DFDL description of cross-domain data types.

This `divvied` source becomes the input to the GAPS Enclave Definition Language (GEDL) generator tool. The GEDL drives further code generation and modification needed to build the application binaries for each enclave.

The usage of the program divider is straightforward:

```
divider [-h] -f TOPOLOGYJSON [-o OUTPUT_DIR] [-m] [-c CLANG_ARGS]
```

The divider will always look in a directory called `refactored` under the working directory for any `*.c` or `*.h` files that may comprise the program. The `-m` produces a debug mapping and `-c` can be useful because the divider uses clang internally to parse each file.

3.4 Autogeneration

3.4.1 GEDL

The GEDL is a JSON document specifying all the cross domain calls and their associated data including the arguments and return type. For function arguments, the directional assignment of that argument is indicated: `input` (a source argument, the function is reading in this value), `output` (a destination argument, the function is writing data to this argument), `both` (the argument is used as both an input and output argument in the function). This specification enables the RPC logic to auto-generate the required code to marshal the data accordingly.

This gedl is generated in an llvm `opt` pass which analyzes the divvied code. Whether a parameter is an input or an output is determined using heuristics based on function signatures, for example, in `memset` the first argument is an output. If the `opt` pass is

unable to infer whether a parameter is an input or output, then it will leave placeholders in the gedl JSON, given by `user_input`.

The gedl file format is described in `gedl_schema.json` which can be found in the appendix

The usage of the gedl pass is as follows:

```
opt -load libgedl.so
-prog <programName>    Name of partitioned program
-schema <schemaPath>   Relative path to gedl schema
-prefix <prefix>       Specify prefix for output
-he <heuristicPath>    Path to heuristics directory
-l <level>             Parameter tree expand level
-d <debugInfo>         use debug information
-u <uprefix>           Specify prefix for untrusted side
-t <tprefix>           Specify prefix for trusted side
-defined <defined>     Specify defined functions file path
-suffix <suffix>       Specify suffix for code injection
```

3.4.2 IDL

The IDL is a text file with a set of C style struct data type definitions, used to facilitate subsequent serialization and marshaling of data types. The IDL syntax is based on C; an IDL file contains one or more C struct datatypes. Two structs are generated for each TAG request and response pair, with the in arguments in the request and the out arguments in the response.

Datatype tags are assigned numerically in the order the structs are found in the file. Not all valid C struct declarations are supported.

Currently we support the following atomic types:

1. char (8b)
2. unsigned char (8b)
3. short (16b)
4. unsigned short (16b)
5. int (32b)
6. unsigned int (32b)
7. long (64b)
8. unsigned long (64b)
9. float (4B)
10. double (8B).

Fixed-size arrays of the supported atomic types are also supported.

The `idl_generator` script takes a gedl JSON and produces the idl. The usage is as follows:

```
usage: idl_generator.py [-h] -g GEDL -o OFILE -i
 ↪  {Singlethreaded,Multithreaded} [-s SCHEMA] [-L]

CLOSURE IDL File Generator

optional arguments:
  -h, --help            show this help message and exit
  -g GEDL, --gedl GEDL  Input GEDL Filepath
  -o OFILE, --ofile OFILE
                        Output Filepath
  -i {Singlethreaded,Multithreaded}, --ipc
 ↪  {Singlethreaded,Multithreaded}
                        IPC Type
  -s SCHEMA, --schema SCHEMA
                        override the location of the of the schema if
 ↪  required
  -L, --liberal         Liberal mode: disable gedl schema check
```

Note: The schema is the gedl schema which validates the input gedl during the IDL generation. `-L` disables this check. `-i` refers to the ipc threading mode. See the RPC section for more info.

A sample .idl can be found in the appendix.

3.4.3 Codecs

For each struct in the IDL, a codecs is generated to facilitate serialization to the remote enclave. The codecs consist of encode, decode, print functions for each of the structs, which handle byte order conversions between host and network byte order.

The codecs can be generated using `hal_autogen` which is described in the DFDL section.

In addition to the codecs, DFDL description of each serialized request/response is generated by the DFDL writer.

The generated codecs are registered in the HAL daemon. The generated RPC code registers these codecs for use of the xdcomms send/recv functions. It includes a rpc_wrapper/handler for each function called cross domain.

3.4.4 Remote Procedure Call (RPC)

The `rpc_generator.py` is used to generate the `*_rpc.{c,h}` for compilation. The rpc code automates cross domain function invocation and replaces calls to those functions with ones that additionally marshall and send the data across the network using the generated codecs and IDL.

The rpc generator takes as input:

1. Partitioned application code including name of main program,
2. CLE annotations for each individual function
3. The generated gedl
4. Input/output ZMQ uris
5. Base values for `mux`, `sec`, `typ` parameters
6. The cle user annotations for reliability parameters (retries, timeout and idempotence).

It produces C source and header files for CLE-annotated RPC code for each partition including the RPC wrapper and peer call handler. Additionally, a `xdconf.ini` is generated, which a separate script uses to configure HAL. The CLE-annotated RPC code contains the definitions for each `TAG_` request/response pair. It also generates the input application code with the following modifications:

1. It adds HAL init and RPC headers to main program
2. It renames cross domain calls in original source from foo() to _rpc_foo()
3. On the partition without the main, it will create a main program and a handler loop awaiting the RPC calls

Additionally, there are two IPC modes for the generated rpc code, which either can either generate singlethreaded or multithreaded rpc handlers. The multithreaded mode provides one RPC handler thread per cross domain function, while the singlethreaded mode has one global rpc handler for all cross domain calls.

The RPC generator usage is as follows:

```
CLOSURE RPC File and Wrapper Generator

optional arguments:
  -h, --help            show this help message and exit
  -a HAL, --hal HAL     HAL Api Directory Path
  -e EDIR, --edir EDIR  Input Directory
  -c CLE, --cle CLE     Input Filepath for CLE user annotations
  -g GEDL, --gedl GEDL  Input GEDL Filepath
  -i IPC, --ipc IPC     IPC Type (Singlethreaded/Multithreaded)
  -m MAINPROG, --mainprog MAINPROG
```

```
                        Application program name, <mainprog>.c must
↪  exsit
 -n INURI, --inuri INURI
                        Input URI
 -o ODIR, --odir ODIR  Output Directory
 -s SCHEMA, --schema SCHEMA
                        override location of cle schema if required
 -t OUTURI, --outuri OUTURI
                        Output URI
 -v, --verbose
 -x XDCONF, --xdconf XDCONF
                        Hal Config Map Filename
 -E ENCLAVE_LIST [ENCLAVE_LIST ...], --enclave_list ENCLAVE_LIST
↪  [ENCLAVE_LIST ...]
                        List of enclaves
 -M MUX_BASE, --mux_base MUX_BASE
                        Application mux base index for tags
 -S SEC_BASE, --sec_base SEC_BASE
                        Application sec base index for tags
 -T TYP_BASE, --typ_base TYP_BASE
                        Application typ base index for tags
```

3.4.5 DFDL

DFDL is an extension of XSD which provides a way to describe binary formats and easily encode/decode from binary to an xml infoset. CLOSURE has the ability to create DFDL schemas for each cross domain request/response pair with use of the `hal_autogen`.

The `hal_autogen` is additionally used to write the codecs so it takes as input both the idl and the a `typ` base (which must match the one given to the `rpc_generator.py`) and outputs both the DFDL and the codecs.

The `hal_autogen` script can be used as follows:

```
usage: autogen.py [-h] -i IDL_FILE -g GAPS_DEVTYP -d DFDL_OUTFILE [-e
↪  ENCODER_OUTFILE] [-T TYP_BASE] [-c CLANG_ARGS]

CLOSURE Autogeneration Utility

optional arguments:
  -h, --help            show this help message and exit
  -i IDL_FILE, --idl_file IDL_FILE
                        Input IDL file
  -g GAPS_DEVTYP, --gaps_devtyp GAPS_DEVTYP
```

```
                        GAPS device type [bw_v1 or be_v1]
  -d DFDL_OUTFILE, --dfdl_outfile DFDL_OUTFILE
                        Output DFDL file
  -e ENCODER_OUTFILE, --encoder_outfile ENCODER_OUTFILE
                        Output codec filename without .c/.h suffix
  -T TYP_BASE, --typ_base TYP_BASE
                        Application typ base index for tags (must match
 ↪  RPC Generator)
  -c CLANG_ARGS, --clang_args CLANG_ARGS
                        Arguments for clang
```

3.4.6 HAL configuration forwarding rules

The `hal_autoconfig.py` script takes a `xdconf.ini` generated by the rpc generator and a `devices.json` and combines them to produce a *HAL-Daemon* configuration, the format of which is described in the next section.

```
usage: hal_autoconfig.py [-h] [-d JSON_DEVICES_FILE] [-o OUTPUT_DIR] [-p
 ↪  OUTPUT_FILE_PREFIX] [-v]
                          [-x JSON_API_FILE]

Create HAL configuration file

optional arguments:
  -h, --help            show this help message and exit
  -d JSON_DEVICES_FILE, --json_devices_file JSON_DEVICES_FILE
                        Input JSON file name of HAL device conig
  -o OUTPUT_DIR, --output_dir OUTPUT_DIR
                        Output directory path
  -p OUTPUT_FILE_PREFIX, --output_file_prefix OUTPUT_FILE_PREFIX
                        Output HAL configuration file name prefix
  -v, --verbose         run in verbose mode
  -x JSON_API_FILE, --json_api_file JSON_API_FILE
                        Input JSON file name of HAL API and tag-maps
```

Here, the `-d` refers to the device config and the `-x` refers to the `xdconf.ini` file. An example *HAL-Daemon* configuration can be found in the appendix. Note that the *HAL-Library* conifugration does not use `hal_autoconfig.py`, rather it directly uses the *xdconf.ini* file.

3.5 Hardware Abstraction Layer (HAL)

Partitioned application programs use HAL to communicate data through the GAPS devices, which we call Cross-Domain Guards (CDGs). HAL provides: a) data-plane routing, b) abstraction from the particulars of each CDG including device initialization and formatting, and c) invocation of CLOSURE autogenerated codecs to handle data serialization and de-serialization. This section describes the two current HAL implementations and their configuration.

- The *HAL-Daemon*: a highly flexible middleware implementation that communicates with application using 0MQ.
- The *HAL-Library*: a simpler, more efficient library implementation directly linked to partitioned applications.

First, however, we describe how HAL interfaces to: i) the Applications, which is common to both implementations, and ii) to the various supported CDGs, including all the GAPS CDGs (ESCAPE, ILIP and MIND) and the commercial CDG X-ARBITOR.

3.5.1 HAL XDCOMMS API

This section describes the HAL cross-domain communication (XDCOMMS) data-plane API, which is independent of the particular CDG or HAL implementation.

- The API simplifies cross-domain design by abstracting CDG details from the software application developer and the rest of the CLOSURE toolchain. In particular, HAL abstracts the details of each CDG initialization, networking and formatting requirements.
- Both HAL implementations (*HAL-Daemon* and *HAL-Library*) support all the same API function calls, allowing both implementations to run the same compiled code.

Applications interact with the HAL by linking to HAL API functions. These calls to the functions are auto-generated as part of the *RPC generator* portion of the CLOSURE Automagic build step; thus the API is primarily of interest to cross-domain CLOSURE tool development. The API can also be directly called by application developers to provide an abstract data-plane API to supported CDGs. Note, however, that the CDG control-planes API for setting up rules for different data types is not currently part of HAL.

3.5.1.1 HAL Data Plane Send and Receive API Applications can send and receive data by simply passing pointers to in-memory Application Data Unit (ADU) structures. In addition, applications pass control information in a *HAL tag* structure that identifies the session, security policy and date type. HAL uses the tag information to route data to the correct interface and process the data. The *HAL tag* structure has three orthogonal 32-bit unsigned identifiers: *<mux, sec, typ>*, where:

- **mux** is the session multiplexing handle used to identify a unidirectional application flow.
- **sec** identifies the CDG security policy used to processing an ADU.
- **typ** identifies the type of ADU (based on DFDL xsd definition). The tag *typ* tells HAL how to serialize the ADU. The CDG can also use the tag *typ* (and its associated description) in order to process (e.g., downgrade) the ADU contents based on its rulesets.

In particular, the HAL API provides one send and two receive calls:

- Asynchronous send, where the message is sent and the call returns immediately.
- Blocking receive, where the call will block until a message matching the specified tag is received.
- Non-blocking receive, where the call blocks until either the message is received or a timeout interval has passed

The value of the timeout interval for the Non-blocking receive is set in the xdc_sub_socket_non_blocking() call (see below). The associated send and receive function calls are:

```
void  xdc_asyn_send(void *socket, void *adu, gaps_tag *tag);
int   xdc_recv(void *socket, void *adu, gaps_tag *tag);
void  xdc_blocking_recv(void *socket, void *adu, gaps_tag *tag);
```

Note that, in the above send/receive functions, the *socket* pointer is only used by the *HAL-Daemon* implementation, which uses it to identify the configured publish and subscribe sockets on the HAL daemon. The *socket* pointer is not needed by *HAL-Library* implementation, where its value is ignored.

3.5.1.2 HAL Data Plane Control API In addition to the send/receive calls, other HAL API calls specify the:

- **Codec function**: The application must register (de-)serialization codec functions for all the datatypes that can be sent over the CDG. Once registered, the correct codec will be selected and invoked when data is sent or received by HAL.
- **Timeout**: used in a non-blocking receive. The application can specify a timeout value (in milliseconds) for each receive tag. If the timeout value is -1, then an xdc_recv() call will block until a message is available; else, for all positive timeout values, an xdc_recv() call will wait for a message for that amount of time before returning with -1 value.
- **Log level**: 0=TRACE, 1=DEBUG, 2=INFO, 3=LOG_WARN, 4=LOG_ERROR, 5=LOG_FATAL. Each log level prints its own level information and all higher levels. Level 2 (the default log level), for example, prints level 2 and higher log information, but no level 1 or level 0 information.

```
void  xdc_register(codec_func_ptr encode, codec_func_ptr decode, int typ);
void *xdc_sub_socket_non_blocking(gaps_tag tag, int timeout);
void  xdc_log_level(int new_level);
```

The HAL Data Plane Control API also defines the inter-process communication identifiers used between the Application and the *HAL-Daemon* implementation. This part of the API configures the 0MQ socket interface. The *HAL-library* implementation simply ignores the parts that setup the API (and allowing it to run the same compiled code used by the *HAL-Daemon* implementation).

```
extern char *xdc_set_in (char *addr);
extern char *xdc_set_out(char *addr);
extern void *xdc_ctx(void);
extern void *xdc_pub_socket(void);
extern void *xdc_sub_socket(gaps_tag tag);
```

Currently, the *HAL-Daemon* implements a 0MQ pub/sub pattern, with addresses (URIs) associated with the 0MQ publish and subscribe endpoints bound to the HAL daemon. The API provides two functions to set the HAL 0MQ endpoint addresses, and a third function to create the 0MQ context (returning a pointer to the context). Each application creates a single socket to send (publish) its data and one socket per receive tag to receive it data. All these functions connect to the HAL daemon listening 0MQ sockets, in order to send (on the API pub socket) or receive (on the API sub socket) data. The functions return a (void *) socket pointer.

3.5.2 HAL Supported CDGs

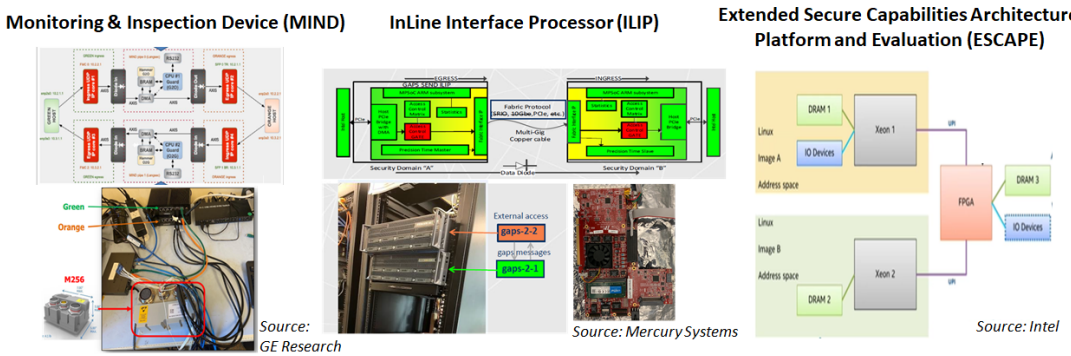


Figure 5: Range of Supported GAPS Hardware

HAL supports the three GAPS CDG hardware available today:

- Monitoring & Inspection Device (MIND)
  - Payload parsing/redaction in VHDL
  - Isolated Forward/reverse pipelines
  - Xilinx and GE avionics M256 form-factors
- InLine Interface Processor (ILIP)
  - Xilinx MPSoC bookends
  - Segmentation/Reassembly for large payloads (1 MB+, theoretical 1 GB)
  - Redaction guided by payload offsets
- Extended Secure Capabilities Architecture Platform and Evaluation (ESCAPE)
  - 2-Xeon CPUs connected over UPI to FPGA
  - Address-filtering to allow or disallow read/writes to shared memory pool
  - UPI transfer speeds up to 10.4 GT/s

In addition HAL supports:

- X-domain Agile Rules-Based Information Transfer OrchestratoR (X-ARBITOR)
  - Commercial gateway
  - Simultaneous, bidirectional transfers among different security domains
  - Reading and writing files via X-ARBITOR send and receive proxies.
- NULL Mode that requires no GAPS hardware.
  - Pass-through mode simplifies testing of the CLOSURE tool.
  - Used by the CLOSURE Emulator (EMU).

All the CDGs listed have been exhaustively tested with HAL. However, as shown in the final two columns of the table below, each HAL implementation - (Daemon and Library) - currently support only a subset of the CDGs.

CDG	Network	Message Format	Rx/Tx	Dae	Lib
ESCAPE	Shared Mem	CLD SHM pools	mmap memcpy	No	Yes
ILIP	INET	ILIP packets	read/write	Yes	No
MIND	DMA	C-Tag packet	mmap ioctl	Yes	Yes
X-ARBITOR	Unix Files	C-Tag packet	fread/fwrite	No	Yes
NULL	Serial/INET	C-Tag packet	read/write	Yes	No

The Message Format column defines one of several currently supported formats:

- *CLD SHM pools* are per-tag (per unidirectional flow) pools of shared memory (SHM). Each pool contains the flow's Control information (including Tag and buffer read and write indexes) and n buffers containing ADU Length and ADU Data.

- *ILIP packets* are the four ILIP packet formats supporting immediate and type 1, 2 or 3 payload messages.
- *C-Tag packet* contain the coded data (ADU) plus a header, which has a 32-bit compressed Tag, a 16-bit data length and 16-bit CRC.

HAL communicates with CDGs through host devices. The CDGs may be on the same host as the partitioned application (as a *bookend* device) or connected via one of the host's network interfaces (as a *bump-in-the-wire*). Examples of host devices include:

- *Serial devices* carrying TCP/IP packets: e.g., using /dev/dma_proxy_rx and /dev/dma_proxy_rx to communicate with the MIND FPGA (via the dma_proxy module).
- *Network devices* in client or server mode: e.g., using the host's eth0 device to communicate without a CDG (via TCP/IP or UDP/IP).
- *Shared-Memory* that has been memory-mapped (mmap'ed): e.g., using /dev/mem to communicate with the ESCAPE FPGA.
- *Files* in a Unix directory: e.g., using /tmp/xdc to communicate with the X-ARBITOR (via the X-ARBITOR proxies).

3.5.2.1 Emulated CDGs To simplify testing of partitioned application, HAL can also run partitioned application end-to-end on a single host computer using Emulated GAPS device. These emulated devices use the same network and message format used by the actual CDG hardware:

- *ESCAPE Emulation*: Shared Memory (SHM) mapped regions of the host computer act as a stand-in for the shared memory on the ESCAPE FPGA board. This is enabled by editing the SHM header file to define the 'SHM_MMAP_HOST' variable, then recompiling
- *ILIP Emulation*: Use the ILIP loopback driver to emulate the ILIP hardware using Immediate or Payload packets.
- *MIND Emulation*: We use the pseudo driver to emulate the MIND proxy DMA driver with the same ioctl commands. After installing the drive, it creates four unix devices: /dev/sue_donimous_rx0, /dev/sue_donimous_rx1, /dev/sue_donimous_tx0, and /dev/sue_donimous_tx1.
- X-ARBITOR Emulation: Files in directories on the host computer act as stand-in for the X-ARBITOR. This can be done by defining the 'FILE_DIR_SHARE' variable, then recompiling. This will put each flow in a separate directory named based on the compressed tag.
- NULL: This creates a loopback on a single host. This is implemented using a *socat* or *netcat* to loop the host's serial (e.g., /dev/tty) or INET (e.g., TCP or UDP) connections.

3.5.3 HAL Daemon Implementation

HAL daemon runs as a separate process from the application with the API being through a ZeroMQ (0MQ) pub/sub interface. The Application uses the HAL-API *xdcomms C library*, in order to connect to the two (a publish and a subscribe) HAL Daemon listening 0MQ sockets. The 0MQ sockets can use IPC or INET (e.g., ipc:///tmp/halpub, ipc:///tmp/halsub).

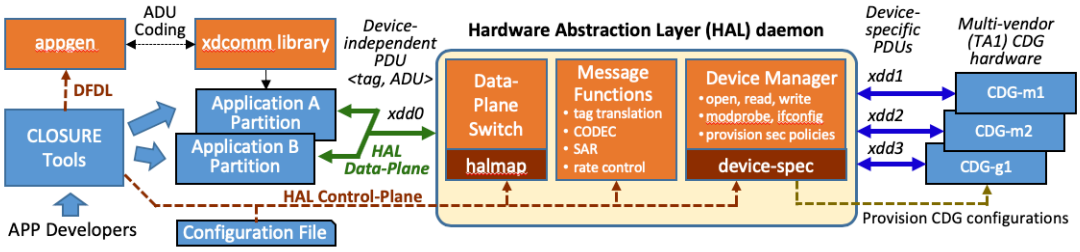


Figure 6: HAL interfaces between applications and Network Interfaces.

The HAL daemon service can be started manually or started by a *systemd* script at boot time. The figure shows an example of the HAL daemon supporting multiple applications and CDGs. The figure highlights the three major HAL daemon components:

3.5.3.1 Data Plane Switch The **Data Plane Switch** forwards packets (containing a tag and ADU) from one interface to another (e.g., from xdd0 to xdd1). Its forwarding in based on the arriving packet's interface name, the packet's *tag* value, and the HAL configuration file unidirectional mapping rules (**halmap**).

- When sending data from the applications (on the left side of HAL in the figure above) into the network (on the right side of HAL), HAL Message Functions will encode (and possibly translate) the **Application tag** into a **Network tag**.
- When receiving data from the network, HAL will decode (and possibly translate) the **Network tag** back into an **Application tag**.

3.5.3.2 Device Manager The **Device Manager** opens, configures and manages the different types of interfaces (real or emulated) based on the configuration file's device specification (**devices-spec**):

- Opening the devices specified in the configuration file, using each one's specified addressing/port and communication mode.
- Reading and writing packets. It waits for received packets on all the opened read interfaces (using a select() function) and transmits packets back out onto the halmap-specified write interface.

3.5.3.3 Message Functions The **Message Functions** transform and control packets exchanged between the applications and guard devices:

- *Tag translation* between the internal HAL format and the different CDG packet formats. Each CDG packet format has a separate HAL sub-component that performs the tag encoding and decoding: e.g., packetize_sdh_bw_v1.c and packetize_sdh_bw_v1.h.
- *Message mediation* is not currently supported, but may include functions such as multiplexing/demultiplexing, segmentation/reassembly and rate control.

3.5.3.4 HAL Daemon Command Options To see the HAL daemon command options, run with the -h option. Below shows the current options:

```
~/gaps/hal$ daemon/hal -h
Hardware Abstraction Layer (HAL) for GAPS CLOSURE project (version 0.11)
Usage: hal [OPTIONS]... CONFIG-FILE
OPTIONS: are one of the following:
 -f : log file name (default = no log file)
 -h : print this message
 -l : log level: 0=TRACE, 1=DEBUG, 2=INFO, 3=WARN, 4=ERROR, 5=FATAL (default = 0)
 -q : quiet: disable logging on stderr (default = enabled)
 -w : device not ready (EAGAIN) wait time in microseconds (default = 1000us): -1 exits i
CONFIG-FILE: path to HAL configuration file (e.g., test/sample.cfg)
```

3.5.3.5 HAL Daemon Configuration HAL daemon Configuration uses a a libconfig file specified when starting the HAL daemon (see the Quick Start Guide and the Install Guide).

The HAL daemon configuration file contains two sections:

- **devices-spec**, which specifies the device configuration for each HAL interface, including:
  - Interface ID (e.g., xdd1),
  - enable flag,
  - packet format,
  - communication mode,
  - device paths,
  - [optional] addresses and ports,
  - [optional] max packet size (HAL may perform Segment and Reassemble (SAR)),
  - [optional] max rate (bits/second).

- **halmap** routing rules and message functions applied to each allowed unidirectional link.
  - *from_* fields specifying the inbound HAL Interface ID and packet tag values,
  - *to_* fields specifying the outbound HAL Interface ID and packet tag values,
  - message functions specific to this path (e.g., ADU codec).

The test directory has examples of configuration files (with a .cfg) extension. Note that, if there are multiple HAL daemon instances on a node (e.g., for testing), then they must be configured with different interfaces.

3.5.3.6 Quick Start Guide

3.5.3.6.1 Download Sources, Build, and Install We have built and tested HAL on a Linux Ubuntu 20.04 system, and while HAL can run on other operating systems / versions, the package installation instructions are for that particular OS and version.

Install the HAL pre-requisite libraries.

```
sudo apt install -y libzmq3-dev
sudo apt install -y libconfig-dev
```

See the CLOSURE Dev Server Setup for full listing of CLOSURE external dependencies (some of which may be required for HAL on a newly installed system).

Clone the repository, then run make in order to compile HAL, together with its libraries (API and codecs) and test programs:

```
git clone https://github.com/gaps-closure/hal
cd hal
make clean; make
```

Some SDH devices also require installation of a device driver via an associated kernel module. Consult the GAPS Device provider's documentation.

3.5.3.6.2 Static Binaries To build a static version of you may need the additional packages for the included minimized static build of 0MQ

```
sudo apt install -y liblzma-dev
sudo apt install -y libunwind-dev
sudo apt install -y libsodium-dev
```

Once you have these dependencies you should simply need to run

```
make clean; make static
```

3.5.3.6.3 Configure/Run HAL on Target Hardware An instance of HAL daemon runs on each host or server that directly utilizes the SDH (cross-domain host), and requires a configuration file. If GAPS devices are already configured on enclave hosts in the target environment, we can simply start HAL daemon with the appropriate configuration file in each enclave host:

```
hal$ daemon/hal test/sample_6modemo_b{e|w}_{orange|green}.cfg # e.g. sample_6modemo_be_o
```

For this purpose, we have provided sample HAL daemon configuration files that model the Apr '20 demo setup, i.e., green-side and orange-side HAL configurations for either SDH-BE or SDH-BW usage. Note that provided configurations do not mix SDH types for the forward and reverse directions; we will provide these once the hybrid setup becomes available. Also note that contents of the config file may need to be changed depending on the target setup (i.e. SDH-BE device names and SDH-BW end-point IP addresses may differ from those used in current files).

Once the HAL daemon is started, we can run the mission application or a test application such as halperf on each enclave.

3.5.3.7 Quick Test of HAL with SDH-BE Loopback or SDH-BW Emulated Network During development, for testing HAL with SDH-BE loopback drivers or SDH-BW emulated networking, it is possible to run HAL instances for the different enclaves on the same physical machine using their respective configurations. If running this localized setup and if using SDH-BE, the loopback ILIP device driver kernel module `gaps_ilip.ko` must be built and installed using `insmod` before starting HAL.

```
# Obtain and untar driver source package from SDh-BE developer
cd loopback/ilip
# If using v0.2.0, edit line 426 in ilip_nl.c from #if 0  to  #if 1
vi ilip_nl.c
make clean; make install
insmod gaps_ilip.ko
```

If using SDH-BW, an emulated network (e.g., `test/6MoDemo_BW.net.sh` as shown below) must be configured before starting HAL to instantiate virtual ethernet devices and netcat processes to facilitate the packet movement. The `halperf` test application can then be invoked to send and receive the application traffic workload.

Steps for an end-to-end test for Apr '20 Demo testing on a single host are provided below.

1. Open five terminals (terminal1, terminal2, … terminal5).
2. Assuming SDH-BW for this example; start the emulated network in terminal3 (skip for SDH-BE):

```
terminal3:
  hal$ cd tests
  hal/tests$: sudo ./6MoDemo_BW.net.sh
```

3. Start HAL (this example assumes SDH-BW) for green and orange enclaves

```
terminal1 (green):
  hal$ daemon/hal test/sample_6modemo_bw_green.cfg

terminal2 (orange):
  hal$ daemon/hal test/sample_6modemo_bw_orange.cfg
```

4. An instance of halperf.py can both send and receive messages. Run an instance on both green and orange enclaves and send the appropriate mux/sec/typ combinations that correspond to the HAL tag specification for the Mission App datatypes:

```
terminal4 (green):
  export LD_LIBRARY_PATH=../appgen/6month-demo
  hal/test$ ./halperf.py -s 1 1 1 100 -r 2 2 1 -r 2 2 2 -i ipc:///tmp/halsubbwgreen -o ip

terminal5 (orange):
  export LD_LIBRARY_PATH=../appgen/6month-demo
  hal/test$ ./halperf.py -s 2 2 1 10 -s 2 2 2 100 -r 1 1 1 -i ipc:///tmp/halsubbworange -
```

Note the -i and -o arguments which correspond to input/output ZMQ interfaces utilized by HAL. The example provided is for SDH-BW. If using SDH-BE, replace 'bw' with 'be' for each -i and -o argument (e.g. halpubbworange –> halpubbeorange)

The sending rates in the above calls are 1 Hz for simplicity. (For the representative mission application rates, you can send 1,1,1 at 100Hz, 2,2,1 at 10Hz, and 2,2,2 at 100Hz instead. Other rates and application mixes can be used for stress-testing or for testing policy rules.) Example output:

```
terminal4 (green):
sent: [1/1/1] -- (-74.574489,40.695545,102.100000)
recv: [2/2/2] -- (-1.021000,2.334000,0.900000)
recv: [2/2/1] -- (-74.574489,40.695545,102.400000)

terminal5 (orange):
recv: [1/1/1] -- (-74.574489,40.695545,102.100000)
sent: [2/2/2] -- (-1.021000,2.334000,0.900000)
sent: [2/2/1] -- (-74.574489,40.695545,102.400000)
```

3.5.3.7.1 Cleanup of HAL Components Ctrl-C can be used to kill most processes. Additional cleanup scripts are provided if needed:

```
hal/test$ ./kill_my_hall.sh f
hal/test$ sudo pkill -f "nc -klu"
hal/test$ sudo pkill -f "nc -u"
```

3.5.3.8 HAL Daemon Installation and Usage

3.5.3.8.1 Build HAL See Download Sources, Build, and Install for required steps.

3.5.3.8.2 Run HAL Starting the HAL daemon requires specifying a HAL configuration file and any options. The test directory has examples of configuration files (with a .cfg) extension.

HAL Loopback Mode

At its simplest, we can start HAL to echo send requests made back on the application interface. Loopback mode is enabled by specifying the loopback configuration file test/config_simple_exampes/sample_loopback.cfg

```
cd hal
hal$ daemon/hal test/config_simple_examples/sample_loopback.cfg
```

In this case, HAL receives packets on its application read interface and routes them back to its application write interface. This requires no network devices (or network access).

Below is an example of the logging output from the HAL daemon, showing its configuration:

- Single device called *xdd0*, using a pub/sub ipc connection (using connection mode sdh_ha_v1), with file descriptors 3 and 6 for reading and writing.
- A single HAL map (*halmap*) routing entry, which forwards application data from the application *xdd0* device with a HAL tag *<mux, sec, typ> = <1,1,1>* back to the application *xdd0* device. It also translates that tag to *<mux, sec, typ> = <1,2309737967,1>*

```
hal$ daemon/hal test/sample_loopback.cfg
HAL device list:
 xdd0 [v=1 d=./zc/zc m=sdh_ha_v1 c=ipc mi=sub mo=pub fr=3 fw=6]
HAL map list (0x5597a6af8150):
 xdd0 [mux=01 sec=01 typ=01] ->  xdd0 [mux=01 sec=2309737967 typ=01] , codec=NULL

HAL Waiting for input on fds, 3
```

HAL Test Driver (halperf.py)

We provide an easy to use utility, halperf.py, for sending and receiving Mission App datatypes (Position/Distance) while utilizing HAL and SDH. halperf constructs an in-memory instance of the datatype, provides it to HAL with appropriate application tag, HAL maps it to the configured SDH, constructs the on-wire format, and releases a frame to the SDH. The receive-side HAL unrolls the frame and provides it to the receiving halperf instance.

```
usage: halperf.py [-h] [-s MUX SEC TYP RATE] [-r MUX SEC TYP] [-l PATH]
                  [-x PATH] [-i URI] [-o URI]

optional arguments:
  -h, --help            show this help message and exit
  -s MUX SEC TYP RATE, --send MUX SEC TYP RATE
                        send cross-domain flow using MUX/SEC/TYP at RATE (Hz)
  -r MUX SEC TYP, --recv MUX SEC TYP
                        recv cross-domain flow mapped to MUX/SEC/TYP
  -l PATH               path to mission app shared libraries
                        (default=../appgen/6month-demo)
  -x PATH               path to libxdcomms.so (default=../api)
  -i URI                in URI (default=ipc:///tmp/halpub1)
  -o URI                out URI (default=ipc:///tmp/halsub1)
```

The HAL daemon configuration uses a libconfig File, which contains HAL maps (routes) and Device configurations.

3.5.4 HAL Library Implementation

This section describes the HAL-Library implementation, a simple, efficient implementation directly linked to partitioned applications. The application run the linked HAL API functions directly within their process.

3.5.4.1 HAL Library Configuration The *HAL-Library* uses three sources of configuration information:

- **JSON configuration file**. The file defines the one-way channel definitions, specifying channel tags and enclave end-point names.
- **UNIX environment variables**. Selection of device configuration is done through UNIX environment variables specified when running the partitioned application.
- **Device header files**. XDCOMMS-lib has a header file that defines the communication device configuration:
  - **DMA**: dma-proxy.h.
  - **SHM**: shm.h.
  - **FILE**: file_info.h.

A simple example of a *HAL-Library* JSON configuration file is the one used by the test application: xdconf_app_req_rep.json. It supports two types of client requests/responses (using position or raw data) JSON configuration files are auto-generated by the RPC generator portion of the CLOSURE tools and output as an *xdconf.ini* file. An auto-configured example with 10 different unidirectional tags, used in the websrv demo, is xdconf.ini.

The table below list the current list of environment variables when using the HAL Library Implementation.

ENVIR. VARIABLE	Description
CONFIG_FILE	JSON configuration file
DEV_NAME_RX	Name of receive device
DEV_NAME_TX	Name of transmit device
DEV_TYPE_RX	Receiver Device type
DEV_TYPE_TX	Transmitter Device type
ENCLAVE	Enclave Name (in CONFIG_FILE)
LD_LIBRARY_PATH	Path to xdcomms library
SHM_WAIT4NEW	Wait for new client if not 0
XARB_IP	X-ARBITOR Proxy IP address
XARB_PORT	X-ARBITOR Proxy TCP PORT
XDCLOGLEVEL	Debug log level

The HAL XDCOMMS General Control APIs section describes the various log levels. The default values, which in some cases depend on the DEV_TYPE are given below. If the variable is not used with a DEV_TYPE, then it is marked as -).

ENVIR. VARIABLE	default value in {dma, file, shm}
CONFIG_FILE	REQUIRED (no default)
DEV_NAME_RX	{/dev/dma_proxy_rx, /tmp/xdc, /dev/mem}
DEV_NAME_TX	{/dev/dma_proxy_tx, /tmp/xdc, /dev/mem}
DEV_TYPE_RX	dma
DEV_TYPE_TX	dma
ENCLAVE	REQUIRED (no default)
LD_LIBRARY_PATH	Linux OS path
SHM_WAIT4NEW	{ -, -, 0 }
XARB_IP	{ -, 192.168.100.101, - }
XARB_PORT	{ -, 1124, - }
XDCLOGLEVEL	xdc_log_level() call; else 2

3.5.4.2 HAL Library Quick Start Guide

3.5.4.2.1 Install and Build the HAL-Library To install and build the *HAL-Library*, together with the test application and pseudo driver, run the following commands:

```
cd ~/gaps
git clone git@github.com:gaps-closure/xdcomms-dma
cd xdcomms-dma/
make clean
make
```

3.5.4.2.2 Load the Pseudo DMA Driver Once the xdcomms library is installed and built, we can install the pseudo DMA driver module into the kernel, if required, in order to emulate the MIND DMA CDG. The following are the commands to install the modules and re-compile the HAL xdcomms library:

```
cd ~/gaps/xdcomms-dma/pseudo
./reset_sue_dominous.sh
lsmod | grep sue; ls -l /dev/sue*
```

After installing the driver (see below), it should list four unix devices:

- /dev/sue_donimous_rx0,

- /dev/sue_donimous_rx1,
- /dev/sue_donimous_tx0,
- /dev/sue_donimous_tx1.

3.5.4.3 Test Application Examples The *HAL-Library* has a C application program written to test HAL functionality and performance. This test application (*test-app*) is included in the xdcomms-dma repository in the test_app directory. Below we describe how to configure and run the *test-app* both: a) on hosts with emulated CDGs, and b) in a testbed with the real CDG hardware. In the examples below, the *test-app* is used in its simplest form to generate a single request-response handshake, containing *x,y,z* position information (data type 1). In particular:

- The Green node sends a *client_request_position* message (with tag <1,1,1>), which causes
- The Orange to reply with a *server_response_position* message (with tag <2,2,1>).

3.5.4.3.1 Run Test Application on Emulated MIND CDG Hardware First we must Ensure the emulated pseudo DMA driver module loaded, then recompile XDCOMMS-lib based on the pseudo-driver parameters:

```
cd ~/gaps/xdcomms-dma/api
make clean
make -f Makefile.pseudo
```

Run the test application in two terminal windows on the same Linux host. In the first start the app for the orange enclave as a server:

```
cd ~/gaps/xdcomms-dma/test_app
ENCLAVE=orange CONFIG_FILE=xdconf_app_req_rep.json DEV_NAME_RX=sue_donimous_rx1 \
DEV_NAME_TX=sue_donimous_tx1 LD_LIBRARY_PATH=~/gaps/xdcomms-dma/api \
XDCLOGLEVEL=1 ./app_req_rep -e 2
```

In the second window start the app for the green enclave as a client:

```
cd ~/gaps/xdcomms-dma/test_app
ENCLAVE=green CONFIG_FILE=xdconf_app_req_rep.json DEV_NAME_RX=sue_donimous_rx0 \
DEV_NAME_TX=sue_donimous_tx0 LD_LIBRARY_PATH=~/gaps/xdcomms-dma/api \
XDCLOGLEVEL=1 ./app_req_rep
```

3.5.4.3.2 Run Test Application on Emulated SHM CDG Hardware Recompile XDCOMMS-lib to use use the host Shared Memory by defining the SHM_MMAP_HOST variable in *shm.h*:

```
cd ~/gaps/xdcomms-dma/api
sed -i 's@^//#define\ SHM_MMAP_HOST@#define\ SHM_MMAP_HOST@g' shm.h
make clean
make
```

Run the test application in two terminal windows. In the first start the app for the orange enclave as a server:

```
cd ~/gaps/xdcomms-dma/test_app
sudo ENCLAVE=orange CONFIG_FILE=xdconf_app_req_rep.json DEV_TYPE_RX=shm \
DEV_TYPE_TX=shm SHM_WAIT4NEW=1 LD_LIBRARY_PATH=~/gaps/xdcomms-dma/api \
XDCLOGLEVEL=1 ./app_req_rep -e 2
```

In the second window start the app for the green enclave as a client:

```
cd ~/gaps/xdcomms-dma/test_app
sudo ENCLAVE=green CONFIG_FILE=xdconf_app_req_rep.json DEV_TYPE_RX=shm \
DEV_TYPE_TX=shm LD_LIBRARY_PATH=~/gaps/xdcomms-dma/api \
XDCLOGLEVEL=1 ./app_req_rep
```

3.5.4.3.3 Run Test Application on Emulated X-ARBIOR Hardware Recompile XDCOMMS-lib to use use the put each flow in a separate directory by defining the FILE_DIR_SHARE variable in file_info.h to equal 0:

```
cd ~/gaps/xdcomms-dma/api
sed -i 's@1 // Flows share@0 // Flows share@g' file_info.h
make clean
make
```

Run the test application in two terminal windows. In the first start the app for the orange enclave as a server:

```
cd ~/gaps/xdcomms-dma/test_app
ENCLAVE=orange CONFIG_FILE=xdconf_app_req_rep.json DEV_TYPE_RX=file \
DEV_TYPE_TX=file LD_LIBRARY_PATH=~/gaps/xdcomms-dma/api \
DEV_NAME_RX=/tmp/xdc/2 DEV_NAME_TX=/tmp/xdc/1 XARB_IP=1.2.3.4 \
XDCLOGLEVEL=1 ./app_req_rep -e 2
```

In the second window start the app for the green enclave as a client:

```
cd ~/gaps/xdcomms-dma/test_app
ENCLAVE=green CONFIG_FILE=xdconf_app_req_rep.json DEV_TYPE_RX=file \
DEV_TYPE_TX=file LD_LIBRARY_PATH=~/gaps/xdcomms-dma/api \
DEV_NAME_TX=/tmp/xdc/2 DEV_NAME_RX=/tmp/xdc/1 XARB_IP=1.2.3.5 \
XDCLOGLEVEL=1 ./app_req_rep
```

3.5.4.3.4 Run Test Application on MIND CDG Hardware VNC into a demo laptop with the XILINX DMA BOARD (e.g., 10.109.23.205:1). Instruction for loading the PetaLinux images and starting the a53 process and the MicroBlaze soft microprocessor core can be found in the End of Phase 2 demo repository. Specifically, the setup and run notes. Note that:

- The a53 and mb PetaLinux images are put the compiled test code and its json configuration file into the /opt/closure/test_app directory.
- The DMA modules should be reset using the *modprobe* command (see below) for each run
- The green enclave should be started first followed within 3 seconds by the orange enclave.
- These notes also give instructions for running the compiled websrv application in the /opt/closure/websrv directory.

Run the test application in a53 and mb minicom terminal windows. In the first window start the app for the green enclave as a client (with the -l option specifying the log level as 1 (debug), which is equivalent to specifying the environmental variable XDCLOGLEVEL=1):

```
cd /opt/closure/test_app
modprobe -r dma_proxy && modprobe dma_proxy
ENCLAVE=green CONFIG_FILE=xdconf_app_req_rep.json ./app_req_rep -l 1
```

In the second window start the app for the orange enclave as a server:

```
cd /opt/closure/test_app
modprobe -r dma_proxy && modprobe dma_proxy
ENCLAVE=orange CONFIG_FILE=xdconf_app_req_rep.json ./app_req_rep -e 2 -l 1
```

3.5.4.3.5 Run Test Application on ESCAPE FPGA CDG Hardware VNC or SSH into two ESCAPE servers: escape-orange (e.g., 10.109.23.130) and escape-green (e.g., 10.109.23.131). First enable the ESCAPE Ruleset (pass-through mode) for the shared memory.

```
sudo escape_app -w 0x810 0x00000000 0x82000000
sudo escape_app -w 0x818 0x00000000 0x8200ffff
sudo escape_app -w 0x800 0x00000000 0x00000001
```

Then test the shared memory using the ESCAPE *memtest* code. On green terminal write some data into the start of the FPGA shared memory:

```
sudo memtest 0x2080000000 8 w w 0x123456789abcdef
```

In the orange terminal read the start of the FPGA shared memory:

```
sudo memtest 0x2080000000 32 w r
```

Load and Compile XDCOMMS-lib into each enclave

```
cd ~/gaps
git clone git@github.com:gaps-closure/xdcomms-dma
cd xdcomms-dma/
make clean
make
```

On the first server start the app for the orange enclave as a server:

```
cd ~/gaps/xdcomms-dma/test_app
sudo ENCLAVE=orange CONFIG_FILE=xdconf_app_req_rep.json DEV_TYPE_RX=shm \
DEV_TYPE_TX=shm SHM_WAIT4NEW=1 LD_LIBRARY_PATH=~/gaps/xdcomms-dma/api \
XDCLOGLEVEL=1 ./app_req_rep -e 2
```

In the second window start the app for the green enclave as a client:

```
cd ~/gaps/xdcomms-dma/test_app
sudo ENCLAVE=green CONFIG_FILE=xdconf_app_req_rep.json DEV_TYPE_RX=shm \
DEV_TYPE_TX=shm LD_LIBRARY_PATH=~/gaps/xdcomms-dma/api \
XDCLOGLEVEL=1 ./app_req_rep
```

3.5.4.4 Websrv Application Examples

This section gives an example of configuring and running a partitioned application created using the CLOSURE tools. The application is called Websrv and can be installed with:

```
git clone git@github.com:gaps-closure/eop2-closure-mind-demo
```

The .solution/partitioned/multithreaded directory has the partitioned binaries for the two enclaves (in the orange and green directories).

- The orange websrv binary (called *websrv-vid* in the DMA testbed image) sends video frames (using HAL) from a camera to the green node.
- The green Websrv binary (called *websrv-web* in the DMA testbed image) receives the frames (using HAL) acts as a Web server for the Websrv GUI
- Websrv GUI (running on a firefox browser) connects to green node on port 8443.

When starting the partitioned application binaries, the user must specify both the environmental variables used to configure HAL and the environmental variables to define orange node's Websrv IP addresses:

ENVIR. VARIABLE	Description	default value
CAMADDR	Camera IP address	REQUIRED (no default)
MYADDR	APPs IP address	REQUIRED (no default)

3.5.4.4.1 Configure/Run Websrv Application on ESCAPE

On the ESCAPE testbed we run on two terminals the first on the escape-orange node and the second on the escape-green node. In the first window start the orange partitioned app (changing the MYADDR and CAMADDR to the correct IP addresses):

```
  cd ~/gaps/eop2-closure-mind-demo/websrv/.solution/partitioned/multithreaded/orange
  make -f ../../../makefiles/Makefile.pseudo
  sudo CONFIG_FILE=../xdconf.ini LD_LIBRARY_PATH=~/gaps/xdcomms-dma/api \
ENCLAVE=orange MYADDR=10.109.23.128 CAMADDR=10.109.23.151 SHM_WAIT4NEW=1 \
DEV_TYPE_RX=shm DEV_TYPE_TX=shm XDCLOGLEVEL=1 ./websrv
```

In the second window start the green partitioned app:

```
  cd ~/gaps/eop2-closure-mind-demo/websrv/.solution/partitioned/multithreaded/green
  make -f ../../../makefiles/Makefile.pseudo
  sudo CONFIG_FILE=../xdconf.ini LD_LIBRARY_PATH=~/gaps/xdcomms-dma/api \
ENCLAVE=green DEV_TYPE_RX=shm DEV_TYPE_TX=shm XDCLOGLEVEL=1 ./websrv
```

The Websrv GUI (running on a Firefox browser) connects to IP address of the green node on port 8443.

The commands to run an emulation on a single host are the same as the above, except we recompile XDCOMMS-lib to use use the host Shared Memory by defining the SHM_MMAP_HOST variable in shm.h, as described in the description of the test_app emulation of shared memory.

3.5.4.4.2 Configure/Run Websrv Application on MIND hardware The MIND hardware runs Peta-linux images that already have the Websrv binaries compiled for the respective hardware:

- The a53 processor runs the *websrv-vid* server in the orange enclave.
- The MicroBlaze processor runs the *websrv-web* server in the green enclave.

The commands for the a53 window are therefore (changing the MYADDR and CAMADDR to the correct IP addresses):

```
cd /opt/closure/websrv
ENCLAVE=orange MYADDR=10.109.23.245 CAMADDR=10.109.23.151 CONFIG_FILE=xdconf.ini XDCLOG
```

The commands in the MicroBlaze window are:

```
cd /opt/closure/websrv
ENCLAVE=green CONFIG_FILE=xdconf.ini XDCLOGLEVEL=1 ./websrv-web
```

3.5.4.4.3 Configure/Run Websrv Application on MIND Emulated hardware We run on a host with the emulated pseudo DMA driver module loaded and recompile XDCOMMS-lib with pseudo-driver parameters.

In the first window start the orange partitioned app (changing the MYADDR and CAMADDR to the correct IP addresses):

```
cd ~/gaps/eop2-closure-mind-demo/websrv/.solution/partitioned/multithreaded/orange
make -f ../../../makefiles/Makefile.pseudo
CONFIG_FILE=../xdconf.ini LD_LIBRARY_PATH=~/gaps/xdcomms-dma/api \
ENCLAVE=orange MYADDR=10.109.23.128 CAMADDR=10.109.23.151 \
DEV_NAME_RX=sue_donimous_rx1 DEV_NAME_TX=sue_donimous_tx1 XDCLOGLEVEL=1 ./websrv
```

In the second window start the green partitioned app:

```
  cd ~/gaps/eop2-closure-mind-demo/websrv/.solution/partitioned/multithreaded/green
  make -f ../../../makefiles/Makefile.pseudo
  CONFIG_FILE=../xdconf.ini LD_LIBRARY_PATH=~/gaps/xdcomms-dma/api \
ENCLAVE=green \
DEV_NAME_RX=sue_donimous_rx0 DEV_NAME_TX=sue_donimous_tx0 XDCLOGLEVEL=1 ./websrv
```

3.6 Verifier

3.6.1 Overview

ECT/ParTV, a translation validation tool for post-partition verification [9] that certifies a partition is behaviorally equivalent to its original program and complies with the annotated fine-grained data sharing—the first such verification tool for program partitioners.

Determining and generating a secure partition that both respects the developer's annotations and is semantically equivalent to the original program is a non-trivial process prone to subtle bugs.

Given the security aims of program partitioning, partitioner correctness is paramount, but manually inspecting the generated partition for security violations and semantic inconsistencies would be nearly as burdensome as constructing the partition manually. ECT/ParTV confirms a form of behavioral equivalence between the original application and the generated partition checks the partition's adherence to the security policy defined by cle annotations and is solver-backed at each step for increased assurance.

ECT/ParTV verifies the equivalence between the refactored program and those in the various enclaves by checking, function-by-function, that the generated code and its annotations have been paritioned without undue modification and will thus behave like the refactored code. The tool, written in Haskell [10], loads both programs then starts establishing their equivalence. As it proceeds, it construct a modular, bottom-up proof in the Z3 [11] theorem prover that is both checked as the tool proceeds and written out in SMT-LIB format, a format that can audited both manually and by Z3 or another theorem prover able to read and check SMT-LIB files.

3.6.2 Invocation

ECT/ParTV can be invoked as follows:

```
Usage: ect [options] (orange.ll | purple.ll | ..)+ ref.ll (orange.json |
 ↪  purple.json | ..)+ ref.json
  -h                  --help                              Print help
  -f <function-name>  --entry-function=<function-name>  Specify the
 ↪  entry function (default: main)
```

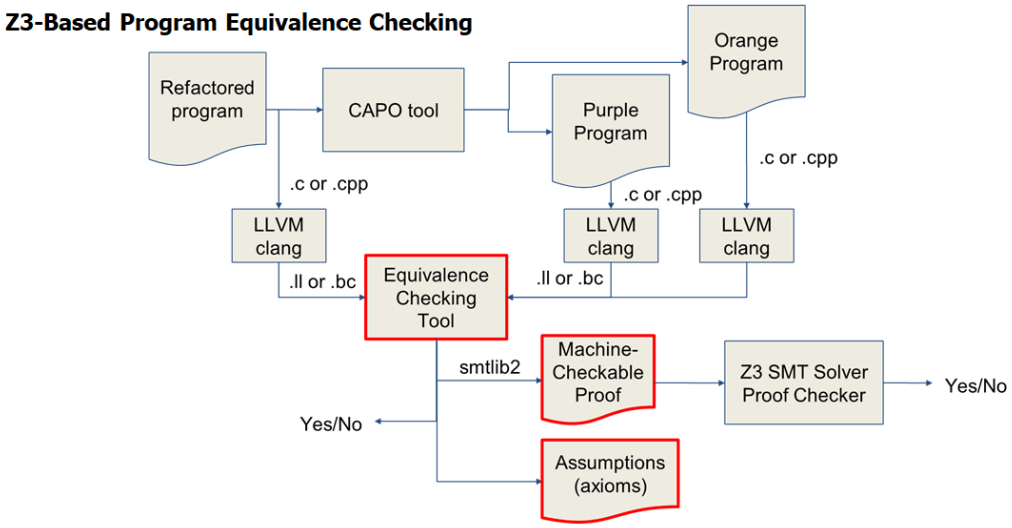


Figure 7: ECT/ParTV Workflow

```
  -d                    --display                          Dump information
↪  about the entry functions
  -l <log-file.smt2>  --logfile=<log-file.smt2>          Write the proof
↪  log to the given file after solving
```

As inputs, it takes in LLVM linked `.ll` files whose source files have been preprocessed by CLE, and it takes the corresponding CLE-JSON. It expects the partitioned `.ll` for each enclave first and the unpartitioned `.ll` last. It expects the CLE-JSON to correspond one-to-one with the LLVM files.

A linked `.ll` can be obtained from usage of `clang -S -emit-llvm` for each source file and `llvm-link` to combine multiple `.ll`s. More information on creating these `.ll`s can be found in the examples.

3.6.3 Downstream conflict analyzer

The conflict analyzer is run once more on each partition to check the consistency of the annotations in each partition. Each partition should have annotations associated with a single level, and should be internally consistent, with additional generated `TAG_` from the `rpc_generator` annotations being checked as well. The invocation of the conflict analyzer on the partitioned code is similar to that of the unpartitioned code, however, the exact invocation of the downstream conflict analyzer can be found in the examples.

3.7 Emulator (EMU)

The CLOSURE project provides a general purpose emulation tool for instantiating virtual cross domain architectures and running partitioned applications within them. The Emulator supports Multi-ISA and has been tested with x86 (Ubuntu focal kernel) and ARM64 (Ubuntu Xenial kernel) Linux. Built on top of QEMU, the emulator is flexible and can easily incoporate a QEMU instance of the target platform if need be. Integrated with the CLOSURE toolchain, the emulator can also be used stand-alone and is not limited to CLOSURE-partitioned applications (though this is the common usage). For a virtualized, distributed cross-domain scenario emulation, we build upon the NRL CORE [12] software.

3.7.1 Topology configuration, Generation and Plumbing using CORE and qemu

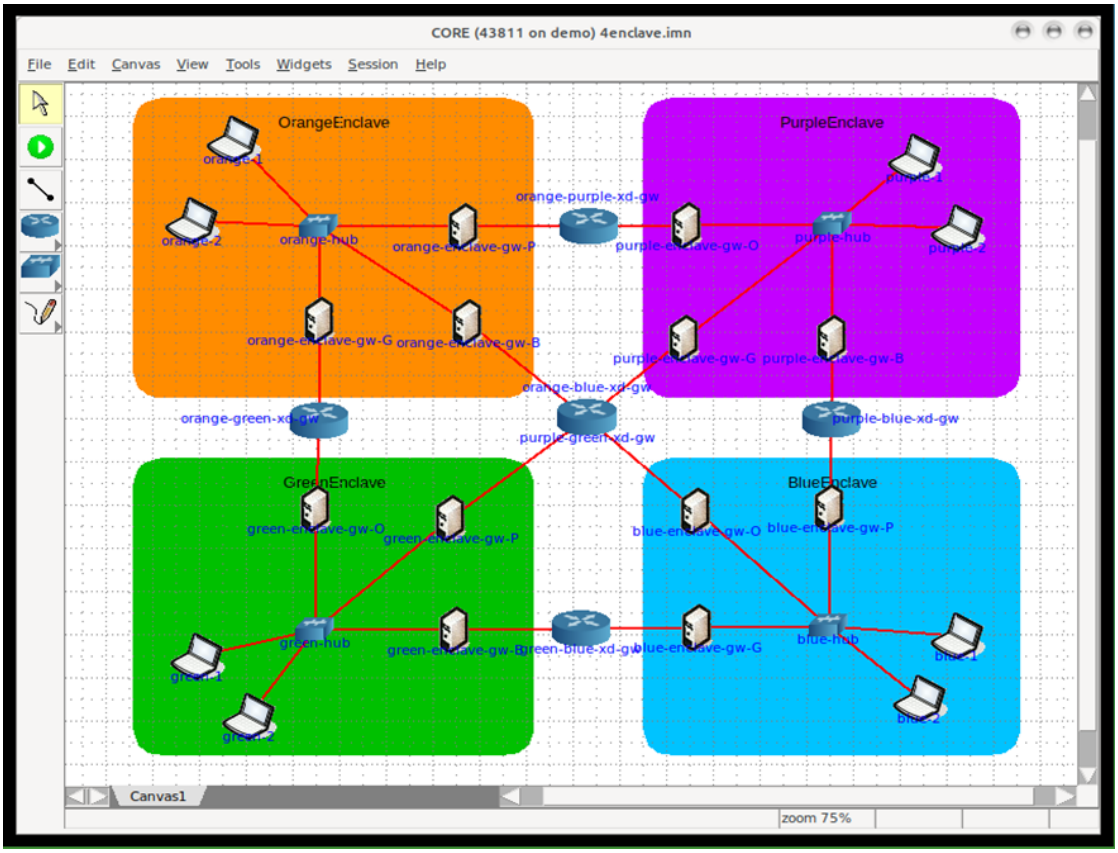


Figure 8: EMU GUI View, 4-enclave

Upon running the emulator, several enclaves (indicated by color) are instantiated per the configuration file. Within each enclave are two types of nodes:

1. enclave-gateway node (e.g., orange-enclave-gw-P) which is co-located with emulated cross-domain hardware used for cross-domain communication to a peer enclave, and
2. a local enclave host (e.g., orange-1) within the enclave but without emulated GAPS hardware.

Enclave gateways are named using the following convention: `<local color>-enclave-gw-<first letter of peer enclave in capital letter>`. Between enclave gateways are cross-domain gateway nodes `<color1>-<color2>-xd-gw`. The cross-domain gateways facilitate bump-in-the-wire guard configuration described in subsequent sections. The Emulator has been tested with up to 4 enclaves. Note the enclave limit is attributed to the processing capacity of the underlying machine running the emulator (there is no fundamental limit to number of nodes and enclaves otherwise). A node can be accessed by double clicking its icon in the GUI to obtain a terminal to the node.

Each enclave-gateway node runs an instance of QEMU to model the node. The QEMU is configured using socat and netcat such that there is a device (`/dev/vcom`) with with the node reads/writes to communicate cross domain. Data written to `/dev/vcom` is passed to the corresponding xd-gw node which either passes the data through or applies redaction (see guard discussion below). The data is then available for reading at the remote enclave-gw QEMU instance upon reading from /dev/vcom. This configuration emulates reading/writing to the guard in the real deployment - no IP-based communication is occurring between the applications (even though under the hood the emulator uses IP transport to move the data). From the applications' perspective they are reading/writing directly to the guard device driver. Note that when double-clicking the enclave-gw node, you are immediately within the QEMU [13] instance (rather than the CORE [12] BSD container wrapping it).

3.7.2 Invoking the emulator

To start the emulator (stand-alone), cd into the emu directory and run ./start [scenario name] where scenario name is that of a directory name in emu/config. Within a scenario configuration are three files:

- `settings.json`: basic settings that typically do not need to be modified. `instdir` should be the absolute path of the parent to which emu is located, which is `/opt/closure/` by default.
- `enclaves.json`: the scenario specification that describes the nodes, connectivity, and attributes
- `layout.json`: the graphical layout of the nodes as they will appear on the emulator GUI

The emulator comes with configuration for 2,3, and 4 enclave scenarios which can be used as a basis for most projects.

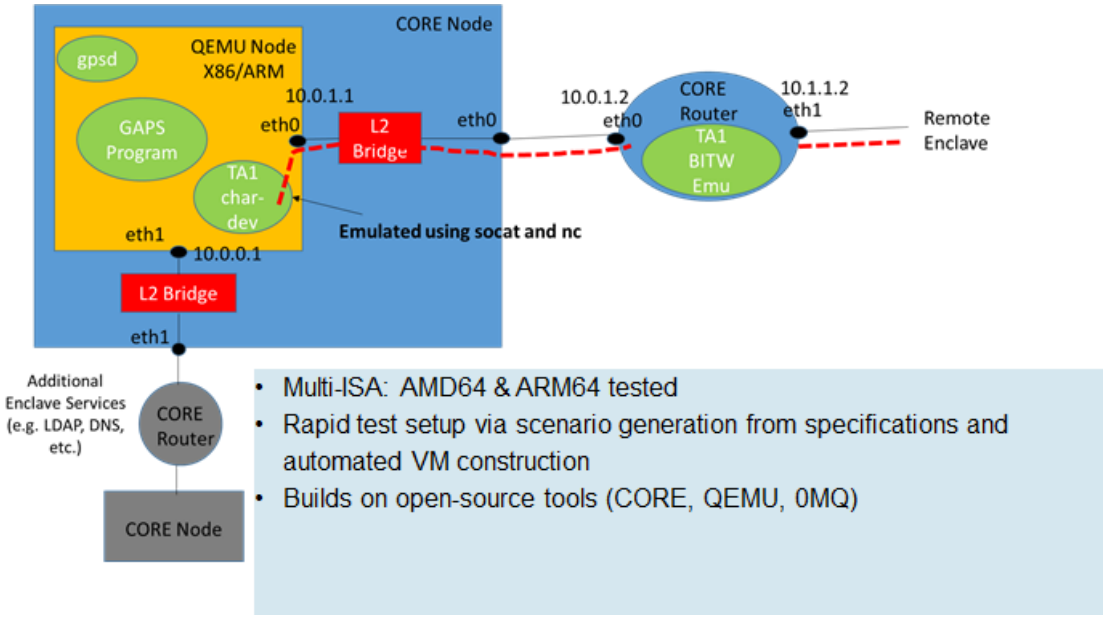


Figure 9: EMU Architecture

Upon running a scenario, perform a basic testing of the cross-domain communication pathway. Use `echo` and `cat` to test reading/writing to the device. On the orange-enclave-gw-P, run `cat /dev/vcom` and on purple-enclave-gw-O, run `echo "abc" > /dev/vcom`. You will see "abc" appear on the terminal of orange-enclave-gw-P.

3.7.3 Building QEMU images for Different Architectures

EMU uses QEMU instances to represent enclave gateways, the nodes designated for cross-domain transactions via a character device to the cross-domain device hardware (denoted by SDH within the GAPS program). This allows us to model multi-domain, multi-ISA environments on which the partitioned software will execute. As a prerequisite to executing the emulator, it is necessary to build clean VM instances (referred to as the "golden images") from which EMU will generate runtime snapshots per experiment. The snapshots allow EMU to quickly spawn clean VM instances for each experiment as well as support multiple experiments in parallel without interfering among users.

VM images can be automatically built using `qemu-build-vm-images.sh`. The script fetches the kernel, builds and minimally configures the VM disk images, and saves a golden copy of the kernels and images.

```
cd scripts/qemu
./qemu-build-vm-images.sh -h
# Usage: ./qemu-build-vm-images.sh [ -h ] [ -p ] [ -c ] \
```

```
#           [ -a QARCH ] [ -d UDIST ] [-s SIZE ] [-k KDIST ] [-o OUTDIR]
# -h        Help
# -p        Install pre-requisites on build server
# -c        Intall NRL CORE on build server
# -a QARCH  Architecture [arm64(default), amd64]
# -d UDIST  Ubuntu distro [focal(default)]
# -s SIZE   Image size [20G(default),<any>]
# -k KDIST  Ubuntu distro for kernel [xenial(default),<any>]
# -o OUTDIR Directory to output images [./build(default)]
```

We recommend storing the built images in a common directory accessible to all users (this README assumes that directory is `/IMAGES`). Ensure sudo group is allowed to work without passwords, otherwise expect scripting to fail on sudo attempts. Note: Pre-built VMs and kernels available under assets in EMU releases. Consider downloading and placing in your image directory to skip the VM build process.

If building your own images, create a virgin image for each architecture for the supported distro (currently eoan):

```
# AMD64
./qemu-build-vm-images.sh -a amd64 -d focal -k focal -s 20G -o /IMAGES
# ARM64
./qemu-build-vm-images.sh -a arm64 -d focal -k xenial -s 20G -o /IMAGES
```

This will fetch the kernel (e.g., linux-kernel-amd64-eoan), initrd (linux-initrd-amd64-eoan.gz), and build the virgin qemu vm image (e.g., ubuntu-amd64-eoan-qemu.qcow2.virgin) using debootstrap.

Now configure the virgin image to make it usable generally with user networking support (allows host-based NAT-ted access to Internet):

```
# AMD64
./qemu-build-vm-images.sh -a amd64 -d focal -k focal -s 20G -o /IMAGES -u
# ARM64
./qemu-build-vm-images.sh -a arm64 -d focal -k xenial -s 20G -o /IMAGES -u
```

You should find the golden copy (e.g., ubuntu-amd64-eoan-qemu.qcow2) created in the directory specified by the `-o` argument (e.g. `/IMAGES`). Note that the Emulator Configuration `settings.json` file requires you to specify the images directory if not using `/IMAGES`.

3.7.4 Guard Models

EMU supports both Bump-In-The-Wire (BITW) and Book-Ends (BE) deployments. In BITW configuration, redaction occurs on the xd-gw node. A 'flowspec' can be loaded – essentially a python program that performs the redaction function on data passing through the node. In BE configuration, the 'flowspec' is invoked at the enclave-gateway before releasing it to the xd-gw (which is merely passthrough in this case). Note the 'flowspec' is a future feature and not general purpose at this time.

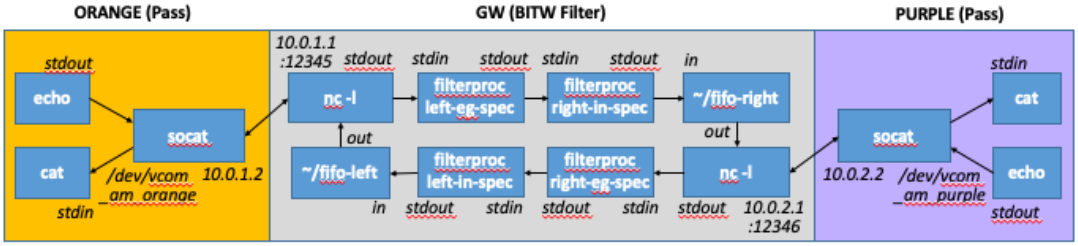


Figure 10: Bump-In-The-Wire (BITW) Plumbing

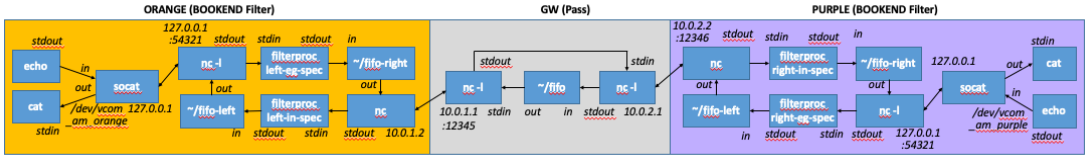


Figure 11: Bookends (BE) Plumbing

3.7.5 Scenario Generation: Various Input and Generated Configuration Files

To generate a scenario, create a directory within emu/config and name it accordingly. Then copy the template configuration files from 2enclave, 3enclave, or 4enclave (depending on which scenario closely matches your intended topology).

Modify your enclaves.json as needed. For example, you may want to adjust the CPU architecture or toggle Bookends (BE) vs Bump-in-the-Wire (BITW).

3.7.5.1 Selecting the ISA for Enclave Gateways/Cross-Domain Hosts (xdhost)

Consider the following snippet from `config/2enclave/enclaves.json`:

```
"hostname": "orange-enclave-gw-P",
"hwconf":{"arch": "amd64"},
"swconf":{"os": "ubuntu", "distro": "focal", "kernel": "focal",
```

To change orange-enclave-gw-P to use ARM64, modify the configuration as follows:

```
"hostname": "orange-enclave-gw-P",
"hwconf":{"arch": "arm64"},
"swconf":{"os": "ubuntu", "distro": "focal", "kernel": "xenial",
```

EMU has been tested using AMD64(eoan) and ARM64(xenial) images. Other architecture/OS instances can be built by following the above provisioning steps, but has not been tested.

3.7.5.2 Selecting the SDH Model for Cross-Domain Links (xdlink)

EMU supports Bookends (BKND) and Bump-In-The-Wire (BITW) SDH deployments. Selection of the model is specified in the `xdlink` section of enclaves.json:

```
"xdlink":
  [
    { "model":  "BITW",
      "left":   {"f": "orange-enclave-gw-P", "t":"orange-purple-xd-gw",
                 "egress":   {"filterspec": "left-egress-spec",
                  ↪  "bandwidth":"100000000", "delay": 0},
                 "ingress":  {"filterspec": "left-ingress-spec",
                  ↪  "bandwidth":"100000000", "delay": 0}},
      "right":  {"f": "orange-purple-xd-gw", "t":"purple-enclave-gw-0",
                 "egress":   {"filterspec": "right-egress-spec",
                  ↪  "bandwidth":"100000000", "delay": 0},
                 "ingress":   {"filterspec": "right-ingress-spec",
                  ↪  "bandwidth":"100000000", "delay": 0}}
    }
  ]
```

The above specifies a BITW model. Simply change to the following to use BKND:

```
"model": "BKND"
```

3.7.6 Running Scenarios

Prior to running the emulator, follow the instructions above to generate the QEMU Golden Images.

In general, the Emulator is best run using the EMULATE build target of a project. This automates several steps such as constructing HAL configuration, preparing application tarballs, installing application dependencies etc. See the tasks.json and Makefile.mbig for respective projects. Furthermore the QEMU and CORE dependencies are all preinstalled in the CLOSURE container.

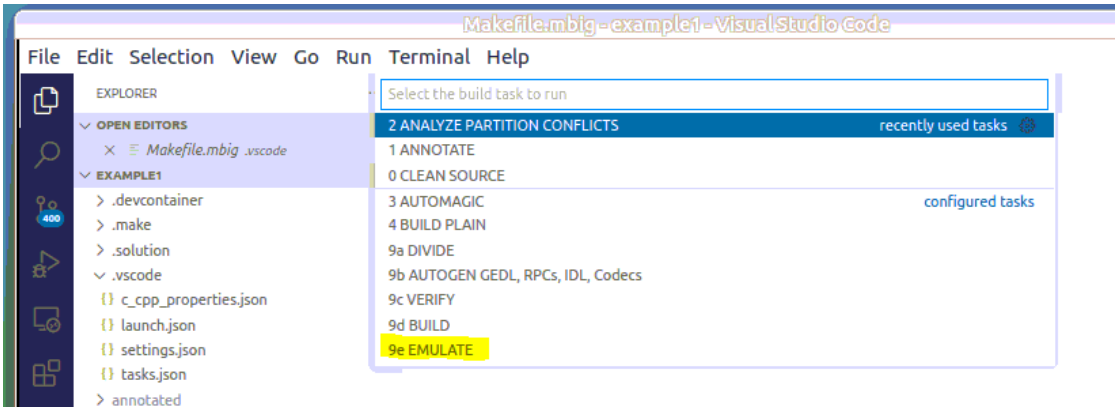


Figure 12: Running Emulator from CVI

Additionally, the emulator can be run stand-alone outside the container. Change directory to the emulator install location (emu directory) and run ./start [scenario name], e.g., ./start.sh 2enclave. The standard out will provide feedback on the emulator execution and print errors if any.

Note that QEMU images are stored on your local machines and mounted to the container, the VMs persist to simplify startup in subsequent iterations. If necessary, the snapshots can be removed (cd .snapshots; rm *) for clean start up.

3.7.7 Deploying on objective hardware

Additional manual steps are required when deploying the application and associated cross-domain tools (e.g., HAL) on target objective hardware.

3.7.7.1 Generating HAL configuration from `xdconf.ini` and `device.json` The HAL configuration files are generated as described in hal conf.

3.7.7.2 Copy and build each partition on respective hardware For a given application, copy the appropriate `partitioned/multithreaded/<color>` directory to the target and proceed to build with gcc/clang on the target. Alternatively cross-compile frm a development machine. The project will include a Makefile for the application (developed during the project construction process).

3.7.7.3 Install xdcomms and HAL on Enclaves On the target setup, copy the HAL repository or checkout via git. CD into hal and run `make clean;make` to build HAL and the xdcomms static and shared libraries (see ./api and ./daemon directories for

binaries). When running HAL and CLOSURE partitioned applications utilizing HAL, these libraries need to be available in the `LD_LIBRARY_PATH` environment variable.

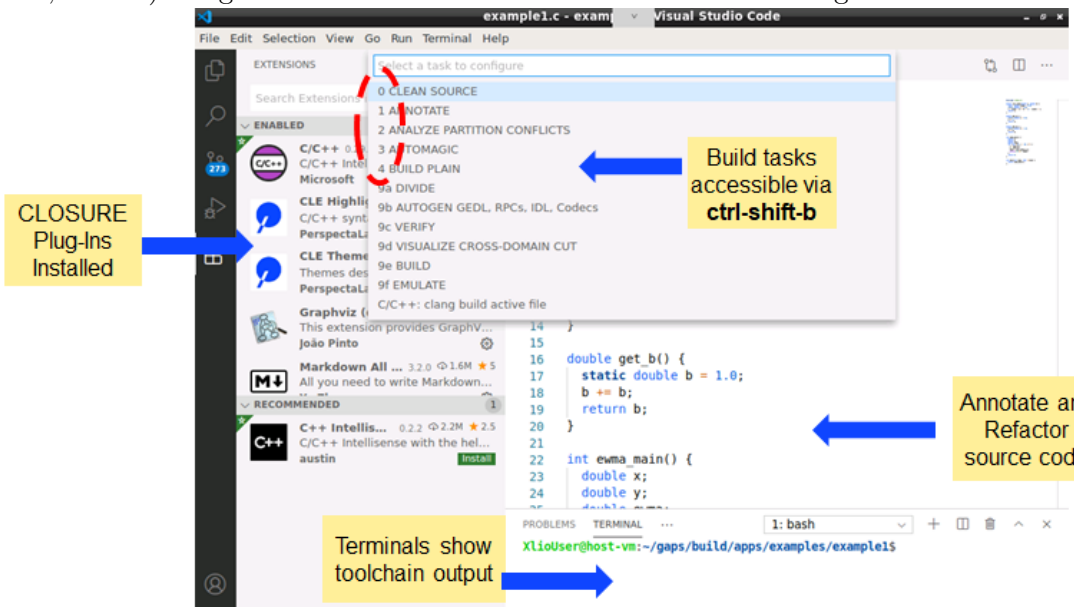
3.7.7.4 Configure and start HAL for each Enclave HAL is executed by running the following:

```
cd hal
daemon/hal <configuration file>
```

3.7.7.5 Run the Application Once the application is partitioned, compiled, and HAL is running, the target application may be run on each `enlcave.ss`

3.8 CLOSURE Visual Interface (CVI)

The CLOSURE Visual Interface (CVI) is the editor supporting IDE tools built on top of VSCode to support development of cross-domain applications. The CVI includes i) plugins for syntax highlighting and partitioning diagnostics, ii) Makefiles to invoke CLOSURE toolchain components throughout the workflow, iii) Build Targets making the toolchain "push-button", and iv) integration with the Emulator for end-to-end testing prior to deployment.



3.8.1 CVI Workflow

During the development lifecycle, the user follows the following CVI steps to build the project:

1. ANNOTATE - plain code (original source) is copied to a ./annotated directory. developer applies annotations to source here
2. ANALYZER PARTITION CONFLICTS - conflict analyzer assesses the annotation for feasibility. upon success a topology is generated
3. AUTOMAGIC - CVI runs through the automated code generation portion including code dividing, gedl, RPC generation, serialization, HAL configuration, and deploy in emulation for test and evaluation.

3.8.2 Startup and usage

3.8.2.1 Quick start

Under the `cle-extension` subdirectory under `cvi` repo

```
npm install
```

Press `f5` to build and open the extension in a new window. A `*.c` or `*.cpp` file must be opened to activate the extension.

The plugin can be built and installed using `vsce`

```
npm install -g vsce
# .vsix generated
vsce package
```

Then you can install the extension in vscode under the extensions window. There you can click the `...` menu and a dropdown will appear containing `Install from VSIX`. From there you can give vscode the `.vsix` that was built.

Similarly, the `vsce package` command can be repeated to build the `cle-highlighter` and `cle-themes` extensions within each of their respective subdirectories in CVI.

3.8.3 CLE plugin and language server

The language server is an intermediary between vscode and the conflict analyzer which handles translating diagnostics from the conflict analyzer into a standardized form defined by the Language Server Protocol (LSP).

Since there is no significant coupling between the client and server, the language server could be run separately and could be used to supply diagnostics for different IDEs such as neovim, emacs, eclipse, atom, etc. However, in order to fully support the CLE plugin, the language client would need to support the `highlight` and `unhighlight` notifications, which are custom commands to highlight ranges of code with colors, to visually see the assignments of functions to enclaves.

When in use, the CLE plugin detects CLE syntax and partitioning errors and provides diagnostics to the developer. The feedback may include tips in plain english or minimum set of unstatisfied constraints in MUS MiniZinc form.

Figure 13: CLE Plugin providing diagnostics

3.8.4 Setting up a new project (Makefiles and VScode tasks)

In order to set up a new project, the developer must customize all these files for their project.

Use example1 as a template for setting up a new project. Copy the following sub-directories into your new project folder, maintaining the directory structure found in example1:

- .make
  - closure_env.sh: environment variable settings for the project
  - conflicts.make: invokes the conflict analyzer during ANALYZE PARTITION CONFLICTS
  - divvied.make: invokes the divider to divide code into separate code-bases by enclave
  - ect.make: invokes the program equivalence checker to ensure unpartitioned and partitioned functionality remains unchanged

  - example.make: the Makefile that builds the application source. This file will be unique for each project and is invoked during BUILD stage
  - gedl.make: invokes the cross-domain cut analysis and produces detailed description of such data types
  - join.sh: tool facilitating HAL configuration
  - mbig.make: supports multi-target compilation and packaging of application
  - verifier.make: deprecated, Phase 1 verification

- .vscode
  - tasks.json: includes the definitions and commands associated with the build targets.
  - debs.list(optional): list of Ubuntu package dependencies to be installed prior to running the target application. Emulator will automatically install these during VM customization
  - pips.json(optional): list of Python3 dependencies to be installed prior to running the target application. Emulator will automatically install these during VM customization
  - clean.list(optional): Alternative means for downloading project dependencies (some dependencies cannot be retrieved during customization). List of packages to be downloaded, saving the packages in .dependencies. Useful when deploying outside of Emulator or if specific packages are not installable during VM customization stage via debs.list
  - c_cpp_properties.json, launch.json, settings.json: standard VSCode files

- .devcontainer
  - devcontainer.json: specifies the CLOSUREDEV container for use (see VSCode remote-containers standard plugin). This file can be customized for more complex projects requiring specific packages.

Any application specific dependencies must be included in a dockerfile that extends `gapsclosure/closuredev:latest`. This will be used during analysis and partitioning using the CLOSURE toolchain. For including the same dependencies for testing in the emulator, they must also be listed in `debs.json` as described above.

3.9 Partitioning of Message-Flow Model

CLOSURE supports the partitioning of models of message-flow based applications, that is, applications that have already been partitioned into components, but use a messaging service such as ActiveMQ to exchange messages via publish/subscribe blackboard.

The process begins with the developer specifying a topological model of components and messages along with the schema of the messages. The developer also specifies cross-domain concerns using CLE annotations. We describe the subsequent steps involved including the model specification and model format details, analysis and partitioning of

the annotated model, and auto generation of CLE-annotated C code from the partitioned model. The generated C code is then processed using the CLOSURE toolchain for C as described earlier.

The model-driven approach is part of a larger workflow shown in the figure. Rapid capture of message flow data through use of sniffer, wildcard-subscriber, or other instrumentation provides listing of message types and field contents (types can be inferred and tweaked by developer if need be). A design-tool can then be used to annotate the message flows, structures, and cross-domain security intent in language-agnostic fashion. Automated generation of CLE annotated XDCC in C language is performed. XDCC program isolates per-message code paths facilitating annotations and compliant cross-domain partitioning, covering a large class of message-based cross-domain applications. We consider this technique relevant and transitionable to RHEL AMQ Interconnect for which it could enable cross-domain message routing.

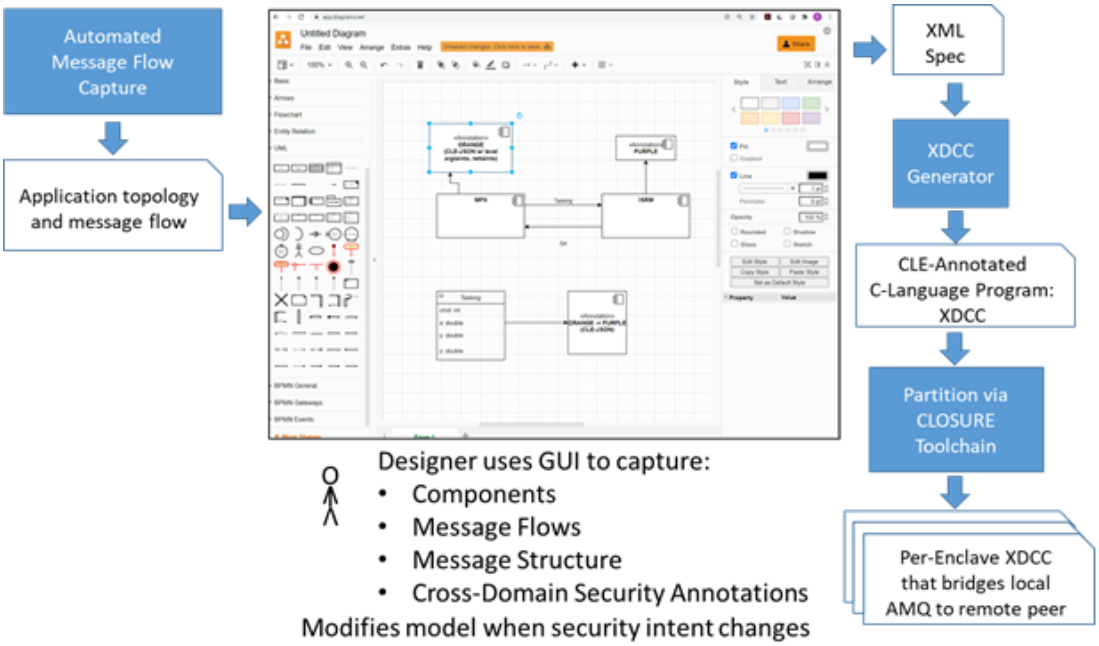


Figure 14: Concept for Design-Level Workflow of Message-Based Applications

An application of this type was evaluated during the EoP1 exercises. CLOSURE enables message-flow partitioning by generating a cross-domain communication component (XDCC) from a message flow specification. Using the specification, CLOSURE tools generate a C program that subscribes to those messages that will be cross-domain and facilitates their transfer over the guard. When a cross-domain message is received on the remote XDCC, the message is reconstructed and published to ActiveMQ for consumption by the remote enclave components. See partitioning of message-flow model for more details on how the specification is processed.

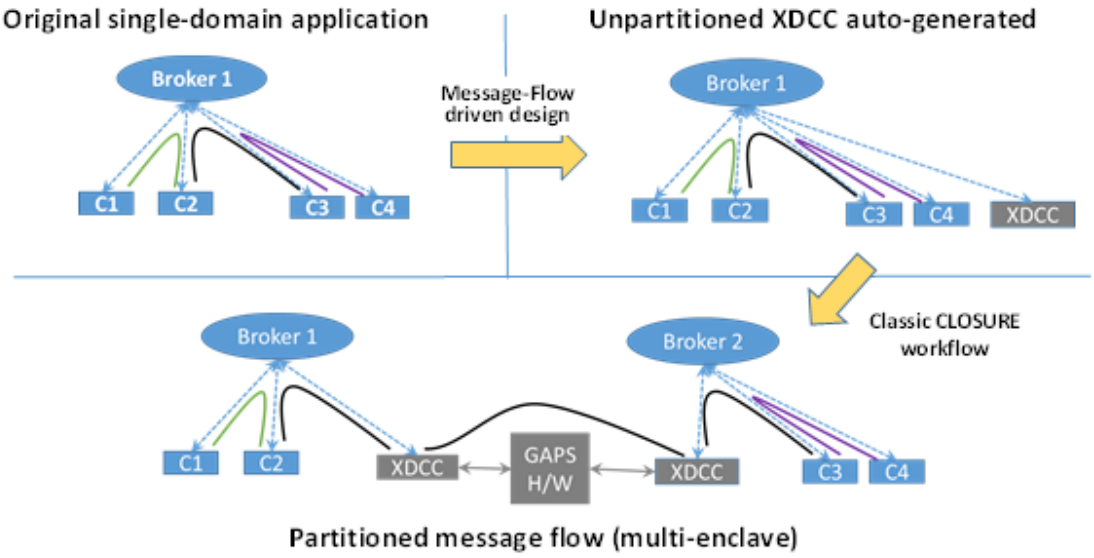


Figure 15: XDCC concept

3.9.1 Specification Format

A snippet of EoP1 Message Specification is reproduced below. The structure of the specification is as follows:

- **comments**: for human readable purposes, the component, and message types are listed with their unique numbers for easy identification in rest of the file. FlowIDs are formed by combining src/dst component identifiers with message id
- **topology**: lists the application components, CLE label for the component, and the inflows and outflows.
- **flows**: listing of all flows, message that traverses the flow, and associated label
- **messages**: listing of all message types and associated schema
- **cles**: definition of all CLE labels in the model

```
{
    "comment": [
      "Case 1: TA3-proposed PEST",
      "Components [MPU:1,MPX:2,ISRM:3,RDR:5] in orange:1",
      "Components [EOIR:4, External:6] in green:2",
      "Messages: ",
      "component_heartbeats:01",
      "updateMissionPlan:02",
      "pnt:03",
      "requestISRMDetections:04",
      "recieveISRMDetections:05",
```

```
  "requestEOIRDetections:06",
  "recieveEOIRDetections:07",
  "requestRDRDetections:08",
  "recieveRDRDetections:09",
  "groundMovers:10",
  "FlowID (encodes mux and typ): PQMM component P->Q, message MM"
],
"topology": [
    {
        "component": "MPU",
        "label": "MPU_CLE_LABEL",
        "inFlows":  [ 2101, 3101, 4101, 5101 ],
        "outFlows": [ 1201, 1301, 1401, 1501,
                      1202, 1302, 1402, 1502 ]
    }

],
"flows": [
    {"flowId":
     ↪ 1201,"message":"component_heartbeats","label":"ALLOW_ORANGE_ORANGE"},
    {"flowId":
     ↪ 1301,"message":"component_heartbeats","label":"ALLOW_ORANGE_ORANGE"},
    {"flowId":
     ↪ 1401,"message":"component_heartbeats","label":"ALLOW_ORANGE_GREEN"}
],
"messages": [
    {
        "name": "component_heartbeats",
        "topic": true,
        "schemaType": "JSONSchema",
        "schemaFile": "schema/component_heartbeats_schema.json"
    },
    {
        "name": "updateMissionPlan",
        "topic": true,
        "schemaType": "JSONSchema",
        "schemaFile": "schema/updateMissionPlan_schema.json"
    }

],
"cles": [
    {
        "cle-label": "MPU_CLE_LABEL",
        "cle-json": {
```

```
        "level": "orange",
        "cdf": [
            {
                "remotelevel": "green",
                "direction": "egress",
                "guarddirective": {
                    "operation": "allow"
                },
                "argtaints": [
                    [ "ALLOW_ORANGE_ORANGE" ],
                    [ "ALLOW_ORANGE_ORANGE" ],
                    [ "ALLOW_GREEN_ORANGE" ],
                    [ "ALLOW_ORANGE_ORANGE" ],
                    [ "ALLOW_ORANGE_ORANGE" ],
                    [ "ALLOW_ORANGE_ORANGE" ],
                    [ "ALLOW_ORANGE_GREEN" ],
                    [ "ALLOW_ORANGE_ORANGE" ],
                    [ "ALLOW_ORANGE_ORANGE" ],
                    [ "ALLOW_ORANGE_ORANGE" ],
                    [ "ALLOW_ORANGE_GREEN" ],
                    [ "ALLOW_ORANGE_ORANGE" ]
                ],
                "codtaints": [
                ],
                "rettaints": [
                ]
            }
        ]
    }
}
{
    "cle-label": "ALLOW_ORANGE_ORANGE",
    "cle-json": {
       "level":"orange",
       "cdf":[
          {
             "remotelevel":"orange",
             "direction":"egress",
             "guarddirective":{
                "operation":"allow",
                "oneway":true
             }
          }
       ]
```

```
            }
        }
    ]
}
```

3.9.2 Analyzing the Specification

The Flow Solver is a z3-backed solver/verifier for GAPS-CLOSURE application design specifications. It verifies that specifications are self-consistent and can find satisfying values for fields which are omitted from the specification, such as component levels or flow labels. It automatically derives and outputs a minimal cross-domain message flow policy for the specification.

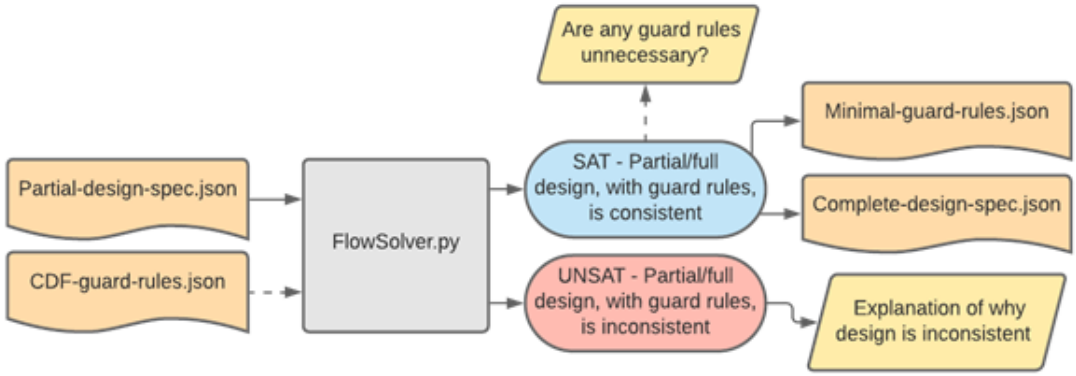


Figure 16: Flow Solver Workflow

If the command-line option is specified, it will also constrain the solution to a provided cross-domain message policy, and report on whether the policy is overly permissive.

If there is a problem with the provided specification such that it is not consistent with itself or the provided policy, the solver will output a simple English explanation of what went wrong.

3.9.3 Assumptions

The solver currently makes a number of simplifying assumptions about the specification, as it is currently in development. We document those assumptions here.

The simplified specification, explained below, has the following form:

```
Component
    id:          Int
```

```
    inflows:     [Flow]
    outflows:    [Flow]
    argtaints:   [FlowLabel]
    level:       Enum("orange" | "green")
    remotelevel: Enum("orange" | "green")

Flow:
    id:        Int
    msg:       Int
    label:     FlowLabel

FlowLabel:
    id:        Int
    local:     Enum("orange" | "green")
    remote:    Enum("orange" | "green")
```

CLE labels are split between labels for components and labels for flows - it is assumed that no CLE label is used for both a flow and a component.

Further, the solver assumes that each component has one unique CLE label - thus, the label information is merged with the component information, and the only CLE labels the solver tracks explicitly are flow labels.

Message data beyond the name (`topic`, `schemaType`, and `schemaFile`) is ignored, so the solver does not track message objects, only the message string used by each flow.

A number of fields are omitted: `codtaints`, `rettaints`, `direction`, `oneway`, and `guarddirective`. All directions are assumed to be `egress`, and all operations are assumed to be `allow`.

It is assumed that the `cdf` field in each CLE label is a singleton list, and that each argtaint is a singleton list. Finally, the solver assumes that there are exactly two levels, `orange` and `green`.

Internally, the solver passes integer IDs to z3 and converts between IDs and component/flow/label/message names, because z3 works far more quickly with integers. Thus each component, flow, flow label, and message has an ID. The strings "orange" and "green" are also tied to integer IDs.

These assumptions will be gradually phased out as the solver matures.

3.9.4 Constraint formulas

Relationships between Components, Flows, and FlowLabels, and their fields, are modeled as integers and functions mapping integers to other integers. The solver gets a

partial interpretation of each function from the provided spec/rules, and uses the constraints to assign a full interpretation to each mapping function.

Integer IDs are constrained to valid values in context. For example, the `label` function maps a flow to its corresponding label, so `label`'s domain is technically `Int` x `Int` (`id -> id`), but it is bounded such that the input must be an integer which corresponds to a flow ID, and the output must be an integer which corresponds to a label ID.

Beyond these boundary conditions, the main constraints are given below:

3.9.4.1 Inflows and outflows must match flow levels and component levels

`f` is a flow, `c` is a component, `i` is an index.

1. `Forall f c i, c.outflows[i] == f => c.level == f.label.local`
2. `Forall f c i, c.inflows[i] == f => c.level == f.label.remote`

3.9.4.2 Argtaints must match component inflows, outflows, level, and remotelevel

`c` is a component, `i` is an index.

1. `Forall c i, c.inflows[i].label == c.argtaints[i]`
2. `Forall c i, c.outflows[i].label == c.argtaints[len(c.inflows) + i]`
3. `Forall c i, i < len(c.inflows) => c.argtaints[i].remote == c.level && (c.argtaints[i].local == c.level || c.argtaints[i].local == c.remotelevel)`
4. `Forall c i, i >= len(c.inflows) => c.argtaints[i].local == c.level && (c.argtaints[i].remote == c.level || c.argtaints[i].remote == c.remotelevel)`

3.9.4.3 Deriving a cross-domain message flow policy

A function `cdf_allowed` is used to track whether a cross-domain message flow is a.) needed by the application and b.) allowed by the given policy (if a policy was given). During solving, if the constraints imply a CDF which was already denied by the provided policy, `cdf_allowed` is unsatisfiable. After solving, `cdf_allowed` is queried for each message and cross-domain flow to determine what policy the application needs.

The function is defined in the solver as follows:

```
cdf_allowed(MessageID m, ColorID c1, ColorID c2) == Exists (Flow f),
f.label.local == c1 && f.label.remote == c2 && f.msg == m
```

3.9.5 Testing the Model Partitioner

Several example application design specs, and the results given by the solver, are provided in the `examples` folder, with descriptions in the respective `README.md` files.

This solver requires z3. To install z3 for python, run:

```
pip3 install z3-solver
```

To use the solver, in the `flowspec` directory run:

```
python3 FlowSolver.py examples/valid/case1.json
```

To add a cross-domain message policy, use the `--rules` option:

```
python3 FlowSolver.py examples/valid/case1.json --rules examples/rules/case1.json
```

To see all options, use:

```
python3 FlowSolver.py -h
```

3.9.6 Auto-generating Annotated C code

From the output of the model partitioner (`FlowSolver.py`), the `xdcc_gen` tool generates CLE-annotated C programs on which the rest of the CLOSURE toolchain for the C language can be applied to get partitioned binary executables.

The `xdcc_gen` tool produces two programs per pair of enclaves, one each per message flow direction. Each program when partitioned acts in a "pitcher-catcher" pair, with the pitcher subscribing to the ActiveMQ broker on its side to message types that must be sent to the other side, and generates an RPC invocation for each such message instance. The catcher receives the message and sends it to the local message broker. In order to prevent loops (due to the same message type being generated both locally as well as being received from the remote side), the catcher adds a field to mark the message as received from remote side. The RPC is one-way and no response is expected.

The usage summary of 'xdcc_gen' is provided below.

```
$ ./xdcc_gen --help
./xdcc_gen
  -e/--egress    egress output directory
  -i/--ingress   ingress output directory
  -k/--echo      echo output directory
  -f/--design    design JSON file
  -n/--enclave   enclave (e.g. purple)
  -c/--config    configuration file
  -h/--help      print this message and exit
```

For more information about its usage, see one of the included EoP1 examples.

3.10 Example applications

3.10.1 Pedagogical Examples (examples 1-3)

The pedagogical examples are available for basic understanding of the CLOSURE workflow, annotations, and testing via emulation. Each example uses the same plain source, however, the **partitioning objectives** differ for each:

- Example1
  - variable a in get_a() is in ORANGE and can be shared with PURPLE
  - variable b in get_b() is in PURPLE and cannot be shared
  - calculated ewma must be available on PURPLE side (for printing)
- Example2
  - variable a in get_a() is in ORANGE and can be shared with PURPLE
  - Variable b in get_b() is in PURPLE and cannot be shared
  - calculated ewma must be available on ORANGE side (for printing)
- Example3
  - variable a in get_a() is in ORANGE and cannot be shared
  - variable b in get_b() is in ORANGE and cannot be shared
  - ewma must therefore be computed on ORANGE; EWMA is shareable to PURPLE
  - calculated ewma must be available on PURPLE side (for printing)

Example 1 is an exercise in applying annotations, example 2 and example 3 are exercises in applying annotations with code refactoring.

Example1 Solution

Example2 Solution

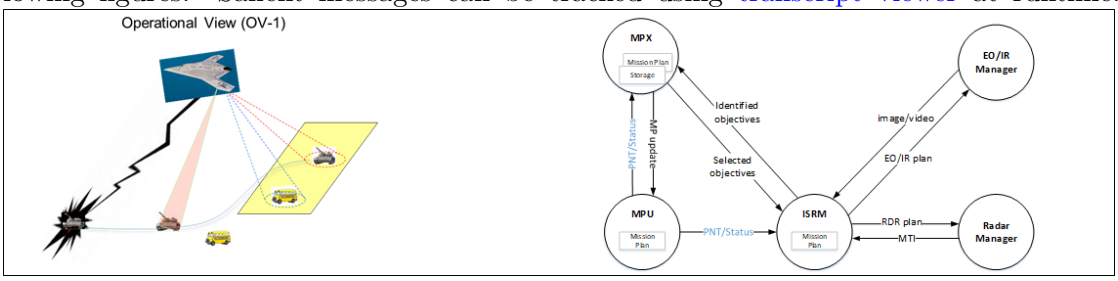
Example3 Solution

3.10.2 EoP1 Applications

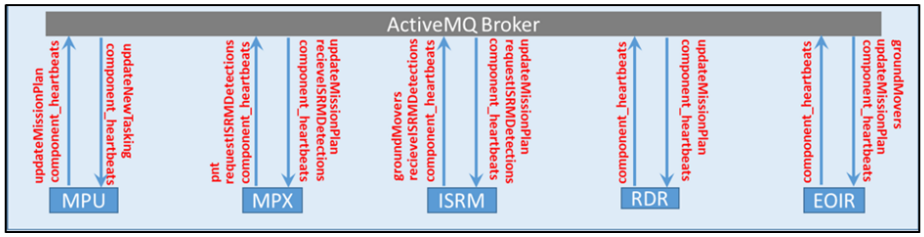
The EoP1 application is a toy application, loosely based on Find, Fix, Track, Target, Engage, Assess (F2T2EA) missions. The application source was provided by the TA4 Integration Partners. It consists of a pre-partitioned C++ message-based application using ActiveMQ to send a variety of messages to coordinate the simulated mission. The components include:

- MPU: Mission Planner
- MPX: Mission Executor
- ISRM: Intelligence Surveillance Recon Manager
- RDR: Radar Sensor
- EOIR: Video Sensor
- External: GPS simulator

High-level architecture of the application and message flows shown in the following figures. Salient messages can be tracked using transcript viewer at runtime.



- Modeled after Find Fix Track Target Engage Assess (F2T2EA) missions
- Written in C++ using ActiveMQ and JSON



- **start**:
  - all components except MPU send component_heatbeats (which are shared cross-domain)
  - we will skip some details: e.g., pnt (from MPX) and groundMovers (from External)
- **sync**:
  - when all component_heartbeats are received MPU issues special heartbeat "All"
- **mission plan:**
  - MPU sends updateMissionPlan to all
- **find phase – request and receive detections:**
  - MPX sends requestISRMDetections to ISRM for find phase
  - ISRM sends requestEOIRDetections and requestRDRDetections to EOIR and RDR respectively
  - EOIR sends recieveEOIRDetections to ISRM
  - RDR sends recieveRDRDetections to ISRM
  - ISRM collates detections and sends recieveISRMDetections to MPX
- **fix phase – request and receive detections:**
  - similar sequence repeats as find phase
- **end**:
  - MA continues running, but no interesting activity after the fix phase

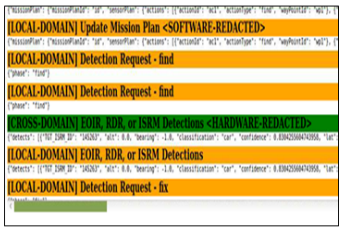


Figure 17: Salient Messages

Three cases were evaluated during the end-of-phase demonstration:

- Case 1: Normative policy provided by TA4
  - MPU, MPX, ISRM, RDR (orange)
  - EOIR, External (green)
- Case 2: Coalition partner concept (e.g., planners on orange, sensors on green)
  - MPU, MPX (orange)
  - ISRM, RDR, EOIR, External (green)
- Case 3: Incorporates manual code partitioning, ISRM functionality manually divided between planning and sensor reading w/ redaction)
  - MPU, MPX, ISRM (orange)
  - ISRMshadow, EOIR, RDR, External (green)

4 Limitations and Future Work

4.1 Current Limitations and C Language Coverage

The CLOSURE toolchain supports most of the c99 standard. Our solution is based on LLVM, and in particular, the Program Dependency Graph abstraction [8] of a the LLVM IR representation of a C program. The constraint model is straight-forward and can be studied for more details on coverage.

Some C language and pre-processor features not currently supported include: (i) module static functions for which the compiler creates synthetic names are not handled, (ii) inlined functions, macro generated functions and conditionally compiled code that are not visible in the LLVM IR and are not handled, (iii) functions to be wrapped in cross-domain RPC must have arguments that are primitive types or fixed size arrays of primitive types.

In our current solution, every global variable and function must be assigned to a separate enclave. Any functions called from multiple enclaves must be in an external library (and not subject to program analysis), and currently we do not provide a sandboxing mechanism for external library functions. Our program divider has limited awareness of C language syntax. The solution supports at most one enclave at each security level.

Currently the constraint model is a single pass that requires functions to be called cross-domain to be annotated by the developer. It may be desirable to do constraint solving in two passes: the first pass can allow the locus of the cut to be moved for optimization, and the second pass can check that all functions ultimately involved in cross-domain cuts have been inspected and correctly annotated by the developer.

The CLOSURE C toolchain has been exhaustively tested with 2-enclave scenarios, but not with scenarios involving more than 2 enclaves. The largest program we have analyzed is about 25KLoC and it takes several minutes to analyze.

CLOSURE message-flow toolchain currently supports ActiveMQ-based communication, though the approach is general and can be extended to other messaging middleware if needed.

Subsequent releases may address some of these limitations and add features as discussed in our roadmap below.

4.2 Roadmap for Future Work

We plan to continue to refine and enhance the CLOSURE C toolchain beyond this release. The enhancements will include relaxing known limitations as well as adding new features, and will be prioritized based on the needs of the ongoing DARPA GAPS program. Our current research and development roadmap includes:

1. More complete coverage of the C language along with more extensive testing for language coverage
2. Support for the analysis and partitioning of distributed applications, namely, the analysis of message flows between components and program partitioning of components themselves
3. Support for analysis and partitioning of concurrent (multi-threaded) program
4. Support for handling both cross-domain and high-performance concerns in application partitioning, for example, through the integration of CLOSURE and oneAPI toolchains
5. Support for additional cross-domain communication modalities as well as RPC mechanisms
6. Support for non-Linux platforms included embedded RTOS and bare-metal targets
7. Support for additional languages, in particular, C++ and Java
8. Integration with other formal verification tools and increasing the scope of verification
9. Annotation hints, refactoring guidance, and diagnostics to the developer that are friendlier than what is currently provided based on the output of the constraint solver
10. Enhanced support for the provisioning of cross-domain guards through user-friendly specification of data formats and filter/transform rules, with traceability to application source
11. Scalability to larger programs and increased performance
12. More examples, application use cases, and user stories as they become available

5 Appendices

5.1 CLE JSON example and schema

5.1.1 Example

Below is an example of `cle-json`. From the source code, the preprocessor produces a json with an array of label-json pairs, which are objects with two fields `"cle-label"` and `"cle-json"` for the label name and label definition/json respectively.

```
[
  {
    "cle-label": "PURPLE",
    "cle-json": {
      "level": "purple"
    }
  },
  {
    "cle-label": "ORANGE",
    "cle-json": {
      "level": "orange",
      "cdf": [
        {
          "remotelevel": "purple",
          "direction": "egress",
          "guarddirective": {
            "operation": "allow"
          }
        }
      ]
    }
  }
]
```

5.1.2 Schema

The preprocessor validates cle-json produced from the source code using jsonschema. The schema for cle-json is shown in detail below:

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "$id": "com.perspectalabs.gaps-closure.cle",
    "$comment": "JSON schema for GAPS-Closure CLE json definition",
```

```
"oneOf":[
    {
        "description": "List of CLE entries",
        "type": "array",
        "default": [],
        "items": { "$ref": "#/definitions/cleLabel" }
    },
    {
      "$ref": "#/definitions/rootNode"
    }
],

"definitions": {
    "guarddirectiveOperationTypes": {
        "$comment": "the guarddirective type enum",
        "description": "[ENUM] Guard Directive",
        "enum": [
            "allow",
            "block",
            "redact"
        ]
    },
    "directionTypes": {
        "$comment": "the direction type enum",
        "description": "[ENUM] traffic direction",
        "type": "string",
        "enum": [
            "egress",
            "ingress",
            "bidirectional"
        ]
    },

    "guarddirectiveTypes":{
        "description": "Guard Directive parameters",
        "type": "object",
        "properties": {
            "operation":{
                "$ref": "#/definitions/guarddirectiveOperationTypes"
            },
            "oneway": {
                "description": "Communication only in one
                 ↪  direction",
```

```
            "type": "boolean",
            "default": false

        },
        "gapstag": {
            "description": "Gaps tag to link remote CLE data
             ↪  [mux,sec,type]",
            "type": "array",
            "maxLength": 3,
            "minLength": 3,
            "items":[
                {
                    "type": "number",
                    "minimum": 0,
                    "description": "mux value"
                },
                {
                    "type": "number",
                    "minimum": 0,
                    "description": "sec value"
                },
                {
                    "type": "number",
                    "minimum": 0,
                    "description": "type value"
                }
            ]
        }
    }
},

"argtaintsTypes":{
    "description": "argument taints",
    "type": "array",
    "default": [],
    "uniqueItems": false,
    "items": {
        "description": "Taint levels of each argument",
        "type": "array",
        "default": [],
        "items":{
            "type": "string",
            "description": "CLE Definition Name"
        }
```

```
    }
},

"cdfType": {
    "description": "Cross Domain Flow",
    "type": "object",
    "properties": {
        "remotelevel":{
            "description": "The remote side's Enclave",
            "type": "string"
        },
        "direction":{
            "$ref": "#/definitions/directionTypes"
        },
        "guarddirective":{
            "$comment": "active version guarddirective",
            "$ref": "#/definitions/guarddirectiveTypes"
        },
        "guardhint":{
            "$comment": "deprecated version of guarddirective",
            "$ref": "#/definitions/guarddirectiveTypes"
        },
        "argtaints":{
            "$ref": "#/definitions/argtaintsTypes"
        },
        "codtaints":{
            "description": "Taint level",
            "type": "array",
            "default": [],
            "items":{
                "type": "string",
                "description": "CLE Definition Name"
            }
        },
        "rettaints":{
            "description": "Return level",
            "type": "array",
            "default": [],
            "items":{
                "type": "string",
                "description": "CLE Definition Name"
            }
        },
        "idempotent":{
```

```
            "description": "Idempotent Function",
            "type": "boolean",
            "default": true
        },
        "num_tries":{
            "description": "Num tries",
            "type": "number",
            "default": 5
        },
        "timeout":{
            "description": "Timeout",
            "type": "number",
            "default": 1000
        },
        "pure":{
            "description": "Pure Function",
            "type": "boolean",
            "default": false
        }
    },
    "dependencies": {
        "argtaints": {
            "required": ["argtaints", "codtaints", "rettaints"]
        },
        "codtaints": {
            "required": ["argtaints", "codtaints", "rettaints"]
        },
        "rettaints": {
            "required": ["argtaints", "codtaints", "rettaints"]
        }
    },
    "oneOf":[
        {
            "required": ["remotelevel", "direction",
             ↪  "guarddirective"]
        },
        {
            "required": ["remotelevel", "direction",
             ↪  "guardhint"]
        }
    ]
},

"cleLabel":{
```

```
    "type": "object",
    "required": ["cle-label", "cle-json"],
    "description": "CLE Lable (in full clemap.json)",
    "additionalProperties": false,

    "properties": {
        "cle-label": {
            "description": "Name of the CLE label",
            "type": "string"
        },
        "cle-json":{
            "$ref": "#/definitions/rootNode"
        }
    }
},

"rootNode":{
    "type": "object",
    "required": ["level"],
    "description": "CLE Definition",
    "additionalProperties": false,
    "properties": {
        "$schema":{
            "description": "The cle-schema reference (for
             ↪  standalone json files)",
            "type": "string"
        },
        "$comment":{
            "description": "Optional comment entry",
            "type": "string"
        },
        "level":{
            "description": "The enclave level",
            "type":"string"
        },
        "cdf": {
            "description": "List of cross domain flows",
            "type": "array",
            "uniqueItems": true,
            "default": [],
            "items": { "$ref": "#/definitions/cdfType" }
        }
    }
}
```

```
    }
}
```

5.2 Program Dependency Graph (PDG)

PDG Specification (Version 1.0.2) (4/14/2021)

This specification focuses on the PDG model definition only, the representation syntaxes for portable exchange and visualization will be addressed in a separate document.

Companion files containing C source code and LLVM IR[14] [15] code are included to provide a running example that will be referenced throughout this document. The example.ll is generated by issuing the following command: clang -c -emit-llvm -g example.c

5.2.1 PDG

The Program Dependency Graph (PDG) is a graphical representation of a salient portion of the LLVM IR of a program; it is composed of PDG nodes and PDG edges. We use a TYPE_SUBTYPE naming convention for PDG nodes and PDG edges. We define two properties that we will use below:

- AllDisjoint(...) : The specified sets are disjoint
- Universe(...) = <UNIV> : The union of specified sets is the set <UNIV>

5.2.2 Nodes

5.2.2.1 Node Types Summary The different types/subtypes of PDG nodes are listed below.

```
INST_FUNCALL,
INST_RET,
INST_BR,
INST_OTHER,

VAR_STATICALLOCGLOBALSCOPE,     // C: global variable          LLVM:
 ↪  global with common linkage
VAR_STATICALLOCMODULESCOPE,     // C: static global variable   LLVM:
 ↪  global with internal linkage
VAR_STATICALLOCFUNCTIONSCOPE,   // C: static function variable LLVM:
 ↪  global with internal linkage, variable name is prefixed by function
 ↪  name
VAR_OTHER            // In the future we may also call out VAR_STACK and
 ↪  VAR_HEAP
```

```
FUNCTIONENTRY,

PARAM_FORMALIN,
PARAM_FORMALOUT,
PARAM_ACTUALIN,
PARAM_ACTUALOUT

ANNOTATION_VAR,
ANNOTATION_GLOBAL,
ANNOTATION_OTHER
```

5.2.2.2 Node Definitions We define each node type/sub-type below and provide an example for each.

5.2.2.2.1 Instructions

Description: The PDG nodes representing LLVM instructions have the type <INST>_. For convenience, we create separate subtypes for some instructions, and the rest are included in other. Each type of instruction node is disjoint and the union of each type is the universe of possible instruction nodes. * INST_FUNCALL** Description: Represents an LLVM *call* instruction.

- Example:
  - Source Line: `36: greeter(username, &age);`
  - IR Line: `159:  call void @greeter(i8* %10, i32* %2), !dbg !114`

INST_RET Description: Represents an LLVM *ret* instruction.

- Example:
  - Source Line: `27:      return sz;`
  - IR Line: `140:  ret i32 %35, !dbg !93`

INST_BR Description: Represents an LLVM *br* instruction that contains a condition.

- Example:
  - Source Line: `25:  for (i=0; i<sz; i++)`

- IR Line: `108: br i1 %11, label %12, label %34, !dbg !79`

INST_OTHER Description: Represents all other LLVM instructions not specified above. Future versions of this specification may call out instructions for switch and indirect branches as separate subtypes.

5.2.2.2.2 Variables

- Description: Variable nodes denote where memory is allocated and have the type <VAR>_. Each type of variable node is disjoint and the union of each type is the universe of possible variable nodes.
  * VAR_STATICGLOBAL** Description: Represents the allocation site for a *global* variable.
- Example:
  - Source Line: `6: char *ciphertext;`
  - IR Line: `11: @ciphertext = common dso_local global i8* null, align 8, !dbg !16`

VAR_STATICMODULE Description: Represents the allocation site for a *static global* variable.

- Example:
  - Source Line: `7: static unsigned int i;`
  - IR Line: `10: @i = internal global i32 0, align 4, !dbg !18`

VAR_STATICFUNCTION Description: Represents the allocation site for a *static function* variable.

- Example:
  - Source Line: `11: static int sample = 1;`
  - IR Line: `6: @greeter.sample = internal global i32 1, align 4, !dbg !0`

VAR_OTHER Description: Represents a variable allocation site not captured by VAR_STATICGLOBAL, VAR_STATICMODULE, or VAR_STATICFUNCTION.

5.2.2.3 Function Entry

FUNCTIONENTRY Description: Defines an entry point to a function.

- Example:
  - Source Line: `17:  void initkey (int sz) {`
  - IR Line: `51:  define dso_local void @initkey(i32 %0) #0 !dbg !38 {`

5.2.2.4 Parameters

- Description: Parameter nodes denote formal input/output parameters or actual input/output parameters and have the type <PARAM>_. These parameter types are disjoint and the union of their types is the universe of possible parameter types. * PARAM_FORMALIN** Description: Associated with every formal parameter in a function is a tree of PARAM_FORMALIN node(s).
- Example:
  - Source Line: `9:  void greeter (char *str, int* s) {`
  - IR Line: `24:  define dso_local void @greeter(i8* %0, i32* %1) #0 !dbg !2 {`

PARAM_FORMALOUT Description: Associated with every formal parameter in a function that is modified in the function body is a tree of PARAM_FORMALOUT node(s).

- Example:
  - Source Line: `9:  void greeter (char *str, int* s) {`
  - IR Line: `26:  define dso_local void @greeter(i8* %0, i32* %1) #0 !dbg !2 {`

PARAM_ACTUALIN Description: Associated with every actual parameter of a function call is a tree of PARAM_ACTUALIN node(s).

- Example:
  - Source Line: `26:  greeter(username, &age);`
  - IR Line: `161:  call void @greeter(i8* %10, i32* %2), !dbg !114`

PARAM_ACTUALOUT Description: Associated with every actual parameter received by the caller after the corresponding argument has been modified during the function call is a tree of PARAM_ACTUALOUT node(s).

- Example:
  - Source Line: `26: greeter(username, &age);`
  - IR Line: `161: call void @greeter(i8* %10, i32* %2), !dbg !114`

5.2.2.4.1 Annotations

- Description: Annotation nodes denote LLVM annotations and have the type <ANNOTATION>_. The label associated with an annotation can have multiple values. Each annotation type is disjoint and the union of annotation types gives the universe of possible annotation nodes. * ANNOTATION_VAR** Description: Represents the annotation of a variable.
- Example:
  - Source Line: `32: char __attribute__((annotate("confidential")))username[20];`
  - IR Line: `14: @.str.4 = private unnamed_addr constant [13 x i8] c"confidential\00", section "llvm.metadata"`
  - IR Line: `155: call void @llvm.var.annotation(i8* %6, i8* getelementptr inbounds ([13 x i8], [13 x i8]* @.str.4, i32 0, i32 0), i8* getelementptr inbounds ([10 x i8], [10 x i8]* @.str.3, i32 0, i32 0), i32 32), !dbg !104`

ANNOTATION_GLOBAL Description: Contains the annotations for exactly one function or global variable.

- Example:
  - Source Line: `23: int __attribute__((annotate("sensitive"))) encrypt (char *plaintext, int sz) {`
  - IR Line: `12: @.str.2 = private unnamed_addr constant [10 x i8] c"sensitive\00", section "llvm.metadata"`
  - IR Line:`23: @llvm.global.annotations = appending global [2 x { i8*, i8*, i8*, i32 }] [{ i8*, i8*, i8*, i32 } { i8* bitcast (i32 (i8*, i32)* @encrypt to i8*), i8* getelementptr inbounds ([10 x i8], [10 x i8]* @.str.2, i32 0, i32 0), i8* getelementptr inbounds ([10 x i8], [10 x i8]* @.str.3, i32 0, i32 0), i32 23 }, { i8*, i8*, i8*, i32 } { i8* bitcast (i8** @key to i8*), i8* getelementptr inbounds ([7 x i8], [7 x i8]* @.str.12, i32 0, i32 0), i8* getelementptr inbounds ([10 x i8], [10 x i8]* @.str.3, i32 0, i32 0), i32 5 }], section "llvm.metadata"`
  - Note: The node will refer to the first index of the llvm.global.annotations array in this example.

ANNOTATION_OTHER Description: Represents an annotation not captured by ANNOTATION_VAR or ANNOTATION_GLOBAL.

5.2.3 Node Type Properties

Listed below are some useful properties of PDG node types and subtypes. Recall that AllDisjoint(<set1>, <set2>, ..., <setn>) indicates the set(s) specified are disjoint, meaning their intersection is empty. Universe(<set1>, <set2>, ..., <setn>) results in the union of the specified set(s).

- Every instruction in an LLVM IR instance will have a corresponding node in the PDG, and the set of these nodes is INST.
- Universe(INST_FUNCALL, INST_RET, INST_BR, INST_OTHER) = INST
- Every variable allocation in an LLVM IR instance will have a corresponding node in the PDG, and the set of these nodes is VAR.
- Universe(VAR_STATICGLOBAL, VAR_STATICFUNCTION, VAR_STATICMODULE, VAR_OTHER) = VAR
- Every annotation in an LLVM IR instance will have a corresponding node in the PDG, and the set of these nodes is ANNOTATION.
- Universe(ANNOTATION_VAR, ANNOTATION_GLOBAL, ANNOTATION_OTHER) = ANNOTATION
- Universe(PARAM_FORMALIN, PARAM_FORMALOUT, PARAM_ACTUALIN, PARAM_ACTUALOUT) = PARAM
- Universe(INST, VAR, ANNOTATION, FUNCTIONENTRY, PARAM) = PDGNODES
- AllDisjoint(INST_FUNCALL, INST_RET, INST_BR, INST_OTHER)
- AllDisjoint(VAR_STATICGLOBAL, VAR_STATICFUNCTION, VAR_STATICMODULE, VAR_STACK, VAR_HEAP)
- AllDisjoint(ANNOTATION_VAR, ANNOTATION_GLOBAL, ANNOTATION_OTHER)
- AllDisjoint(PARAM_FORMALIN, PARAM_FORMALOUT, PARAM_ACTUALIN, PARAM_ACTUALOUT)
- AllDisjoint(INST, VAR, ANNOTATION, FUNCTIONENTRY, PARAM) ***

5.2.4 Edges

5.2.4.1 Edge Types Summary

We summaizee the PDG edge types below.

```
CONTROLDEP_CALLINV,
CONTROLDEP_CALLRET,
```

```
  CONTROLDEP_ENTRY,
  CONTROLDEP_BR,
  CONTROLDEP_OTHER,

  DATADEP_DEFUSE,
  DATADEP_RAW,
  DATADEP_RET,
  DATADEP_ALIAS,

  PARAMETER_IN,
  PARAMETER_OUT,
  PARAMETER_FIELD,

  ANNO_GLOBAL,
  ANNO_VAR,
 ANNO_OTHER
```

5.2.5 Edge Definitions

5.2.5.1 Control Dependency Edges

- Description: Control dependency edges are used to indicate that the execution of the destination node depends on the execution of the source node and have type: <CONTROLDEP>_. Each type of CONTROLDEP edge is disjoint and the union of each type is the universe of possible CONTROLDEP edges. * CONTROLDEP_CALLINV** Description: CONTROLDEP_CALLINV edge connects an INST_FUNCALL node with the FUNCTIONENTRY node of callee. It indicates the control flow transition from the caller to callee.
- Example:
  - Source
    - Source Line: `26:  greeter(username, &age);`
    - IR Line: `161:  call void @greeter(i8* %10, i32* %2), !dbg !114`
  - Destination
    - Source Line:`9:  void greeter (char *str, int* s) {`
    - IR Line: `26:  define dso_local void @greeter(i8* %0, i32* %1) #0 !dbg !2 {`

CONTROLDEP_CALLRET Description: CONTROLDEP_CALLRET edge connects an INST_RET node with the corresponding INST_FUNCALL of the caller. It indicates the control flow transition from the callee back to the caller. If there are multiple INST_RET instructions, they will map to the same INST_FUNCALL node.

- Example:
  - Source
    - Source Line: `27: return sz;`
    - IR Line: `140: ret i32 %35, !dbg !93`
  - Destination
    - Source Line: `41: int sz = encrypt(text, strlen(text));`
    - IR Line: `174: %21 = call i32 @encrypt(i8* %17, i32 %20), !dbg !126`

CONTROLDEP_ENTRY Description: Connects the FUNCTIONENTRY node with every INST_[TYPE] node that would be unconditionally executed inside a function body.

- Example:
  - Source
    - Source Line: `9: void greeter (char *str, int* s) {`
    - IR Line: `26: define dso_local void @greeter(i8* %0, i32* %1) #0 !dbg !2 {`
  - Destination
    - Source Line: `10: char* p = str;`
    - IR Line: `27: %3 = alloca i8*, align 8`

CONTROLDEP_BR Description: Connects an INST_BR node to every INST_[TYPE] node of the control-dependent basic block(s)'s.

- Example: In the example label %12 corresponds to line 108
  - Source
    - Source Line: `25: for (i=0; i<sz; i++)`
    - IR Line: `108: br i1 %11, label %12, label %34, !dbg !79`
  - Destination
    - Source Line: `26: ciphertext[i]=plaintext[i] ^ key[i];`
    - IR Line: `111: %13 = load i8*, i8** %3, align 8, !dbg !80`

CONTROLDEP_OTHER Description: Captures control dependencies not captured by the above.

5.2.5.2 Data Dependency Edges

- Description: Data dependency edges are used to indicate that the source node refers to data used by the destination node and have type: <DATADEP>_. Each type of DATADEP edge is disjoint and the union of each type is the universe of possible DATADEP edges. * DATADEP_DEFUSE** Description: DATADEP_DEFUSE edge connects a def node (Either an INST_[Type] or VAR_[Type]) and use node (INST_[Type]). It is directly computed from the LLVM def-use chain.
- Example:
  - Source
    - Source Line: `40: initkey(strlen(text));`
    - IR Line: `166:  %15 = call i64 @strlen(i8* %14) #7, !dbg !119`
  - Destination
    - Source Line: `40: initkey(strlen(text));`
    - IR Line: `167:  %16 = trunc i64 %15 to i32, !dbg !119`

DATADEP_RAW Description: DATA_RAW connects two INST_[Type] nodes with read-after-write dependence. This is flow-sensitive. We use memory dependency LLVM pass to compute this information.

- Example:
  - Source
    - Source Line: `24: ciphertext = (char *) (malloc (sz));`
    - IR Line: `95: store i32 %1, i32* %4, align 4`
  - Destination
    - Source Line: `24: ciphertext = (char *) (malloc (sz));`
    - IR Line: `97: %5 = load i32, i32* %4, align 4, !dbg !69`

DATADEP_RET Description: DATA_RET edge connects the return value from an Inst_RET node to the corresponding INST_FUNCALL node in the caller function. It indicates the data flow from the return instruction to the call instruction

- Example:
  - Source
    - Source Line: `27:  return sz;`
    - IR Line: `140:  ret i32 %35, !dbg !93`
  - Destination
    - Source Line: `41: int sz = encrypt(text, strlen(text));`

        * IR Line: `174: %21 = call i32 @encrypt(i8* %17, i32 %20), !dbg !126`

DATADEP_ALIAS Description: DATA_ALIAS edge connects two nodes that have may-alias relations, meaning the source and destination node might (not must) refer to the same memory location. Note that if two nodes n1 and n2 have may-alias relation, then there are two DATA_ALIAS edges exist between them, one from n1 to n2 and one from n2 to n1. This is because the alias relation is bidirectional.

- Example:
  - Source
    * Source Line: `10: char* p = str;`
    * IR Line: `35: %6 = load i8*, i8** %3, align 8, !dbg !31`
  - Destination
    * Source Line: `12: printf("%s\n", p);`
    * IR Line: `37: %7 = load i8*, i8** %5, align 8, !dbg !32`

5.2.5.3 Parameter Tree Edges

- Description: Parameter tree edges show data flow from the caller to callee and vice versa denoted with type: <PARAMETER>_. Each type of PARAMETER edge is disjoint and the union of each type is the universe of possible PARAMETER edges. ***

PARAMETER_IN Description: PARAMETER_IN edge represents interprocedural data flow from caller to the callee. It connects

1. actual_parameter_in_tree node and formal_in tree nodes
2. formal_in_tree node and the IR variables that correspond to these formal_in_tree nodes in the callee.

- Example:
  - Source
    * Source Line: `26: greeter(username, &age);`
    * IR Line: `161: call void @greeter(i8* %10, i32* %2), !dbg !114`
  - Destination
    * Source Line: `9: void greeter (char *str, int* s) {`
    * IR Line: `26: define dso_local void @greeter(i8* %0, i32* %1) #0 !dbg !2 {`

PARAMETER_OUT Description: edge represents data flow from callee to caller. It connects

1. arguments modified in callee to formal_out_tree node.
2. formal_out_tree node to actual_out_tree node.
3. actual_out_tree node to the modified variable in the caller.

- Example:
  - Source
    - Source Line: `9:  void greeter (char *str, int* s) {`
    - IR Line: `26:  define dso_local void @greeter(i8* %0, i32* %1) #0 !dbg !2 {`
  - Destination
    - Source Line: `26:  greeter(username, &age);`
    - IR Line: `151:  call void @greeter(i8* %10, i32* %2), !dbg !114`

PARAMETER_FIELD Description: PARAMETER_FIELD edge connects a parent parameter tree node to a child parameter tree node.

- Example:
  - Source
    - Source Line: `9:  void greeter (char *str, int* s) {`
    - IR Line: `26:  define dso_local void @greeter(i8* %0, i32* %1) #0 !dbg !2 {`
  - Destination
    - Source Line: `9:  void greeter (char *str, int* s) {`
    - IR Line: `26:  define dso_local void @greeter(i8* %0, i32* %1) #0 !dbg !2 {`

5.2.5.4 Annotations

- Description: Annotation Nodes are connected to other nodes with <ANNO>_ edge type. Each type of ANNO edge is disjoint and the union of each type is the universe of possible ANNO edges. ***

ANNO_GLOBAL Description: Connects FUNCTIONENTRY nodes or VAR_STATIC* nodes to ANNOTATION_GLOBAL nodes.

- Example:

  - Source
    - Source Line: `23: int __attribute__((annotate("sensitive"))) encrypt (char *plaintext, int sz) {`
    - IR Line: `90: define dso_local i32 @encrypt(i8* %0, i32 %1) #0 !dbg !62 {`
  - Destination
    - Source Line: `23: int __attribute__((annotate("sensitive"))) encrypt (char *plaintext, int sz) {`
    - IR Line: `12: @.str.2 = private unnamed_addr constant [10 x i8] c"sensitive\00", section "llvm.metadata"`
    - IR Line: `23: @llvm.global.annotations = appending global [2 x { i8*, i8*, i8*, i32 }] [{ i8*, i8*, i8*, i32 } { i8* bitcast (i32 (i8*, i32)* @encrypt to i8*), i8* getelementptr inbounds ([10 x i8], [10 x i8]* @.str.2, i32 0, i32 0), i8* getelementptr inbounds ([10 x i8], [10 x i8]* @.str.3, i32 0, i32 0), i32 23 }, { i8*, i8*, i8*, i32 } { i8* bitcast (i8** @key to i8*), i8* getelementptr inbounds ([7 x i8], [7 x i8]* @.str.12, i32 0, i32 0), i8* getelementptr inbounds ([10 x i8], [10 x i8]* @.str.3, i32 0, i32 0), i32 5 }], section "llvm.metadata"`

ANNO_VAR Description: Connects INST nodes or VAR_OTHER nodes to ANNOTATION_VAR nodes.

- Example:
  - Source
    - Source Line: `32: char __attribute__((annotate("confidential")))username[20];`
    - IR Line: `145: %3 = alloca [20 x i8], align 16`
  - Destination
    - Source Line: `32: 32: char __attribute__((annotate("confidential")))username[20`
    - IR Line: `14: @.str.4 = private unnamed_addr constant [13 x i8] c"confidential\00", section "llvm.metadata""`
    - IR Line: `call void @llvm.var.annotation(i8* %6, i8* getelementptr inbounds ([13 x i8], [13 x i8]* @.str.4, i32 0, i32 0), i8* getelementptr inbounds ([10 x i8], [10 x i8]* @.str.3, i32 0, i32 0), i32 32), !dbg !104`

ANNO_OTHER Description: Connects a PDG node to an annotation type not captured by the above.

5.2.6 Edge Type Properties

Listed below are useful properties of PDG edge types and subtypes.

- Every instruction node inside a function body is reachable from the FUNC_ENTRY node through CONTROLDEP_* edges.
- From the FUNC_ENTRY node for the main function, every other function is reachable unless the function dead code (never called).
- Universe(CONTROLDEP_CALLINV, CONTROLDEP_CALLRET, CONTROLDEP_ENTRY, CONTROLDEP_BR, CONTROLDEP_OTHER) = CONTROLDEP
- Universe(DATADEP_DEFUSE, DATADEP_RAW, DATADEP_READ, DATADEP_RET, DATADEP_ALIAS) = DATADEP
- Universe(ANNO_FUNC, ANNO_VAR, ANNO_OTHER) = ANNO
- Universe(VARDEP_STATIC, VARDEP_OTHER) = VARDEP
- Universe(PARAMETER_IN, PARAMETER_OUT, PARAMETER_FIELD) = PARAMETER
- Universe(CONTROLDEP, DATADEP, ANNO, VARDEP, PARAMETER) = PDGEDGES
- AllDisjoint(CONTROLDEP_CALLINV, CONTROLDEP_CALLRET, CONTROLDEP_ENTRY, CONTROLDEP_BR, CONTROLDEP_OTHER)
- AllDisjoint(DATADEP_DEFUSE, DATADEP_RAW, DATADEP_READ, DATADEP_RET, DATADEP_ALIAS)
- AllDisjoint(ANNO_FUNC, ANNO_VAR, ANNO_OTHER)
- AllDisjoint(VARDEP_STATIC, VARDEP_OTHER)
- AllDisjoint(PARAMETER_IN, PARAMETER_OUT, PARAMETER_FIELD)
- AllDisjoint(CONTROLDEP, DATADEP, ANNO, VARDEP, PARAMETER) ***

5.2.7 Limitations of current PDG implementation

- Does not support C++
- Does not support goto statements
- Does not support variadic functions

5.3 The cross-domain cut specification

The assignments of functions and global variables are described in `topology.json`, and the full assignments of every node in the PDG and the listing of edges in the cut are provided in `artifact.json`. The former is used by downstream tools, and the latter is provided for convenience to anyone who many want to perform independent verification. The CLOSURE project has been using these file names by convention, but developers are free to choose other names.

The `topology.json` file is a description of level and enclave assignments produced by the conflict analyzer and is used as input for the code divider.

The `topology.json` file contains:

1. the set of enclaves and levels relevant to the program
2. an assignment from each function and global variable to a level and an enclave
3. source information, such as the source directory (`source_path`) and the file and line numbers for the functions/global variables involved in the assignments

The `topology.json` generated for example1 is as follows:


```
{
  "source_path": "/workspaces/build/apps/examples/example1/refactored",
  "enclaves": [
    "purple_E",
    "orange_E"
  ],
  "levels": [
    "purple",
    "orange"
  ],
  "functions": [
    {
      "name": "get_a",
      "level": "orange",
      "enclave": "orange_E",
      "line": 47
    },
    {
      "name": "ewma_main",
      "level": "purple",
      "enclave": "purple_E",
      "line": 69
    },
    {
      "name": "get_b",
      "level": "purple",
      "enclave": "purple_E",
      "line": 58
    },
    {
      "name": "calc_ewma",
      "level": "purple",
      "enclave": "purple_E",
      "line": 39
    },
```

```
    {
      "name": "main",
      "level": "purple",
      "enclave": "purple_E",
      "line": 87
    }
  ],
  "global_scoped_vars": []
}
```

Produced by the conflict analyzer, the `artifact.json` contains all the detailed enclave, label and level assignments to every node defined in the PDG for a given program. It also contains some debug information, such as associated lines and names from the source. Here's a sample `artifact.json` for example 1:

```
{
    "source_path": "/workspaces/build/capo/C",
    "function-assignments": [
        {"node": 74, "label": "XDLINKAGE_GET_A", "enclave": "orange_E",
        ↪  "level": "orange", "debug": {"line": 47, "name": "get_a"}},
        {"node": 75, "label": "EWMA_MAIN", "enclave": "purple_E",
        ↪  "level": "purple", "debug": {"line": 69, "name":
        ↪  "ewma_main"}},
        {"node": 76, "label": "PURPLE", "enclave": "purple_E", "level":
        ↪  "purple", "debug": {"line": 58, "name": "get_b"}},
        {"node": 77, "label": "PURPLE", "enclave": "purple_E", "level":
        ↪  "purple", "debug": {"line": 39, "name": "calc_ewma"}},
        {"node": 78, "label": "PURPLE", "enclave": "purple_E", "level":
        ↪  "purple", "debug": {"line": 87, "name": "main"}}
    ],
    "variable-assignments": [],
    "all-assignments": [
        {"node": 74, "label": "XDLINKAGE_GET_A", "enclave": "orange_E",
        ↪  "level": "orange", "debug": {"line": 47, "name": "get_a"}},
        {"node": 75, "label": "EWMA_MAIN", "enclave": "purple_E",
        ↪  "level": "purple", "debug": {"line": 69, "name":
        ↪  "ewma_main"}},
        {"node": 76, "label": "PURPLE", "enclave": "purple_E", "level":
        ↪  "purple", "debug": {"line": 58, "name": "get_b"}},
        {"node": 77, "label": "PURPLE", "enclave": "purple_E", "level":
        ↪  "purple", "debug": {"line": 39, "name": "calc_ewma"}},
        {"node": 78, "label": "PURPLE", "enclave": "purple_E", "level":
        ↪  "purple", "debug": {"line": 87, "name": "main"}},
```

```
{"node": 1, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 82, "name": "calc_ewma"}},
{"node": 2, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 80, "name": "get_a"}},
{"node": 3, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 81, "name": "get_b"}},
{"node": 4, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 83, "name": "printf"}},
{"node": 5, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 88, "name": "ewma_main"}},
{"node": 6, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 72, "name":
 ↪  "llvm.dbg.declare"}},
{"node": 7, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 73, "name":
 ↪  "llvm.dbg.declare"}},
{"node": 8, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 76, "name":
 ↪  "llvm.dbg.declare"}},
{"node": 9, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 79, "name":
 ↪  "llvm.dbg.declare"}},
{"node": 10, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 39, "name":
 ↪  "llvm.dbg.declare"}},
{"node": 11, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 39, "name":
 ↪  "llvm.dbg.declare"}},
{"node": 12, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 40, "name":
 ↪  "llvm.dbg.declare"}},
{"node": 13, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 87, "name":
 ↪  "llvm.dbg.declare"}},
{"node": 14, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 87, "name":
 ↪  "llvm.dbg.declare"}},
{"node": 15, "label": "ORANGE", "enclave": "orange_E", "level":
 ↪  "orange", "debug": {"line": 56, "name": null}},
{"node": 16, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 85, "name": null}},
{"node": 17, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 65, "name": null}},
```

```
{"node": 18, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 43, "name": null}},
{"node": 19, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 88, "name": null}},
{"node": 20, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 79, "name": null}},
{"node": 21, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 79, "name": null}},
{"node": 22, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 84, "name": null}},
{"node": 23, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 79, "name": null}},
{"node": 24, "label": "ORANGE", "enclave": "orange_E", "level":
 ↪  "orange", "debug": {"line": 55, "name": "get_a.a"}},
{"node": 25, "label": "ORANGE", "enclave": "orange_E", "level":
 ↪  "orange", "debug": {"line": 55, "name": null}},
{"node": 26, "label": "ORANGE", "enclave": "orange_E", "level":
 ↪  "orange", "debug": {"line": 55, "name": "get_a.a"}},
{"node": 27, "label": "ORANGE", "enclave": "orange_E", "level":
 ↪  "orange", "debug": {"line": 56, "name": "get_a.a"}},
{"node": 28, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 0, "name": null}},
{"node": 29, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 0, "name": null}},
{"node": 30, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 0, "name": null}},
{"node": 31, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 0, "name": null}},
{"node": 32, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 76, "name": null}},
{"node": 33, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 79, "name": null}},
{"node": 34, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 79, "name": null}},
{"node": 35, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 79, "name": null}},
{"node": 36, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 80, "name": null}},
{"node": 37, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 81, "name": null}},
{"node": 38, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 82, "name": null}},
{"node": 39, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 82, "name": null}},
```

```
{"node": 40, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 82, "name": null}},
{"node": 41, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 83, "name": null}},
{"node": 42, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 79, "name": null}},
{"node": 43, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 79, "name": null}},
{"node": 44, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 79, "name": null}},
{"node": 45, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 64, "name": "get_b.b"}},
{"node": 46, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 64, "name": "get_b.b"}},
{"node": 47, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 64, "name": null}},
{"node": 48, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 64, "name": "get_b.b"}},
{"node": 49, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 65, "name": "get_b.b"}},
{"node": 50, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 0, "name": null}},
{"node": 51, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 0, "name": null}},
{"node": 52, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 0, "name": null}},
{"node": 53, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 0, "name": null}},
{"node": 54, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 0, "name": null}},
{"node": 55, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 40, "name": null}},
{"node": 56, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 42, "name": null}},
{"node": 57, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 42, "name": null}},
{"node": 58, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 42, "name": null}},
{"node": 59, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 42, "name": null}},
{"node": 60, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 42, "name": "calc_ewma.c"}},
{"node": 61, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 42, "name": null}},
```

```
{"node": 62, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 42, "name": null}},
{"node": 63, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 42, "name": "calc_ewma.c"}},
{"node": 64, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 43, "name": "calc_ewma.c"}},
{"node": 65, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 0, "name": null}},
{"node": 66, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 0, "name": null}},
{"node": 67, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 0, "name": null}},
{"node": 68, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 0, "name": null}},
{"node": 69, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 0, "name": null}},
{"node": 70, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 0, "name": null}},
{"node": 71, "label": "ORANGE", "enclave": "orange_E", "level":
 ↪  "orange", "debug": {"line": 52, "name": "get_a.a"}},
{"node": 72, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 61, "name": "get_b.b"}},
{"node": 73, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 41, "name": "calc_ewma.c"}},
{"node": 79, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 39, "name": null}},
{"node": 80, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 39, "name": null}},
{"node": 81, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 87, "name": null}},
{"node": 82, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 87, "name": null}},
{"node": 83, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 87, "name": null}},
{"node": 84, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 39, "name": null}},
{"node": 85, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 39, "name": null}},
{"node": 86, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 87, "name": null}},
{"node": 87, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 87, "name": null}},
{"node": 88, "label": "PURPLE", "enclave": "purple_E", "level":
 ↪  "purple", "debug": {"line": 87, "name": null}},
```

```
        {"node": 89, "label": "PURPLE", "enclave": "purple_E", "level":
         ↪  "purple", "debug": {"line": 39, "name": null}},
        {"node": 90, "label": "PURPLE", "enclave": "purple_E", "level":
         ↪  "purple", "debug": {"line": 39, "name": null}},
        {"node": 91, "label": "PURPLE", "enclave": "purple_E", "level":
         ↪  "purple", "debug": {"line": 39, "name": null}},
        {"node": 92, "label": "PURPLE", "enclave": "purple_E", "level":
         ↪  "purple", "debug": {"line": 39, "name": null}},
        {"node": 93, "label": "nullCleLabel", "enclave": "nullEnclave",
         ↪  "level": "nullLevel", "debug": {"line": 76, "name":
         ↪  "llvm.var.annotation"}},
        {"node": 94, "label": "nullCleLabel", "enclave": "nullEnclave",
         ↪  "level": "nullLevel", "debug": {"line": null, "name":
         ↪  "llvm.global.annotations"}}
    ],
    "cut": [{"summary":
     ↪  "(2:PURPLE)--[purple_E]--||-->[orange_E]--(74:XDLINKAGE_GET_A)",
     ↪  "source-node": 2, "source-label": "PURPLE", "source-enclave":
     ↪  "purple_E", "dest-node": 74, "dest-label": "XDLINKAGE_GET_A",
     ↪  "dest-enclave": "orange_E"}]
}
```

5.4 Additional files for Constraint Model in MiniZinc

The following contains type declarations for the MiniZinc model used within the conflict analyzer. These type declarations, along with a model instance generated in python are inputted to MiniZinc along with the constraints to either produce a satisfiable assignment or some minimally unsatisfiable set of constraints.

5.4.1 Type declarations

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PDG Nodes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

int: Inst_FunCall_start;
int: Inst_FunCall_end;
int: Inst_Ret_start;
int: Inst_Ret_end;
int: Inst_Br_start;
int: Inst_Br_end;
int: Inst_Other_start;
```

```
int: Inst_Other_end;
int: Inst_start;
int: Inst_end;

int: VarNode_StaticGlobal_start;
int: VarNode_StaticGlobal_end;
int: VarNode_StaticModule_start;
int: VarNode_StaticModule_end;
int: VarNode_StaticFunction_start;
int: VarNode_StaticFunction_end;
int: VarNode_StaticOther_start;
int: VarNode_StaticOther_end;
int: VarNode_start;
int: VarNode_end;

int: FunctionEntry_start;
int: FunctionEntry_end;

int: Param_FormalIn_start;
int: Param_FormalIn_end;
int: Param_FormalOut_start;
int: Param_FormalOut_end;
int: Param_ActualIn_start;
int: Param_ActualIn_end;
int: Param_ActualOut_start;
int: Param_ActualOut_end;
int: Param_start;
int: Param_end;

int: Annotation_Var_start;
int: Annotation_Var_end;
int: Annotation_Global_start;
int: Annotation_Global_end;
int: Annotation_Other_start;
int: Annotation_Other_end;
int: Annotation_start;
int: Annotation_end;

int: PDGNode_start;
int: PDGNode_end;

set of int: Inst = Inst_start .. Inst_end;
```

```
set of int: VarNode_StaticGlobal = VarNode_StaticGlobal_start ..
 ↪  VarNode_StaticGlobal_end;
set of int: VarNode_StaticModule = VarNode_StaticModule_start ..
 ↪  VarNode_StaticModule_end;
set of int: VarNode_StaticFunction = VarNode_StaticFunction_start ..
 ↪  VarNode_StaticFunction_end;
set of int: VarNode_StaticOther = VarNode_StaticOther_start ..
 ↪  VarNode_StaticOther_end;
set of int: VarNode = VarNode_start .. VarNode_end;

set of int: FunctionEntry = FunctionEntry_start .. FunctionEntry_end;

set of int: Param_FormalIn = Param_FormalIn_start .. Param_FormalIn_end;
set of int: Param_FormalOut = Param_FormalOut_start ..
 ↪  Param_FormalOut_end;
set of int: Param_ActualIn = Param_ActualIn_start .. Param_ActualIn_end;
set of int: Param_ActualOut = Param_ActualOut_start ..
 ↪  Param_ActualOut_end;
set of int: Param = Param_start .. Param_end;

set of int: Annotation_Var = Annotation_Var_start .. Annotation_Var_end;
set of int: Annotation_Global = Annotation_Global_start ..
 ↪  Annotation_Global_end;
set of int: Annotation_Other = Annotation_Other_start ..
 ↪  Annotation_Other_end;
set of int: Annotation  = Annotation_start .. Annotation_end;

set of int: PDGNodeIdx  = PDGNode_start .. PDGNode_end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PDG Edges
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

int: ControlDep_CallInv_start;
int: ControlDep_CallInv_end;
int: ControlDep_CallRet_start;
int: ControlDep_CallRet_end;
int: ControlDep_Entry_start;
int: ControlDep_Entry_end;
int: ControlDep_Br_start;
int: ControlDep_Br_end;
int: ControlDep_Other_start;
int: ControlDep_Other_end;
int: ControlDep_start;
```

```
int: ControlDep_end;

int: DataDepEdge_DefUse_start;
int: DataDepEdge_DefUse_end;
int: DataDepEdge_RAW_start;
int: DataDepEdge_RAW_end;
int: DataDepEdge_Ret_start;
int: DataDepEdge_Ret_end;
int: DataDepEdge_Alias_start;
int: DataDepEdge_Alias_end;
int: DataDepEdge_start;
int: DataDepEdge_end;

int: Parameter_In_start;
int: Parameter_In_end;
int: Parameter_Out_start;
int: Parameter_Out_end;
int: Parameter_Field_start;
int: Parameter_Field_end;
int: Parameter_start;
int: Parameter_end;

int: Anno_Global_start;
int: Anno_Global_end;
int: Anno_Var_start;
int: Anno_Var_end;
int: Anno_Other_start;
int: Anno_Other_end;
int: Anno_start;
int: Anno_end;

int: PDGEdge_start;
int: PDGEdge_end;

set of int: ControlDep_CallInv = ControlDep_CallInv_start ..
 ↪  ControlDep_CallInv_end;
set of int: ControlDep_CallRet = ControlDep_CallRet_start ..
 ↪  ControlDep_CallRet_end;
set of int: ControlDep_Entry = ControlDep_Entry_start ..
 ↪  ControlDep_Entry_end;
set of int: ControlDep_Br = ControlDep_Br_start .. ControlDep_Br_end;
set of int: ControlDep_Other = ControlDep_Other_start ..
 ↪  ControlDep_Other_end;
set of int: ControlDep = ControlDep_start .. ControlDep_end;
```

```
set of int: DataDepEdge_DefUse = DataDepEdge_DefUse_start ..
 ↪  DataDepEdge_DefUse_end;
set of int: DataDepEdge_RAW  = DataDepEdge_RAW_start ..
 ↪  DataDepEdge_RAW_end;
set of int: DataDepEdge_Ret  = DataDepEdge_Ret_start ..
 ↪  DataDepEdge_Ret_end;
set of int: DataDepEdge_Alias = DataDepEdge_Alias_start ..
 ↪  DataDepEdge_Alias_end;
set of int: DataDepEdge = DataDepEdge_start .. DataDepEdge_end;

set of int: Parameter_In = Parameter_In_start .. Parameter_In_end;
set of int: Parameter_Out = Parameter_Out_start .. Parameter_Out_end;
set of int: Parameter_Field = Parameter_Field_start ..
 ↪  Parameter_Field_end;
set of int: Parameter = Parameter_start .. Parameter_end;

set of int: Anno_Global = Anno_Global_start .. Anno_Global_end;
set of int: Anno_Var = Anno_Var_start .. Anno_Var_end;
set of int: Anno_Other = Anno_Other_start .. Anno_Other_end;
set of int: Anno = Anno_start .. Anno_end;

set of int: PDGEdgeIdx = PDGEdge_start .. PDGEdge_end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Containing Function for PDG Nodes, Endpoints for PDG Edges, Indices of
 ↪  Fucntion Formal Parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

array[PDGNodeIdx]     of int:  hasFunction;
array[PDGEdgeIdx]     of int:  hasSource;
array[PDGEdgeIdx]     of int:  hasDest;
array[Param]          of int:  hasParamIdx;
array[FunctionEntry]  of bool: userAnnotatedFunction;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Convenience Aggregations of PDG Nodes and Edges
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

set of int: Global              = VarNode_StaticGlobal union
 ↪  VarNode_StaticModule;
set of int: NonAnnotation       = Inst union VarNode union FunctionEntry
 ↪  union Param;
```

```
set of int: ControlDep_Call     = ControlDep_CallInv union
 ↪  ControlDep_CallRet;
set of int: ControlDep_NonCall  = ControlDep_Entry union ControlDep_Br
 ↪  union ControlDep_Other;
set of int: DataEdgeNoRet       = DataDepEdge_DefUse union
 ↪  DataDepEdge_RAW union DataDepEdge_Alias;
set of int: DataEdgeNoRetParam  = DataEdgeNoRet union Parameter;
set of int: DataEdgeParam       = DataDepEdge union Parameter;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Security Levels and Enclaves
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

enum Level;
enum Enclave;
array[Enclave] of Level: hasEnclaveLevel;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CLE Input Model
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

enum cleLabel;
enum cdf;
enum GuardOperation = {nullGuardOperation, allow, deny, redact};
enum Direction      = {nullDirection, bidirectional, egress, ingress};

int: MaxFuncParms;  % Max number of function parameters in the program
 ↪  (C<128, C++<256)
set of int: parmIdx = 1..MaxFuncParms;

array[cleLabel]                     of Level:          hasLabelLevel;
array[cleLabel]                     of bool:
 ↪  isFunctionAnnotation;

array[cdf]                          of cleLabel:       fromCleLabel;
array[cdf]                          of Level:          hasRemotelevel;
array[cdf]                          of GuardOperation:
 ↪  hasGuardOperation;
array[cdf]                          of Direction:      hasDirection;
array[cdf]                          of bool:           isOneway;
array[cleLabel, Level]              of cdf:
 ↪  cdfForRemoteLevel;
```

```
set of cdf: functionCdf = { x | x in cdf where
 ↪  isFunctionAnnotation[fromCleLabel[x]]==true };

array[functionCdf, cleLabel]          of bool:           hasRettaints;
array[functionCdf, cleLabel]          of bool:           hasCodtaints;
array[functionCdf, parmIdx, cleLabel] of bool:           hasArgtaints;
array[functionCdf, cleLabel]          of bool:           hasARCtaints;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Debug flag and decision variables for the solver
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

bool:                                                    debug;

array[PDGNodeIdx]                     of var Enclave:    nodeEnclave;
array[PDGNodeIdx]                     of var Level:      nodeLevel;
array[PDGNodeIdx]                     of var cleLabel:   taint;
array[PDGNodeIdx]                     of var cleLabel:   ftaint;

array[PDGEdgeIdx]                     of var Enclave:    esEnclave;
array[PDGEdgeIdx]                     of var Enclave:    edEnclave;

array[PDGEdgeIdx]                     of var cleLabel:   esTaint;
array[PDGEdgeIdx]                     of var cleLabel:   edTaint;

array[PDGEdgeIdx]                     of var cleLabel:   esFunTaint;
array[PDGEdgeIdx]                     of var cleLabel:   edFunTaint;

array[PDGEdgeIdx]                     of var cdf:        esFunCdf;
array[PDGEdgeIdx]                     of var cdf:        edFunCdf;

array[PDGEdgeIdx]                     of var bool:       xdedge;
array[PDGEdgeIdx]                     of var bool:       tcedge;

array[PDGEdgeIdx]                     of var bool:       coerced;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

5.4.2 PDG instance

The following is the MiniZinc representation of the PDG for example 1.

```
Inst_FunCall_start = 1;
Inst_FunCall_end = 14;
Inst_Ret_start = 15;
Inst_Ret_end = 19;
Inst_Br_start = 20;
Inst_Br_end = 23;
Inst_Other_start = 24;
Inst_Other_end = 71;
Inst_start = 1;
Inst_end = 71;
VarNode_StaticGlobal_start = 0;
VarNode_StaticGlobal_end = -1;
VarNode_StaticModule_start = 0;
VarNode_StaticModule_end = -1;
VarNode_StaticFunction_start = 72;
VarNode_StaticFunction_end = 74;
VarNode_StaticOther_start = 0;
VarNode_StaticOther_end = -1;
VarNode_start = 72;
VarNode_end = 74;
FunctionEntry_start = 75;
FunctionEntry_end = 79;
Param_FormalIn_start = 80;
Param_FormalIn_end = 84;
Param_FormalOut_start = 85;
Param_FormalOut_end = 89;
Param_ActualIn_start = 90;
Param_ActualIn_end = 91;
Param_ActualOut_start = 92;
Param_ActualOut_end = 93;
Param_start = 80;
Param_end = 93;
Annotation_Var_start = 94;
Annotation_Var_end = 95;
Annotation_Global_start = 96;
Annotation_Global_end = 96;
Annotation_Other_start = 0;
Annotation_Other_end = -1;
Annotation_start = 94;
Annotation_end = 96;
PDGNode_start = 1;
PDGNode_end = 96;
ControlDep_CallInv_start = 1;
ControlDep_CallInv_end = 4;
```

```
ControlDep_CallRet_start = 5;
ControlDep_CallRet_end = 8;
ControlDep_Entry_start = 9;
ControlDep_Entry_end = 70;
ControlDep_Br_start = 71;
ControlDep_Br_end = 85;
ControlDep_Other_start = 0;
ControlDep_Other_end = -1;
ControlDep_start = 1;
ControlDep_end = 85;
DataDepEdge_DefUse_start = 86;
DataDepEdge_DefUse_end = 141;
DataDepEdge_RAW_start = 142;
DataDepEdge_RAW_end = 148;
DataDepEdge_Ret_start = 149;
DataDepEdge_Ret_end = 152;
DataDepEdge_Alias_start = 153;
DataDepEdge_Alias_end = 154;
DataDepEdge_start = 86;
DataDepEdge_end = 154;
Parameter_In_start = 155;
Parameter_In_end = 166;
Parameter_Out_start = 167;
Parameter_Out_end = 174;
Parameter_Field_start = 175;
Parameter_Field_end = 176;
Parameter_start = 155;
Parameter_end = 176;
Anno_Global_start = 177;
Anno_Global_end = 180;
Anno_Var_start = 181;
Anno_Var_end = 184;
Anno_Other_start = 0;
Anno_Other_end = -1;
Anno_start = 177;
Anno_end = 184;
PDGEdge_start = 1;
PDGEdge_end = 184;
hasFunction = [
76,76,76,76,79,76,76,76,76,78,78,78,79,79,75,76,77,78,79,76,
76,76,76,75,75,75,75,76,76,76,76,76,76,76,76,76,76,76,76,76,
76,76,76,76,76,77,77,77,77,77,78,78,78,78,78,78,78,78,78,78,
78,78,78,78,78,79,79,79,79,79,79,0,0,0,75,76,77,78,79,78,
78,79,79,79,78,78,79,79,79,78,78,78,78,76,76,0
```

```
];
hasSource = [
1,2,3,5,15,16,17,18,75,75,75,75,75,76,76,76,76,76,76,76,
76,76,76,76,76,76,76,76,76,76,76,76,76,76,76,76,76,76,76,76,
77,77,77,77,77,77,78,78,78,78,78,78,78,78,78,78,78,78,78,78,
78,78,79,79,79,79,79,79,79,21,21,21,21,21,21,21,21,21,21,
21,21,21,21,21,72,72,72,73,73,73,73,74,74,74,24,25,27,28,28,
28,29,29,30,30,30,31,31,31,31,32,36,37,1,2,40,3,42,43,44,
46,47,48,50,51,51,52,52,53,57,58,59,60,61,62,63,65,66,67,68,
5,26,39,41,49,54,55,64,15,16,17,18,33,34,32,40,3,3,78,78,
80,81,79,79,90,91,3,3,78,78,79,79,92,93,83,88,72,73,75,76,
76,76,33,34
];
hasDest = [
75,77,78,76,1,5,2,3,25,26,27,15,24,28,29,30,31,33,34,35,
20,36,37,21,1,38,2,39,32,40,3,41,42,4,22,43,44,45,23,16,
47,48,49,50,17,46,51,52,53,54,55,56,57,58,59,60,61,62,63,64,
65,18,67,68,69,70,71,5,19,66,1,38,2,39,32,40,3,41,42,4,
22,43,44,45,23,27,26,24,50,49,47,46,65,64,61,25,26,15,32,38,
33,40,39,42,41,34,45,43,36,35,3,37,21,38,39,3,41,4,44,45,
48,48,49,17,57,54,58,55,56,59,59,60,63,62,63,64,18,69,70,71,
19,27,40,42,50,57,58,65,1,5,2,3,28,30,90,91,90,91,80,81,
57,58,82,83,80,81,92,93,85,86,87,88,32,40,84,89,96,96,96,96,
94,95,94,95
];
hasParamIdx = array1d(Param, [
1,2,1,2,-1,1,2,1,2,-1,1,2,1,2
]);
userAnnotatedFunction = array1d(FunctionEntry, [
 true,true,false,false,false
]);
MaxFuncParms = 3;
constraint ::  "TaintOnNodeIdx33" taint[33]=ORANGE;
constraint ::  "TaintOnNodeIdx34" taint[34]=PURPLE;
constraint ::  "TaintOnNodeIdx72" taint[72]=ORANGE;
constraint ::  "TaintOnNodeIdx73" taint[73]=PURPLE;
constraint ::  "TaintOnNodeIdx75" taint[75]=XDLINKAGE_GET_A;
constraint ::  "TaintOnNodeIdx76" taint[76]=EWMA_MAIN;
```

5.4.3 cle instance

The following is a representation of the annotations in example 1 in MiniZinc:

```
cleLabel = {nullCleLabel, XDLINKAGE_GET_A , TAG_RESPONSE_GET_A ,
 ↪  EWMA_MAIN , PURPLE , ORANGE , orangeDFLT , purpleDFLT };
hasLabelLevel = [nullLevel, orange , nullLevel , purple , purple ,
 ↪  orange , orange , purple ];
isFunctionAnnotation = [false, true , false , true , false , false ,
 ↪  false , false ];
cdf = {nullCdf, XDLINKAGE_GET_A_cdf_0 , XDLINKAGE_GET_A_cdf_1 ,
 ↪  EWMA_MAIN_cdf_0 , ORANGE_cdf_0 };
fromCleLabel = [nullCleLabel, XDLINKAGE_GET_A , XDLINKAGE_GET_A ,
 ↪  EWMA_MAIN , ORANGE ];
hasRemotelevel = [nullLevel, purple , orange , purple , purple ];
hasDirection = [nullDirection, bidirectional , bidirectional ,
 ↪  bidirectional , egress ];
hasGuardOperation = [nullGuardOperation, allow , allow , allow , allow
 ↪  ];
isOneway = [false, false , false , false , false ];
cdfForRemoteLevel = [|
 nullCdf, nullCdf , nullCdf
|nullCdf, XDLINKAGE_GET_A_cdf_1 , XDLINKAGE_GET_A_cdf_0
|nullCdf, nullCdf , nullCdf
|nullCdf, nullCdf , EWMA_MAIN_cdf_0
|nullCdf, nullCdf , nullCdf
|nullCdf, nullCdf , ORANGE_cdf_0
|nullCdf, nullCdf , nullCdf
|nullCdf, nullCdf , nullCdf
|];
hasRettaints = array2d(functionCdf, cleLabel, [
  false , false , true  , false , false , false , false , false
, false , false , true  , false , false , false , false , false
, false , false , false , false , true  , false , false , false
 ]);
hasCodtaints = array2d(functionCdf, cleLabel, [
  false , false , false , false , false , true  , false , false
, false , false , false , false , false , true  , false , false
, false , false , true  , false , true  , false , false , false
 ]);
hasArgtaints = array3d(functionCdf, parmIdx, cleLabel, [
  false , false , false         , false , false , false         , false
 ↪  , false , false         , false , false , false         , false ,
 ↪  false , false         , false , false , false         , false ,
 ↪  false , false         , false , false , false
```

```
, false , false , false         , false , false , false         , false
 ↪  , false , false         , false , false , false         , false ,
 ↪  false , false         , false , false , false         , false ,
 ↪  false , false         , false , false , false
, false , false , false         , false , false , false         , false
 ↪  , false , false         , false , false , false         , false ,
 ↪  false , false         , false , false , false         , false ,
 ↪  false , false         , false , false , false
 ]);
hasARCtaints = array2d(functionCdf, cleLabel, [
  false , true  , true  , false , false , true  , false , false
, false , true  , true  , false , false , true  , false , false
, false , false , true  , true  , true  , false , false , false
 ]);
```

5.4.4 enclave instance

The following is the MiniZinc representation of the mapping between enclaves and levels:

```
Level = {nullLevel,orange,purple};
Enclave = {nullEnclave, orange_E,purple_E};
hasEnclaveLevel = [nullLevel,orange,purple];
```

5.5 Constraint Solver Diagnostic Outputs

When the conflict analyzer cannot find an assignment that simultaneously satisfies all constraints entailed by the CLE annotations on the program under analysis, it will print out a diagnostic. The diagnostic is based on a minimum unsatisfiable subset of constraints determined by the findMUS utility of MiniZinc, which groups conflicts by constraint (name) with references to the PDG nodes and edges involved in the conflict. The diagnostic is further enhanced with source file and line number references for use by the CVI plugin to provide contextual guidance to the developer.

When the conflict analyzer finds a satisfying assignment it generates a `topology.json` file as described in the cross-domain cut specification section.

```
<constraint_name>:
  (<source_node_type>) <file>:<function>@<line> -> (<dest_node_type>) <file>:<function>@<
  (<node_type>) <file>:<function>@<line> # for nodes
```

Here's an example based on example1 with unsatisfiable annotations:

```
XDCallAllowed
    (Inst_FunCall) /workspaces/build/apps/examples/example1/annotated-working/example1.c@
    (Inst_FunCall) /workspaces/build/apps/examples/example1/annotated-working/example1.c@

    (Inst_Other) /workspaces/build/apps/examples/example1/annotated-working/example1.c@No
    (Inst_Other) /workspaces/build/apps/examples/example1/annotated-working/example1.c@ge
    (Inst_Other) /workspaces/build/apps/examples/example1/annotated-working/example1.c@No
    (Inst_Other) /workspaces/build/apps/examples/example1/annotated-working/example1.c@No
    (Inst_Ret) /workspaces/build/apps/examples/example1/annotated-working/example1.c@None
    (Inst_Other) /workspaces/build/apps/examples/example1/annotated-working/example1.c@No
    (Inst_Br) /workspaces/build/apps/examples/example1/annotated-working/example1.c@None
    (Inst_Br) /workspaces/build/apps/examples/example1/annotated-working/example1.c@None
    (Inst_FunCall) /workspaces/build/apps/examples/example1/annotated-working/example1.c@
    (Inst_FunCall) /workspaces/build/apps/examples/example1/annotated-working/example1.c@
    (Inst_Other) /workspaces/build/apps/examples/example1/annotated-working/example1.c@ge
    (Inst_FunCall) /workspaces/build/apps/examples/example1/annotated-working/example1.c@
    (Inst_Other) /workspaces/build/apps/examples/example1/annotated-working/example1.c@No
    (Inst_FunCall) /workspaces/build/apps/examples/example1/annotated-working/example1.c@
    (Inst_FunCall) /workspaces/build/apps/examples/example1/annotated-working/example1.c@
    (Inst_Other) /workspaces/build/apps/examples/example1/annotated-working/example1.c@No
    (Inst_FunCall) /workspaces/build/apps/examples/example1/annotated-working/example1.c@
    (Inst_Other) /workspaces/build/apps/examples/example1/annotated-working/example1.c@No
    (Inst_Br) /workspaces/build/apps/examples/example1/annotated-working/example1.c@None
    (Inst_Ret) /workspaces/build/apps/examples/example1/annotated-working/example1.c@None
    (Inst_Br) /workspaces/build/apps/examples/example1/annotated-working/example1.c@None
    (Inst_FunCall) /workspaces/build/apps/examples/example1/annotated-working/example1.c@
    (Inst_Other) /workspaces/build/apps/examples/example1/annotated-working/example1.c@No
    (Inst_Other) /workspaces/build/apps/examples/example1/annotated-working/example1.c@No
    (Inst_Other) /workspaces/build/apps/examples/example1/annotated-working/example1.c@ge
    (Inst_Other) /workspaces/build/apps/examples/example1/annotated-working/example1.c@No
    (Inst_FunCall) /workspaces/build/apps/examples/example1/annotated-working/example1.c@
    (Inst_FunCall) /workspaces/build/apps/examples/example1/annotated-working/example1.c@
    (Inst_FunCall) /workspaces/build/apps/examples/example1/annotated-working/example1.c@
    (Inst_Ret) /workspaces/build/apps/examples/example1/annotated-working/example1.c@None
    (Inst_Other) /workspaces/build/apps/examples/example1/annotated-working/example1.c@No
    (Inst_Other) /workspaces/build/apps/examples/example1/annotated-working/example1.c@No
    (Inst_Other) /workspaces/build/apps/examples/example1/annotated-working/example1.c@No
    (Inst_Other) /workspaces/build/apps/examples/example1/annotated-working/example1.c@No
    (Inst_Other) /workspaces/build/apps/examples/example1/annotated-working/example1.c@No
    (Inst_Other) /workspaces/build/apps/examples/example1/annotated-working/example1.c@No
    (Inst_Other) /workspaces/build/apps/examples/example1/annotated-working/example1.c@No
    (Inst_Other) /workspaces/build/apps/examples/example1/annotated-working/example1.c@No
    (Inst_Other) /workspaces/build/apps/examples/example1/annotated-working/example1.c@No
```

When `--output-json` mode is activated the solver will output a list of conflicts in JSON format. In this format, it can be read by CVI.


```
[
  {
    "name": "XDCallAllowed",
    "description": "TODO",
    "source": [
      {
        "file":
        ↪ "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
        "range": {
          "start": {
            "line": 85,
            "character": -1
          },
          "end": {
            "line": 85,
            "character": -1
          }
        }
      },
      {
        "file": "annotated/example1.c",
        "range": {
          "start": {
            "line": 46,
            "character": -1
          },
          "end": {
            "line": 46,
            "character": -1
          }
        }
      }
    ],
    "remedy": []
  },
  {
    "name": "XDCallAllowed",
    "description": "TODO",
    "source": [
      {
```

```
        "file":
         ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
        "range": {
          "start": {
            "line": 93,
            "character": -1
          },
          "end": {
            "line": 93,
            "character": -1
          }
        }
      },
      {
        "file": "annotated/example1.c",
        "range": {
          "start": {
            "line": 69,
            "character": -1
          },
          "end": {
            "line": 69,
            "character": -1
          }
        }
      }
    ],
    "remedy": []
  },
  {
    "name": "",
    "description": "TODO",
    "source": [
      {
        "file":
         ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
        "range": {
          "start": {
            "line": 81,
            "character": -1
          },
          "end": {
            "line": 81,
            "character": -1
```

```
        }
      }
    }
  ],
  "remedy": []
},
{
  "name": "",
  "description": "TODO",
  "source": [
    {
      "file":
       ↪ "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
      "range": {
        "start": {
          "line": 56,
          "character": -1
        },
        "end": {
          "line": 56,
          "character": -1
        }
      }
    }
  ],
  "remedy": []
},
{
  "name": "",
  "description": "TODO",
  "source": [
    {
      "file":
       ↪ "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
      "range": {
        "start": {
          "line": 87,
          "character": -1
        },
        "end": {
          "line": 87,
          "character": -1
        }
      }
```

```
      }
    ],
    "remedy": []
  },
  {
    "name": "",
    "description": "TODO",
    "source": [
      {
        "file":
         ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
        "range": {
          "start": {
            "line": 54,
            "character": -1
          },
          "end": {
            "line": 54,
            "character": -1
          }
        }
      }
    ],
    "remedy": []
  },
  {
    "name": "",
    "description": "TODO",
    "source": [
      {
        "file":
         ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
        "range": {
          "start": {
            "line": 90,
            "character": -1
          },
          "end": {
            "line": 90,
            "character": -1
          }
        }
      }
    ],
```

```
  "remedy": []
},
{
  "name": "",
  "description": "TODO",
  "source": [
    {
      "file":
       ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
      "range": {
        "start": {
          "line": 84,
          "character": -1
        },
        "end": {
          "line": 84,
          "character": -1
        }
      }
    }
  ],
  "remedy": []
},
{
  "name": "",
  "description": "TODO",
  "source": [
    {
      "file":
       ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
      "range": {
        "start": {
          "line": 89,
          "character": -1
        },
        "end": {
          "line": 89,
          "character": -1
        }
      }
    }
  ],
  "remedy": []
},
```

```
{
  "name": "",
  "description": "TODO",
  "source": [
    {
      "file":
       ↪ "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
      "range": {
        "start": {
          "line": 84,
          "character": -1
        },
        "end": {
          "line": 84,
          "character": -1
        }
      }
    }
  ],
  "remedy": []
},
{
  "name": "",
  "description": "TODO",
  "source": [
    {
      "file":
       ↪ "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
      "range": {
        "start": {
          "line": 75,
          "character": -1
        },
        "end": {
          "line": 75,
          "character": -1
        }
      }
    }
  ],
  "remedy": []
},
{
  "name": "",
```

```
    "description": "TODO",
    "source": [
      {
        "file":
        ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
        "range": {
          "start": {
            "line": 85,
            "character": -1
          },
          "end": {
            "line": 85,
            "character": -1
          }
        }
      }
    ],
    "remedy": []
  },
  {
    "name": "",
    "description": "TODO",
    "source": [
      {
        "file":
        ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
        "range": {
          "start": {
            "line": 54,
            "character": -1
          },
          "end": {
            "line": 54,
            "character": -1
          }
        }
      }
    ],
    "remedy": []
  },
  {
    "name": "",
    "description": "TODO",
    "source": [
```

```
      {
        "file":
         ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
        "range": {
          "start": {
            "line": 39,
            "character": -1
          },
          "end": {
            "line": 39,
            "character": -1
          }
        }
      }
    ],
    "remedy": []
  },
  {
    "name": "",
    "description": "TODO",
    "source": [
      {
        "file":
         ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
        "range": {
          "start": {
            "line": 84,
            "character": -1
          },
          "end": {
            "line": 84,
            "character": -1
          }
        }
      }
    ],
    "remedy": []
  },
  {
    "name": "",
    "description": "TODO",
    "source": [
      {
```

```
      "file":
       ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
      "range": {
        "start": {
          "line": 86,
          "character": -1
        },
        "end": {
          "line": 86,
          "character": -1
        }
      }
    }
  ],
  "remedy": []
},
{
  "name": "",
  "description": "TODO",
  "source": [
    {
      "file":
       ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
      "range": {
        "start": {
          "line": 81,
          "character": -1
        },
        "end": {
          "line": 81,
          "character": -1
        }
      }
    }
  ],
  "remedy": []
},
{
  "name": "",
  "description": "TODO",
  "source": [
    {
      "file":
       ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
```

```
      "range": {
        "start": {
          "line": 75,
          "character": -1
        },
        "end": {
          "line": 75,
          "character": -1
        }
      }
    }
  ],
  "remedy": []
},
{
  "name": "",
  "description": "TODO",
  "source": [
    {
      "file":
       ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
      "range": {
        "start": {
          "line": 84,
          "character": -1
        },
        "end": {
          "line": 84,
          "character": -1
        }
      }
    }
  ],
  "remedy": []
},
{
  "name": "",
  "description": "TODO",
  "source": [
    {
      "file":
       ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
      "range": {
        "start": {
```

```
          "line": 69,
          "character": -1
        },
        "end": {
          "line": 69,
          "character": -1
        }
      }
    }
  ],
  "remedy": []
},
{
  "name": "",
  "description": "TODO",
  "source": [
    {
      "file":
       ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
      "range": {
        "start": {
          "line": 84,
          "character": -1
        },
        "end": {
          "line": 84,
          "character": -1
        }
      }
    }
  ],
  "remedy": []
},
{
  "name": "",
  "description": "TODO",
  "source": [
    {
      "file":
       ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
      "range": {
        "start": {
          "line": 56,
          "character": -1
```

```
        },
        "end": {
          "line": 56,
          "character": -1
        }
      }
    }
  ],
  "remedy": []
},
{
  "name": "",
  "description": "TODO",
  "source": [
    {
      "file":
       ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
      "range": {
        "start": {
          "line": 84,
          "character": -1
        },
        "end": {
          "line": 84,
          "character": -1
        }
      }
    }
  ],
  "remedy": []
},
{
  "name": "",
  "description": "TODO",
  "source": [
    {
      "file":
       ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
      "range": {
        "start": {
          "line": 87,
          "character": -1
        },
        "end": {
```

```
            "line": 87,
            "character": -1
          }
        }
      }
    ],
    "remedy": []
  },
  {
    "name": "",
    "description": "TODO",
    "source": [
      {
        "file":
        ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
        "range": {
          "start": {
            "line": 69,
            "character": -1
          },
          "end": {
            "line": 69,
            "character": -1
          }
        }
      }
    ],
    "remedy": []
  },
  {
    "name": "",
    "description": "TODO",
    "source": [
      {
        "file":
        ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
        "range": {
          "start": {
            "line": 69,
            "character": -1
          },
          "end": {
            "line": 69,
            "character": -1
```

```
        }
      }
    }
  ],
  "remedy": []
},
{
  "name": "",
  "description": "TODO",
  "source": [
    {
      "file":
       ↪ "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
      "range": {
        "start": {
          "line": 54,
          "character": -1
        },
        "end": {
          "line": 54,
          "character": -1
        }
      }
    }
  ],
  "remedy": []
},
{
  "name": "",
  "description": "TODO",
  "source": [
    {
      "file":
       ↪ "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
      "range": {
        "start": {
          "line": 69,
          "character": -1
        },
        "end": {
          "line": 69,
          "character": -1
        }
      }
```

```
      }
    ],
    "remedy": []
  },
  {
    "name": "",
    "description": "TODO",
    "source": [
      {
        "file":
         ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
        "range": {
          "start": {
            "line": 88,
            "character": -1
          },
          "end": {
            "line": 88,
            "character": -1
          }
        }
      }
    ],
    "remedy": []
  },
  {
    "name": "",
    "description": "TODO",
    "source": [
      {
        "file":
         ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
        "range": {
          "start": {
            "line": 93,
            "character": -1
          },
          "end": {
            "line": 93,
            "character": -1
          }
        }
      }
    ],
```

```
  "remedy": []
},
{
  "name": "",
  "description": "TODO",
  "source": [
    {
      "file":
       ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
      "range": {
        "start": {
          "line": 78,
          "character": -1
        },
        "end": {
          "line": 78,
          "character": -1
        }
      }
    }
  ],
  "remedy": []
},
{
  "name": "",
  "description": "TODO",
  "source": [
    {
      "file":
       ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
      "range": {
        "start": {
          "line": 65,
          "character": -1
        },
        "end": {
          "line": 65,
          "character": -1
        }
      }
    }
  ],
  "remedy": []
},
```

```
{
  "name": "",
  "description": "TODO",
  "source": [
    {
      "file":
       ↪ "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
      "range": {
        "start": {
          "line": 84,
          "character": -1
        },
        "end": {
          "line": 84,
          "character": -1
        }
      }
    }
  ],
  "remedy": []
},
{
  "name": "",
  "description": "TODO",
  "source": [
    {
      "file":
       ↪ "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
      "range": {
        "start": {
          "line": 84,
          "character": -1
        },
        "end": {
          "line": 84,
          "character": -1
        }
      }
    }
  ],
  "remedy": []
},
{
  "name": "",
```

```
  "description": "TODO",
  "source": [
    {
      "file":
       ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
      "range": {
        "start": {
          "line": 87,
          "character": -1
        },
        "end": {
          "line": 87,
          "character": -1
        }
      }
    }
  ],
  "remedy": []
},
{
  "name": "",
  "description": "TODO",
  "source": [
    {
      "file":
       ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
      "range": {
        "start": {
          "line": 85,
          "character": -1
        },
        "end": {
          "line": 85,
          "character": -1
        }
      }
    }
  ],
  "remedy": []
},
{
  "name": "",
  "description": "TODO",
  "source": [
```

```
      {
        "file":
         ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
        "range": {
          "start": {
            "line": 86,
            "character": -1
          },
          "end": {
            "line": 86,
            "character": -1
          }
        }
      }
    ],
    "remedy": []
  },
  {
    "name": "",
    "description": "TODO",
    "source": [
      {
        "file":
         ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
        "range": {
          "start": {
            "line": 87,
            "character": -1
          },
          "end": {
            "line": 87,
            "character": -1
          }
        }
      }
    ],
    "remedy": []
  },
  {
    "name": "",
    "description": "TODO",
    "source": [
      {
```

```
      "file":
       ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
      "range": {
        "start": {
          "line": 88,
          "character": -1
        },
        "end": {
          "line": 88,
          "character": -1
        }
      }
    }
  ],
  "remedy": []
},
{
  "name": "",
  "description": "TODO",
  "source": [
    {
      "file":
       ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
      "range": {
        "start": {
          "line": 84,
          "character": -1
        },
        "end": {
          "line": 84,
          "character": -1
        }
      }
    }
  ],
  "remedy": []
},
{
  "name": "",
  "description": "TODO",
  "source": [
    {
      "file":
       ↪  "/workspaces/build/apps/examples/example1/annotated-working/example1.c",
```

```
        "range": {
          "start": {
            "line": 84,
            "character": -1
          },
          "end": {
            "line": 84,
            "character": -1
          }
        }
      }
    ],
    "remedy": []
  }
]
```

5.6 GAPS Enclave Definition Language (GEDL) File

5.6.1 Example GEDL

Below is an example `gedl` file generated by the gedl llvm pass. It contains type and size information about cross-domain calls and whether each argument or return is an input or output. The gedl is validated by the gedl schema when it is passed as input to the `idl_generator`.

A more detailed explanation of each field in the gedl is given here.

```
{"gedl": [
    {
        "caller": "enclave1",
        "callee": "enclave2",
        "calls": [
            {
                "func":     "sampleFunc",
                "return":   {"type": "double"},
                "clelabel": "enclave1",
                "params": [
                    {"type": "int", "name": "sampleInt", "dir": "in"},
                    {"type": "double", "name": "sampleDoubleArray",
                     ↪  "dir": "inout", "sz":15}
                ],
                "occurs": [
                    {"file": "/sample/Enclave1/Path/Enclave1File.c",
                     ↪  "lines": [44]},
```

```
                    {"file": "/sample/Enclave1/Path/Enclave1File2.c",
                     ↪  "lines": [15,205]}
                ]
            },
        {
                "func":     "sampleFunc2",
                "return":   {"type": "int"},
                "clelabel": "enclave1Extra",
                "params": [
                ],
                "occurs": [
                    {"file": "/sample/Enclave1/Path/Enclave1File.c",
                     ↪  "lines": 45}
                ]
            }
        ]
    },
        {
        "caller": "enclave2",
        "callee": "enclave3",
        "calls": [
            {
                "func":     "sampleFunc3",
                "return":   {"type": "void"},
                "clelabel": "enclave2",
                "params": [
                    {"type": "uint8", "name": "sampleUInt8", "dir":
                     ↪  "in"}
                ],
                "occurs": [
                    {"file": "/sample/Enclave1/Path/Enclave2File.c",
                     ↪  "lines": [55,87]}
                ]
            }
        ]
    }
]}
```

5.6.2 GEDL Format description

The top level key, contains one object for each unique ordered pair of enclaves with cross-domain calls. This is determined by checking which importedFunctions from an imported_func.txt file are also present in a defined_func.txt of a different enclave. These

files are generated by compiling the contents of the indicated directories into .ll files with names matching the directories, then running opt with -llvm-test and -prefix flags on each file.

Represented by a JSON array of objects of arbitrary size.

caller The name of the enclave making the cross-domain call. This will match the name of the directory containing the imported_func.txt file for the considered importedFunction.

Represented by a double quote (“) enclosed string that conforms to linux filename restrictions and in all lowercase.

callee The name of the enclave where the cross-domain call is defined. This will match the name of the directory containing the defined_func.txt file for the considered importedFuncion.

Represented by a double quote (“) enclosed string that conforms to linux filename restrictions and in all lowercase.

calls

An array containing one object for each cross-domain function called from “caller” and defined in “callee”. This is determined by creating an entry for each unique function in the “caller” imported_func.txt file that is present in “callee” defined_func.txt

- func: The function name of the cross-domain call. Determined by name in imported_func.txt. Represented by a double quote (“) enclosed string conforming to c/c++ function name restrictions.
- return: An object defining the type of the function’s return. Represented by a JSON object with a single key type
  - type: A variable type representing the type of the function’s return value. Determined by querying DIUtils.getDITypeName(), which uses debug information to check the return type of the function. Represented by a double quote (“) enclosed string that is one of IDL’s supported C types [double, ffloat, int8, uint8, int16, uint16, int32, uint32, int64, uint64] and not a pointer (no *).
- clelabel: String value denoting the CLE labels that are tainting the function. This is determined by checking the CLE labels present in the LLVM IR in the function definition. Represented as a string value, with plaintext labels separated by commas.
- params: Array containing one object for each argument passed to the function. Determined by querying PDGUtils for the list of arguments for the current function name. Represented as a JSON array of objects, each with keys type, name, dir, and optionally sz.
  - type: A variable type representing the type of the function’s return value following the same format as the return type.

- name: The argument name of the current argument. Determined by calling DIUtils.getArgName() which uses debug information to retrieve argument name. Represented by a double quote (“) enclosed string conforming to c/c++ argument name restrictions.
- dir: A string determining if read from or written to by the function to decide if it needs to be copied in/out. Determined by using arg.getAttribute() and checking if in, out, or both are attributes for arg. Represented by a double quote (“) enclosed string that is one of three values”in”, “out”, “inout”.
- sz: A number or word detailing the size of an array argument. Determined by using arg.getAttribute() and checking if count, size, string, or user_check are attributes for arg. Represented by an unsigned integer or a string that is either [string] or [user_check].
- occurs: Array containing one object for each callsite of function in “caller”. Determined by checking callsiteMap, a Map object created at beginning of AccessInfoTracker.cpp that maps every imported function to a Set of the files in the “caller” enclave where it is called. This is done by a module pass that examines the instructions of every function. Represented as a JSON array of objects, each with keys file and line.

- file: The path to a file in “caller” enclave containing calls to function. Determined by checking value of the current iterator on the Set returned by callsiteMap. Represented by a double quote (“) enclosed string that conforms to linux path restrictions and refers to a valid c/c++ file on the system.
- lines: The line numbers of lines where calls to the function are made in the current file. Determined by querying callsiteLines Map object created in the same manner as callsiteMap but recording lines. Represented by an array of unsigned integers which must not exceed the line count of the current file.

Input

A number of directories containing the *.c/h files for each enclave, These must be defined in the “enclaves” variable at the top of the Makefile.

Criteria

- No functions may have a pointer return type. Any functions with pointer returns must be refactored to instead return void and pass a new argument by reference that will act as the return
- No duplicate functions across domains, except for multithreaded programs where “main” can be duplicated
- Variables should not have any implicit casting to allow for automatic direction and size detection
- Arguments and return types must be be of IDL supported types {“double”,“ffloat”,“int8”,“uint8”,“int16”,“uint16”,“int32”,“uint32”,“int64”,“uint64”}

Warnings List

- Warning and terminating error if return type is a pointer or unsupported
- Warning if direction for an argument is undetermined
- Warning if size of an argument is undetermined
- Warning and if argument or type is not supported by IDL
- Warning and terminating error if function is defined in more than one domain (potentially more expensive than its worth)
- Warning if cross domain function does not have CLE label

5.6.3 GEDL schema

Below is the json schema used to validate the gedl file when using the `idl_generator`.

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "$id": "com.perspectalabs.gaps-closure.gedl",
    "$comment": "JSON schema for GEDL json definition",

    "definitions": {
        "typesEnum":{
            "$comment": "the posible supported variable types",
            "description": "[ENUM] variable type",
            "type": "string",
            "enum": [
                "double","float","char","unsigned
                ↪  char","short","unsigned short","int","unsigned
                ↪  int","long","unsigned long"
            ]
        },
        "directionEnum":{
            "description": "[ENUM] direction",
            "type": "string",
            "enum": [
                "in","inout","out"
            ]
        },
        "dynamicSizeEnum":{
            "description": "[ENUM] dynamic sizes",
            "type": "string",
            "enum":[
                "string"
            ]
```

```
},

"sizeInfo":{
    "description": "size information for arrays, (positive
    ↪ number)",
    "anyOf": [
        {
            "type": "integer",
            "minimum": 0
        },
        {
            "$ref": "#/definitions/dynamicSizeEnum"
        }
    ]
},

"cIdentifier":{
    "$comment": "Valid C identifier (funciton/variable/ect) as
    ↪ new versions support some utf8 filter is just checks we
    ↪ don't start with a number, and contain no spaces",
    "type": "string",
    "pattern": "^[^\\s\\d][^\\s]*$"
},

"paramsType":{
    "description": "Argument definition",
    "type": "object",
    "additionalProperties": false,
    "properties":{
        "type": {
            "$ref": "#/definitions/typesEnum"
        },
        "name":{
            "description": "function name",
            "$ref": "#/definitions/cIdentifier"
        },
        "dir":{
            "$ref": "#/definitions/directionEnum"
        },
        "sz":{
            "$ref": "#/definitions/sizeInfo"
        }
    }
},
```

```
"occursType":{
    "description": "Occurance Instance (callsite)",
    "type": "object",
    "additionalProperties": false,
    "properties":{
        "file": {
            "description": "Source file",
            "type": "string"
        },
        "lines": {
            "description": "Line of line numbers where calls to
             ↪  the function is made",
            "type": "array",
            "default": [],
            "minItems": 1,
            "items":{
                "description": "line number",
                "type": "integer",
                "minimum": 0
            }
        }
    }
},

"callsType":{
    "descripton": "calls object",
    "type": "object",
    "additionalProperties": false,

    "properties": {
        "func":{
            "description": "function name",
            "$ref": "#/definitions/cIdentifier"
        },
        "return":{
            "description": "return information",
            "type": "object",
            "additionalProperties": false,
            "required": ["type"],
            "properties": {
                "type": { "$ref": "#/definitions/typesEnum" }
            }
        },
```

```
        "params":{
            "description": "Array of parameter information",
            "type": "array",
            "uniqueItems": true,
            "default": [],
            "items": { "$ref": "#/definitions/paramsType" }
        },
        "occurs":{
            "description": "Array of callsite information",
            "type": "array",
            "uniqueItems": true,
            "default": [],
            "minItems": 1,
            "items": { "$ref": "#/definitions/occursType" }
        },
        "clelabel":{
            "description": "CLE Tag",
            "type": "string"
        }
    }
},

"gedlType":{
    "descripton": "A GEDL entry",
    "type": "object",
    "additionalProperties": false,

    "properties": {
        "$comment":{
            "description": "Optional comment entry",
            "type": "string"
        },
        "caller":{
            "description": "Name of the enclave making the
             ↪  cross-domain call",
            "type": "string",
            "minLength": 1
        },
        "callee":{
            "description": "Name of the enclave where the
             ↪  cross-domain call is defined",
            "type": "string",
            "minLength": 1
        },
```

```
                "calls": {
                    "description": "An array containing one object for
                     ↪  each cross-domain function",
                    "type": "array",
                    "minItems": 1,
                    "items": { "$ref": "#/definitions/callsType" }
                }
            }
        }
    },

    "type": "object",
    "required": ["gedl"],
    "description": "CLE Definition",
    "additionalProperties": false,

    "properties": {
        "$schema":{
            "description": "The gedl-schema reference (for standalone
             ↪  json files)",
            "type": "string"
        },
        "$comment":{
            "description": "Optional comment entry",
            "type": "string"
        },
        "gedl":{
            "description": "The array of gedl objects",
            "type": "array",
            "uniqueItems": true,
            "default": [],
            "items": { "$ref": "#/definitions/gedlType" }
        }
    }
}
```

5.7 Interface Definition Language (IDL)

The IDL is a text file with a set of C style struct data type definitions that is generated by the `idl_generator` when given a gedl file as its input.

Below is a sample idl which showcases a few structs. `FancyArrayTest` has all of the supported idl types as its fields.

```
/* Sample IDL file */

struct Position {
  double x;
  double y;
  double z;
};

struct Distance {
  double dx;
  double dy;
  double dz;
};

struct ArrayTest {
  double doubloons [3];
};

struct FancyArrayTest {
  char           a;
  unsigned char  b;
  short          c;
  unsigned short d;
  int            e;
  unsigned int   f;
  long           g;
  unsigned long  h;
  float          i;
  double         j;
  char           k[4];
  unsigned char  l[4];
  short          m[4];
  unsigned short n[4];
  int            o[4];
  unsigned int   p[4];
  long           q[4];
  unsigned long  r[4];
  float          s[4];
  double         t[4];
};
```

5.8 Serialization CODECs

Below is a codec.c that is generated from the `rpc_generator` during autogeneration. The codec contains a print, encode and decode function for each request and response. This codec.c is taken from example1, so the relevant functions will be for `get_a` and the functions shown are for the `get_a` request. (Note: the corresponding response functions are omitted).

```
// ...

void request_get_a_print (request_get_a_datatype *request_get_a) {
  fprintf(stderr, "request_get_a(len=%ld): ", sizeof(*request_get_a));
  fprintf(stderr, " %d,", request_get_a->dummy);
  fprintf(stderr, " %u, %u, %u, %hu, %hu\n",
          request_get_a->trailer.seq,
          request_get_a->trailer.rqr,
          request_get_a->trailer.oid,
          request_get_a->trailer.mid,
          request_get_a->trailer.crc);
}

void request_get_a_data_encode (void *buff_out, void *buff_in, size_t
 ↪  *len_out) {
  request_get_a_datatype *p1 = (request_get_a_datatype *) buff_in;
  request_get_a_output   *p2 = (request_get_a_output *)   buff_out;
  p2->dummy = htonl(p1->dummy);
  p2->trailer.seq = htonl(p1->trailer.seq);
  p2->trailer.rqr = htonl(p1->trailer.rqr);
  p2->trailer.oid = htonl(p1->trailer.oid);
  p2->trailer.mid = htons(p1->trailer.mid);
  p2->trailer.crc = htons(p1->trailer.crc);
  *len_out = sizeof(int32_t) + sizeof(trailer_datatype);
}

void request_get_a_data_decode (void *buff_out, void *buff_in, size_t
 ↪  *len_in) {
  request_get_a_output   *p1 = (request_get_a_output *)   buff_in;
  request_get_a_datatype *p2 = (request_get_a_datatype *) buff_out;
  p2->dummy = ntohl(p1->dummy);
  p2->trailer.seq = ntohl(p1->trailer.seq);
  p2->trailer.rqr = ntohl(p1->trailer.rqr);
  p2->trailer.oid = ntohl(p1->trailer.oid);
  p2->trailer.mid = ntohs(p1->trailer.mid);
  p2->trailer.crc = ntohs(p1->trailer.crc);
```

```
}

// ...
```

5.9 ESCAPE Benchmarking

This section describes performance benchmarking of the Intel ESCAPE GAPS Security Engine and investigates the fundamental throughput capability of Shared Memory to communicate among applications. It uses a benchmarking tool to collect throughput for different: a) types of memory, b) copy sizes, c) copy functions, and d) number of parallel threads. These results are then collated by a plotting script to generate performance plots.

5.9.1 ESCAPE GAPS Security Engine

The Intel ESCAPE [1] GAPS Security Engine evaluation system consists of two Xeon CPU hosts (host0 and host1) connected over Intel Ultra Path Interconnect (UPI) links to a Stratix 10 FPGA. The two ESCAPE hosts run Ubuntu 20.04.1 OS with 130 GB of local DDR4 memory, connected over a 64-bit memory bus with a bandwidth of 2933 MegaTransfers per second (MT/s). Each laptop therefore has a local memory bandwidth of 2.933 x 8 = 23.46 GB/s.

[1] Intel, "Extended Secure Capabilities Architecture Platform and Evaluation (ESCAPE) System Bring Up Document," February 17, 2022.

By editing the GRUB boot loader, each Xeon host maps the 16 GB of FPGA physical memory so it appears after its own local physical memory. Thus, each host has 130 + 16 = 146 GB of physical memory, whose physical addresses are as shown in the figure below. To provide GAPS security between enclaves, the FPGA uses an address-filtering mechanism similar to a virtual memory page table. The filtering rules allow or disallow reads and writes from either host to the mediated DDR3 memory pool of 16 GB.
For the benchmarking, the 16GB FPGA memory was also split into 14 GB of shared memory and 1 GB of private memory for each host.

If process A on ESCAPE host0 and process B on ESCAPE host1 memory-maps the shared FPGA memory (e.g., using the mmap() call in a C program) then, as shown below, the virtual address space of each process will include the same shared FPGA memory. The process A on host0 and the process B on host1 can thus both access the same FPGA memory using memory copy instructions allowing inter-host communication.

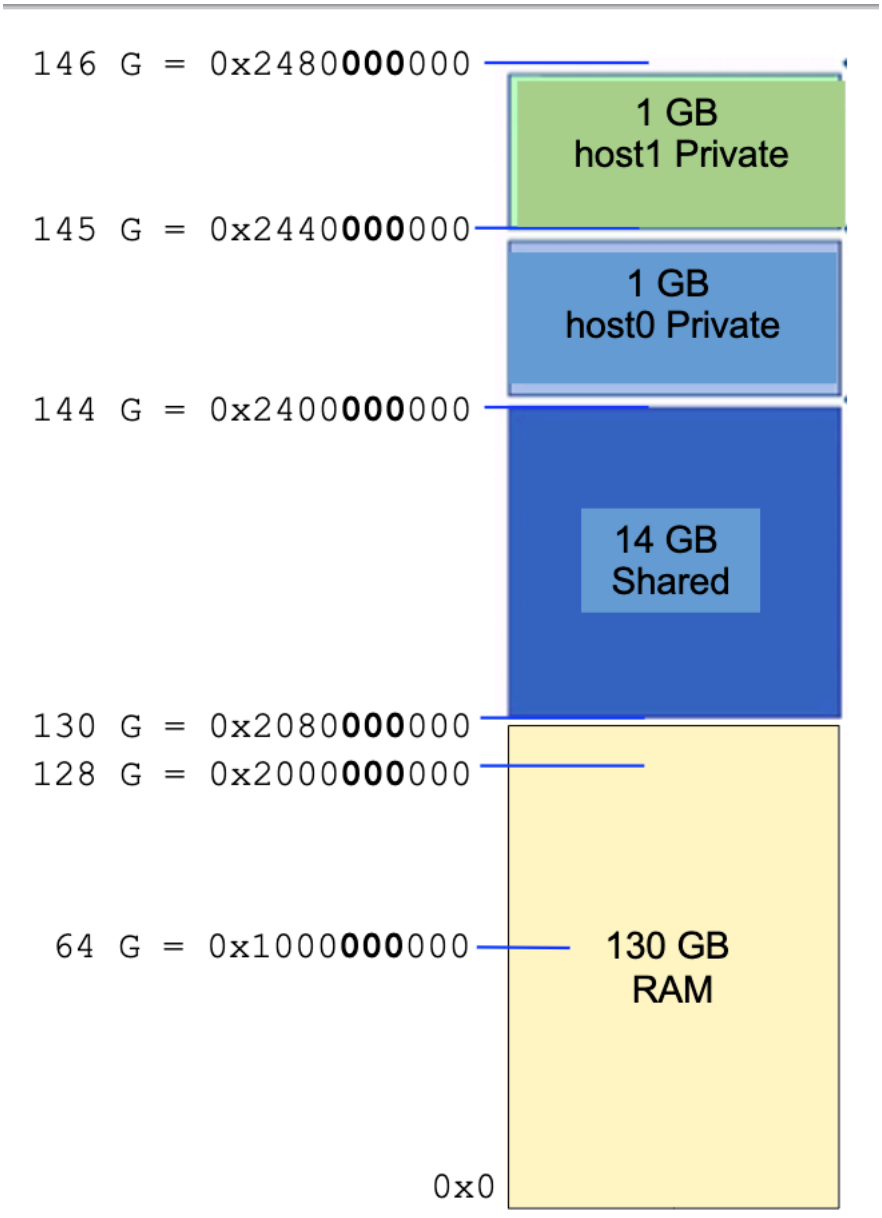


Figure 18: ESCAPE host physical memory

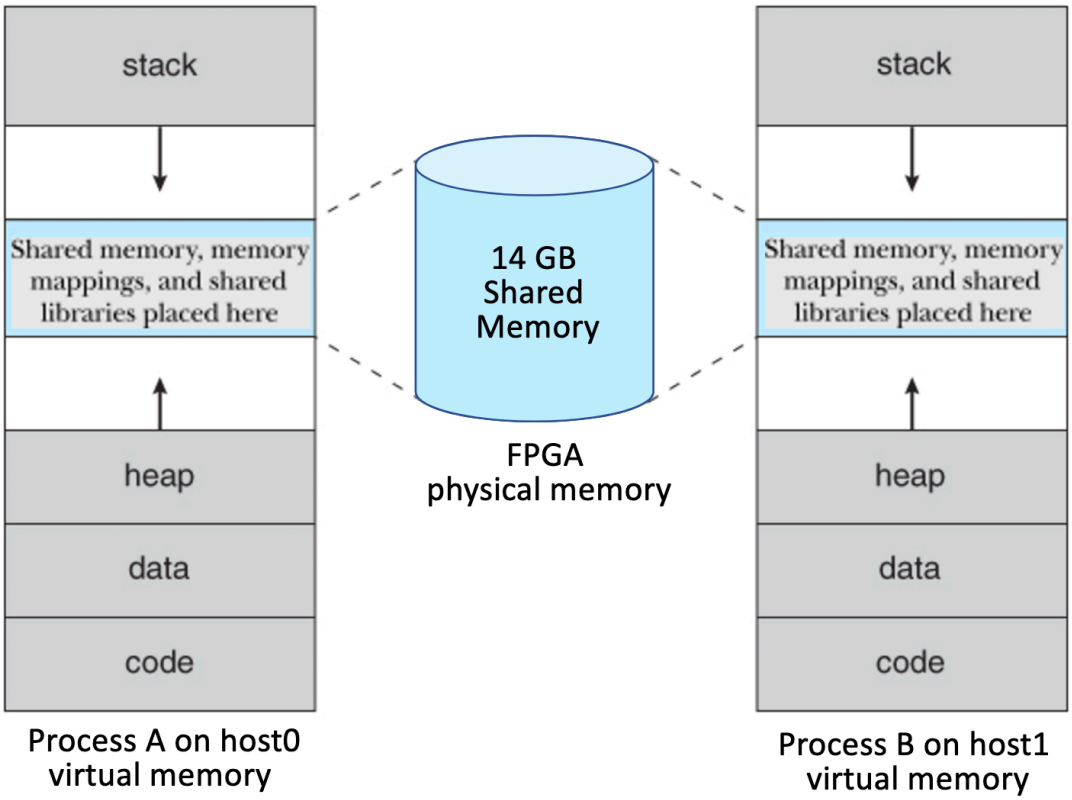


Figure 19: ESCAPE host virtual memory of two local processes

5.9.2 Inter-Process Communication using Shared Memory

To measure raw shared memory performance without the ESCAPE board, we also benchmarked shared memory performance between two processes on the same host. If a process A and process B both on host0 memory-map part of host0's physical memory, then, as shown below, the virtual address space of each process will include the same portion of shared FPGA memory.

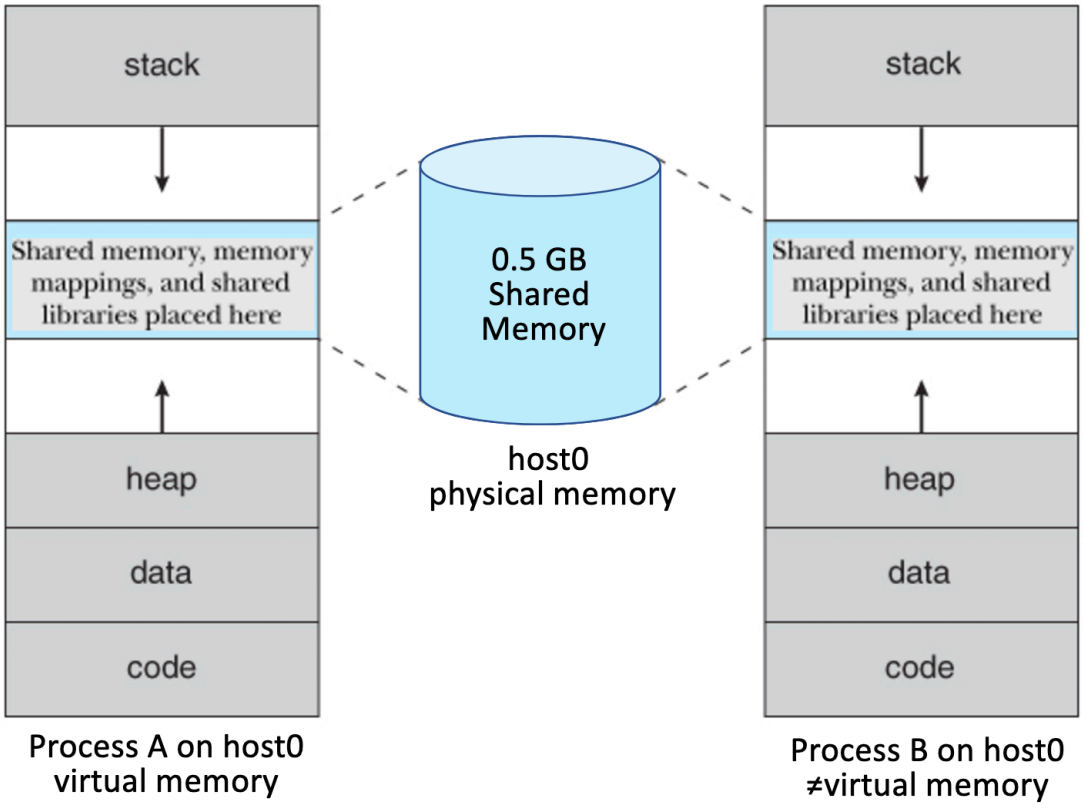


Figure 20: Inter-process Virtual Memory

5.9.3 Benchmarking Tool

The Benchmarking tool is a C program in the CLOSURE github repository with two main files:

- The main program memcpy_test.c that runs through the testing parameter combinations.
- The worker thread pool thread_pool.c that creates and manages the worker threads that perform the copying.

5.9.3.1 Benchmarking Tool Parameters The benchmarking tool provides experiment setup options that can be discovered using the help option as shown below:

```
amcauley@jaga:~/gaps/build/hal/escape/perftests$ ./memcpy_test -h
Shared Memory performance test tool for GAPS CLOSURE project
Usage: [sudo] ./memcpy_test [OPTIONS]... [Experiment ID List]
'sudo' required if using shared memory: /dev/mem (Experiment ID 2, 3, 4, 5)
OPTIONS:
   -i : which source data is initialized
   0 = all sources (default) - both when application or shared memory is the source of da
   1 = only if source is application - use current shared memory content as source (so ca
   -n : number of length tests (default=9, maximum = 10)
   -o : source data initialization offset value (before writing)
   -r : number of test runs per a) memory pair type, b) payload length and c) copy functi
   -t : number of worker threads in thread pool (default=0)
   -z : read/write with given number of second sleep between each (default z=0)
Experiment ID List (default = all):
   0 = tool writes to host heap
   1 = tool reads from host heap
   2 = tool writes to host mmap
   3 = tool reads from host mmap
   4 = tool writes to shared escape mmap
   5 = tool reads from shared escape mmap
EXAMPLES:
   sudo ./memcpy_test
   ./memcpy_test 0 1 -r 1000 -n 2
```

5.9.3.2 Benchmarking Tool Variables The benchmarking tool runs a series of throughput performance measurements for combinations of four variables:

1) memory type, 2) payload length, 3) copy function, and 4) number of worker threads.

5.9.3.2.1 Variable 1) Memory Types The Benchmarking tool copies data between its local heap memory created using malloc() and three types of memory:

- **Host heap**: Allocates memory from the host using malloc(). This test allows measuring raw memory bandwidth, but only allows a single process to write or read the memory.
- **Host mmap**: Allocates memory by opening /dev/mem followed by an mmap() of host memory. The resultant mapped memory can be used to communicate data between two independent processes on the same host.

- **ESCAPE mmap**: Allocates memory by opening /dev/mem followed by an mmap() of FPGA memory. For the ESCAPE system this starts at address 130 GB = 0x2080000000 (as described above), though this address can be changed by modifying the MMAP_ADDR_ESCAPE definition in memcpy_test.c.

By default, in one run, the benchmark tool reads and writes to each of the three types of memory, creating a total of 2 x 3 = 6 different experiments. Each experiment has an ID from 0 to 5, allowing the user to list of IDs to specify any subset of the six experiments to run.

5.9.3.2.2 Variable 2) Payload Lengths For each of the six memory write/read types in a run, the benchmark tool measures the throughput with a series of increasing payload lengths (number of bytes of data that are written or read). It's default is to test with 10 different payload lengths from 16 bytes up to 16 MB (except for *host mmap* memory, which linux limits to 512 KB). However, the number of lengths test can be reduced using the '-n' parameter option when calling the tool. (e.g., *./memcpy_test -n 3* will run just the first 3 payload lengths). The payload length values themselves can be modified by changing the definitions of the global array *copy_size_list* in memcpy_test.c.

5.9.3.2.3 Variable 3) Copy Functions For each payload length in a run, the benchmarking tool uses the copy functions to read or write data between the tool and memory. It currently tests with three different copy functions:

- **glibc memory copy**: memcpy().
- **naive memory copy**: using incrementing unsigned long C pointers that point to the destination and source ($d{+}{+} = s{+}{+}$).
- **Apex memory copy**: A fast memory copy , with available source code.

The benchmark tool runs the test multiple times for each copy function tested. The default number of tests is 5, but the value can be changed by specifying the '-r' parameter option when calling the benchmark tool (larger values provide more reliable throughput measurements, but small values can run much faster for initial testing.

5.9.3.2.4 Variable 4) Number of Worker Threads The each run the user can specify the number of parallel worker threads that copy the data using the '-t' parameter option when calling the tool. For example, *./memcpy_test -t 16* will run just with 16 parallel worker threads.

5.9.3.3 Clone, Compile and Run Benchmarking Tool The benchmarking tool is part of the HAL branch along with the benchmark plotting script, with the latest version (including the multi-threaded option) found in the git multi-threaded branch:

```
git clone git@github.com:gaps-closure/hal.git
cd hal/escape/perftests
git checkout multi-threaded
```

To compile the benchmarking tool we run make:

```
make
```

Below are a few examples of running with the benchmarking tool:

```
Run with default parameters:
  sudo ./memcpy_test

Run at a high priority:
  sudo nice --20 ./memcpy_test

Run only the first two memory types (write and read to heap memory)
  sudo ./memcpy_test 0 1

Run test with greater sampling (average over 1000 runs instead of 5)
  sudo ./memcpy_test -r 1000

Run just for few smaller lengths
  sudo ./memcpy_test -n 3
```

The results for each input variable are printed on the terminal. For example:

```
$ sudo ./memcpy_test 0 1 -r 1000 -n 2
PAGE_MASK=0x00000fff data_off=0 source_init=0 payload_len_num=2 runs=1000 thread count=0
App Memory uses host Heap [len=0x10000000 Bytes] at virtual address 0x7fb55c46c010
--------------------------------------------------------------------------------
    sour data [len=0x10000000 bytes]: 0x fffefdfcfbfaf9f8 f7f6f5f4f3f2f1f0 ... 171615141
0) App writes to host-heap (fd=-1, vir-addr=0x7fb54c46b010, phy-addr=0x0, len=268.435456
      16 bytes using glibc_memcpy =   5.785 GB/s (1000 runs: ave delta = 0.000000003 secs
      16 bytes using naive_memcpy =   9.473 GB/s (1000 runs: ave delta = 0.000000002 secs
      16 bytes using  apex_memcpy =   5.382 GB/s (1000 runs: ave delta = 0.000000003 secs
     256 bytes using glibc_memcpy =  49.089 GB/s (1000 runs: ave delta = 0.000000005 secs
     256 bytes using naive_memcpy =  23.610 GB/s (1000 runs: ave delta = 0.000000011 secs
     256 bytes using  apex_memcpy =  29.190 GB/s (1000 runs: ave delta = 0.000000009 secs
    dest data [len=0x100 bytes]: 0x fffefdfcfbfaf9f8 f7f6f5f4f3f2f1f0 ... 171615141312111
Deallocating memory: fd=-1 pa_virt_addr=0x7fb54c46b010 pa_map_len=0x10000000 mem_typ_pair
run_per_mem_type_pair Done
```

```
--------------------------------------------------------------------------------
    sour data [len=0x10000000 bytes]: 0x fffefdfcfbfaf9f8 f7f6f5f4f3f2f1f0 ... 1716151413
1) App reads from host-heap (fd=-1, vir-addr=0x7fb54c46b010, phy-addr=0x0, len=268.435456
      16 bytes using glibc_memcpy =   5.440 GB/s (1000 runs: ave delta = 0.000000003 secs
      16 bytes using naive_memcpy =   8.879 GB/s (1000 runs: ave delta = 0.000000002 secs
      16 bytes using  apex_memcpy =   6.159 GB/s (1000 runs: ave delta = 0.000000003 secs
     256 bytes using glibc_memcpy =  48.688 GB/s (1000 runs: ave delta = 0.000000005 secs
     256 bytes using naive_memcpy =  23.878 GB/s (1000 runs: ave delta = 0.000000011 secs
     256 bytes using  apex_memcpy =  29.314 GB/s (1000 runs: ave delta = 0.000000009 secs
    dest data [len=0x100 bytes]: 0x fffefdfcfbfaf9f8 f7f6f5f4f3f2f1f0 ... 17161514131211
Deallocating memory: fd=-1 pa_virt_addr=0x7fb54c46b010 pa_map_len=0x10000000 mem_typ_pair
```

The tool put the tabulated results (used by the plotting script)into a single csv file: by default *results.csv* The first line describes the content of each of the six columns and the remaining lines gives the variable values and performance for each run. Below shows an example of initial lines of a run:

```
$ head -5 results.csv
  Experiment Description, Copy length (Bytes), Copy Type, Throughput (GBps), Number of Ru
  App writes to host-heap,16,glibc_memcpy,5.785,1000,0
  App writes to host-heap,16,naive_memcpy,9.473,1000,0
  App writes to host-heap,16, apex_memcpy,5.382,1000,0
  App writes to host-heap,256,glibc_memcpy,49.089,1000,0
```

5.9.4 Benchmark Plotting Script

The benchmark plotting script is located in the same directory as the benchmarking tool: [see above](#clone,-compile-and-run-benchmarking-tool] The script, plot_xy.py , uses the csv file outputs of the benchmarking tool (see above).

The script plots 3-dimensional plots with different x, y and z variables. The three plots in the results section below can be generated by uncommenting the final three lines of the plot_xy.py script (by default the script only creates the first plot). More plots with different x, y and z variables can be generated by adding lines at the end of the script.

5.9.5 RESULTS

We used the benchmarking script to test the performance of:

- Intel ESCAPE GAPS Security Engine (ESCAPE FPGA mmap'ed shared memory) to communicate between two hosts.
- Host shared Memory (Host mmap'ed) to communicate among independent applications on the same host

- Host heap memory used by a single application

For each we looked at different: a) memory copy functions (glibc, naïve, apex),

b) lengths of data, and c) number of worker threads

This section summarizes these results on the ESCAPE testbed with the three x, y, z plots:

A) The first (default) plot has:

- x-axis: Number worker threads.
- y-axis: Throughput.
- z-axis: Data copy length.

It shows that writing to host-heap can achieve the server memory B/W limit even without threads, but ESCAPE is two order of magnitude slower even with 64 threads

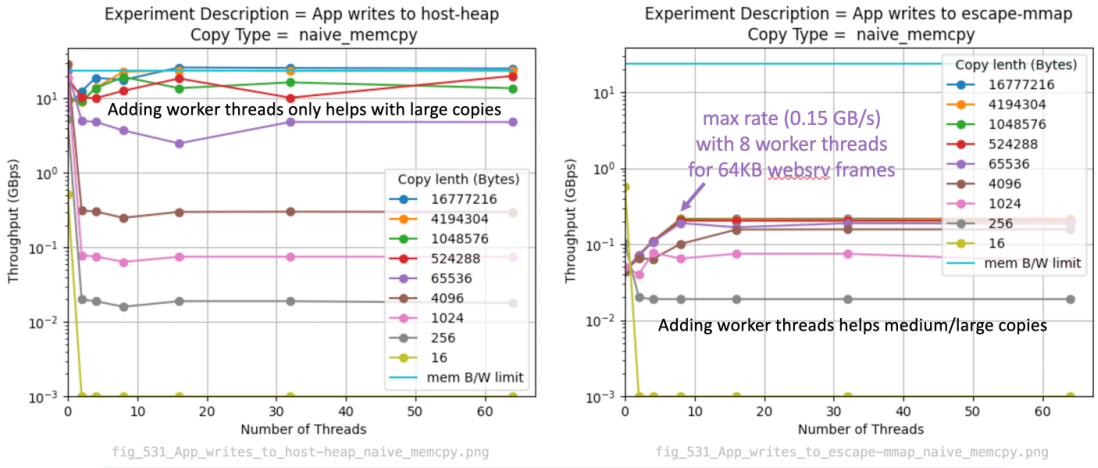


Figure 21: ESCAPE Throughput versus Threads for Different Data Lengths

B) An second plot has:

- x-axis: Number worker threads.
- y-axis: Throughput.
- z-axis: Type of memory copy function.

It shows that adding worker threads significantly helps mmapp'ed copy performance, but does not help significantly above 8 threads.

C) An final alternative plot has:

- x-axis: Data copy length.

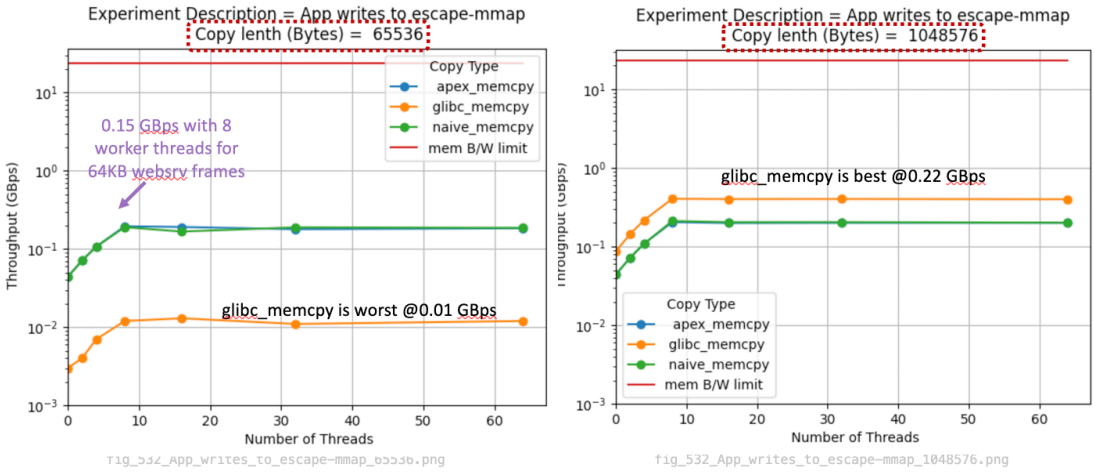


Figure 22: ESCAPE Throughput versus Threads for Different Copy Functions

- y-axis: Throughput.
- z-axis: Type of memory copy Function.

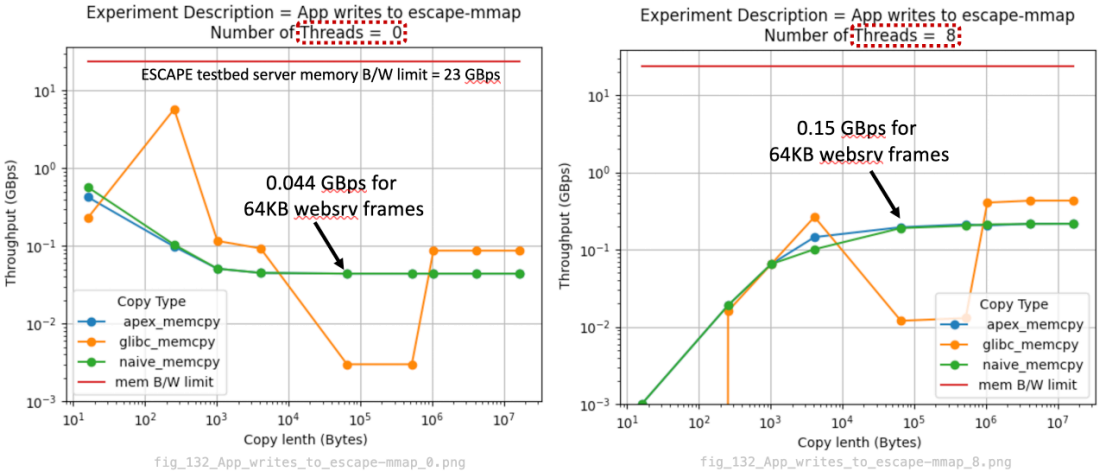


Figure 23: ESCAPE Throughput versus Threads for Different Copy Functions

It shows that copy function type can significantly impact performance. The winner between glibc and naïve/apex memcpy varies significantly based on the copy length

5.9.6 CONCLUSION

We are able to achieve 150 MB/s to communicate 64 KB frames between two hosts using the ESCAPE shared memory. Although fast, this was less than initially expected based

on simply using memory copies. Conventional wisdom holds mmap() should outperform traditional file I/O:

- Access pages via pointers using fast load/store (rather than read/write) as if file resided entirely in memory.
- No user buffer pool: kernel transparently moves data between device and memory for page fault evicts old pages.
- mmap() removes the system call per I/O and pointers void extra copy between kernel and user space a buffer.
- Removes need to serialize/deserialize data using the same in-memory and persistent formats.

We were able to achieve the memory bandwidth limit only when not memory mapping the files (see graphs above). The reason for the difference is explained by recent papers looking at memory-mapped I/O:

- Default memory-mapped I/O path does not scale well with the number of cores:
  - Transparent paging means OS can flush page to secondary storage at any time -causing blocking
  - Propose Fastpath to ameliorate problems with Linux's mmap()
  - [PAP20] A. Papagiannis, G. Xanthakis, G. Saloustros, M. Marazakis, A. Bilas, "Optimizing Memory-mapped I/O for Fast Storage Devices," Proceedings of the USENIX Annual Technical Conference, July 2020.
- Even with additional complexity mmap() will not scale:
  - Slow and unpredictable page fault eviction, TLB shoot-downs and other hidden issues
  - Experiments shows that adding more threads ineffective beyond about 8 threads, which our results confirmed.
  - [CRO22] A. Crotty, V. Leis, A. Pavlo, "Are You Sure You Want to Use MMAP in Your Database Management System?" 12th Annual Conference on Innovative Data Systems Research (CIDR '22) , Chaminade, USA, Jan 2022.

CRO22 argues that TLB shootdowns have the biggest impact. When evicting pages, OS removes mappings from both page table and each CPU's TLB Flushing local TLB of the initiating core is straightforward, but the OS must ensure that no stale entries remain in the TLBs of remote cores. Unfortunately, current CPUs do not provide coherence for remote TLBs, the OS has to issue an expensive inter-processor interrupt to flush them, called a TLB shootdown.

- [BLA89] D. Black, R. Rashid, D. Golub, C. Hill, R. Baron, "Translation Lookaside Buffer Consistency: A Software Approach" In ASPLOS, pages 113–122, 1989.

5.10 Autogenerated RPC Code

The RPC generator (*rpc_generator.py*) adds RPC code for each cross domain function into the partitioned code for each enclave. The RPC functions use the HAL API, with optional ARQ for enhanced reliability, creating a unique *tag=<mux,sec,typ>* information per function per unidirectional flow.

The RPC generator uses the following inputs:

- **PARTITIONED APPLICATION CODE** for each enclave (e.g., partitioned/multithreaded/orange/example1.c)
- **GEDL FILE** specifying the cross domain functions information, with signatures of functions to be wrapped in cross-domain RPC (e.g., partitioned/multithreaded/example1.gedl). The JSON formatted file also has information about return value type (e.g., int or double) and function parameters information (type, name, direction (inputs/outputs/both) and array size). Currently, function arguments must be primitive types or fixed size arrays of primitive types. The GEDL file also has the line numbers where the function is found in the partitioned code.
- **CLE SCHEMA FILE** (e.g., mules/cle-spec/schema/cle-schema.json) which includes default values for user annotations. It allows specifying default ARQ parameters (applied uniformly to all functions), such as number of RPC retries and timeout value.
- **USER CLE MAP FILE** (e.g., partitioned/multithreaded/example1.all.clemap.json) giving the user specified annotations. For example, it can contain user specified ARQ parameter values for each function.

The RPC generator creates:

- **PARTITIONED APPLICATION CODE MODIFICATIONS** adding a HAL init and RPC headers to main program. It replaces cross domain calls (on lines specified by the GEDL file) with a call to the the generated RPC code. For example, it will replace function call foo() with _rpc_foo(). On the side, without the main, it create a main program and a handler loop
- **RPC CODE** for each enclave. For example it may create orange_rpc.{c,h} for the orange enclave and purple_rpc.{c,h} for the purple enclave. The rpc code in each enclave communicates with remote enclave using the HAL API, with the addition of custom ARQ to support more reliable RPC request and reply communications.
- **HAL Cross Domain Configuration File** (xdconf.ini), which contains the information the assignment of tags <mux, sec, type> for each functions unidirectional flow between enclave pairs.

The RPC generator has modes of operation instantiated using C preprocessor macros for conditional compilation (e.g„ vscode Makefile CFLAGS, CLAG_FLAGS or

IPC_MODE). Each of the three modes of operation select between two RPC generator configuration options:

1) **singlethreaded vs. multithreaded (default) Receiver**

   - *multithreaded* provides one RPC listener/subscriber thread per XD function. The RPC protocol is simply get request, run function, send response
   - *singlethreaded* is a special case for completely singlethreaded programs. It adds an extra message exchange (NEXT/OKAY) for single listener thread:
     - callee (listener thread) waits for nextrpc message
     - caller first sends nextrpc type
     - callee (listener thread) sends okay
     - callee (listener thread) waits for message specified in prev nextrpc
     - caller sends actual request
     - callee gets requests, runs function, send response, then waits waits for next nextrpc message

2) **Receive Sockets are one per APP vs. one per function**

   - *One socket per APP* design (legacy) is based on one persistent listener that handled all cross-domain messages. This thread opened a zeromq socket once and reused it for the life of the program. This is efficient but not thread-safe
   - *One socket per Function design (my_xdc). The socket is opened and closed just in time within the thread in question. This is less efficient, but thread-safe (e.g., for a web application framework that assigned each HTTP request to an arbitrary thread).

3. **Request-Response vs One-way Diode** The requestor can send a call (with parameters) and either: a) wait for a response (with results) or b) continue without waiting for a response. In both cases the responder will run the cross-domain function and will either: a) send the result back to the requestor (Request-Response mode) or will not send any response (One-way Diode).

The following are the rpc generated functions for `example1`. Recall that in `example1` there is a single function `get_a` with no arguments in level orange that is called from an enclave in level purple.

In `example1.c` for purple code, the call to `get_a` is replaced with an rpc call

```
// ...
#pragma cle begin EWMA_MAIN
int ewma_main() {
#pragma cle end EWMA_MAIN
  double x;
  double y;
```

```
#pragma cle begin PURPLE
  double ewma;
#pragma cle end PURPLE
  for (int i=0; i < 10; i++) {
    x = _err_handle_rpc_get_a(); // was get_a()
    y = get_b();
    ewma = calc_ewma(x,y);
    printf("%f\n", ewma);
  }
  return 0;
}
// ...
```

In `purple_rpc.c`, this function has the following definition, which makes a call onto the network using 0mq.

```
#pragma cle begin RPC_GET_A
double _rpc_get_a(int *error) {
#pragma cle end RPC_GET_A
    gaps_tag t_tag;
    gaps_tag o_tag;
#ifndef __LEGACY_XDCOMMS__
    my_tag_write(&t_tag, MUX_REQUEST_GET_A, SEC_REQUEST_GET_A,
 ↪  DATA_TYP_REQUEST_GET_A);
#else
    tag_write(&t_tag, MUX_REQUEST_GET_A, SEC_REQUEST_GET_A,
 ↪  DATA_TYP_REQUEST_GET_A);
#endif /* __LEGACY_XDCOMMS__ */
#ifndef __LEGACY_XDCOMMS__
    my_tag_write(&o_tag, MUX_RESPONSE_GET_A, SEC_RESPONSE_GET_A,
 ↪  DATA_TYP_RESPONSE_GET_A);
#else
    tag_write(&o_tag, MUX_RESPONSE_GET_A, SEC_RESPONSE_GET_A,
 ↪  DATA_TYP_RESPONSE_GET_A);
#endif /* __LEGACY_XDCOMMS__ */
    static int req_counter = INT_MIN;
    static double last_processed_result = 0;
    static int last_processed_error = 0;
    static int inited = 0;
#ifndef __LEGACY_XDCOMMS__
    void *psocket;
    void *ssocket;
#else
```

```
    static void *psocket;
    static void *ssocket;
#endif /* __LEGACY_XDCOMMS__ */
    #pragma cle begin TAG_REQUEST_GET_A
    request_get_a_datatype request_get_a;
    #pragma cle end TAG_REQUEST_GET_A
    #pragma cle begin TAG_RESPONSE_GET_A
    response_get_a_datatype response_get_a;
    #pragma cle end TAG_RESPONSE_GET_A
    double result;
#ifndef __LEGACY_XDCOMMS__
    codec_map  mycmap[DATA_TYP_MAX];
    for (int i=0; i < DATA_TYP_MAX; i++)  mycmap[i].valid=0;
    my_xdc_register(request_get_a_data_encode,
 ↪  request_get_a_data_decode, DATA_TYP_REQUEST_GET_A, mycmap);
    my_xdc_register(response_get_a_data_encode,
 ↪  response_get_a_data_decode, DATA_TYP_RESPONSE_GET_A, mycmap);
#endif /* __LEGACY_XDCOMMS__ */
#ifndef __LEGACY_XDCOMMS__
    void * ctx = zmq_ctx_new();
    psocket = my_xdc_pub_socket(ctx, (char *)OUTURI);
    ssocket = my_xdc_sub_socket_non_blocking(o_tag, ctx, 1000,
 ↪  (char*)INURI);
    sleep(1); /* zmq socket join delay */
#else
    if (!inited) {
        inited = 1;
        psocket = xdc_pub_socket();
        ssocket = xdc_sub_socket_non_blocking(o_tag, 1000);
        sleep(1); /* zmq socket join delay */
    }
#endif /* __LEGACY_XDCOMMS__ */
    request_get_a.dummy = 0;
    request_get_a.trailer.seq = req_counter;
#ifndef __LEGACY_XDCOMMS__
    if (req_counter == INT_MIN) {
        int tries_remaining = 30;
        while(tries_remaining != 0){
            my_xdc_asyn_send(psocket, &request_get_a, &t_tag , mycmap);
#ifndef __ONEWAY_RPC__
            *error = my_xdc_recv(ssocket, &response_get_a, &o_tag ,
             ↪  mycmap);
            fprintf(stderr, "REQU get_a: ReqId=%d error=%d tries=%d ",
 ↪  request_get_a.trailer.seq, *error, tries_remaining);
```

```
            if (*error > 0) fprintf(stderr, "ResId=%d Reserr=%d ",
             ↪  response_get_a.trailer.seq >> 2,
             ↪  (response_get_a.trailer.seq >> 1) & 0x01);
            fprintf(stderr, "t_tag=<%02u, %02u, %02u>, ", t_tag.mux,
 ↪  t_tag.sec, t_tag.typ);
            fprintf(stderr, "o_tag=<%02u, %02u, %02u>\n", o_tag.mux,
 ↪  o_tag.sec, o_tag.typ);
            if (*error == -1){
                tries_remaining--;
                continue;
            }
#else
            *error = 0;
            fprintf(stderr, "REQU get_a: ReqId=%d error=%d tries=%d ",
 ↪  request_get_a.trailer.seq, *error, tries_remaining);
            if (*error > 0) fprintf(stderr, "ResId=%d Reserr=%d ",
             ↪  response_get_a.trailer.seq >> 2,
             ↪  (response_get_a.trailer.seq >> 1) & 0x01);
            fprintf(stderr, "t_tag=<%02u, %02u, %02u>, ", t_tag.mux,
 ↪  t_tag.sec, t_tag.typ);
            fprintf(stderr, "o_tag=<%02u, %02u, %02u>\n", o_tag.mux,
 ↪  o_tag.sec, o_tag.typ);
#endif /* __ONEWAY_RPC__ */
            break;  /* Reach here if received a response or
             ↪  __ONEWAY_RPC__ */
        }
#ifndef __ONEWAY_RPC__
        if (*error >= 0) req_counter = 1 + (response_get_a.trailer.seq
         ↪  >> 2);
#else
        req_counter++;
#endif /* __ONEWAY_RPC__ */
    }
#else /* __LEGACY_XDCOMMS__ */
    if (req_counter == INT_MIN) {
        int tries_remaining = 30;
        while(tries_remaining != 0){
            xdc_asyn_send(psocket, &request_get_a, &t_tag);
#ifndef __ONEWAY_RPC__
            *error = xdc_recv(ssocket, &response_get_a, &o_tag);
            fprintf(stderr, "REQU get_a: ReqId=%d error=%d tries=%d ",
 ↪  request_get_a.trailer.seq, *error, tries_remaining);
```

```
            if (*error > 0) fprintf(stderr, "ResId=%d Reserr=%d ",
             ↪  response_get_a.trailer.seq >> 2,
             ↪  (response_get_a.trailer.seq >> 1) & 0x01);
            fprintf(stderr, "t_tag=<%02u, %02u, %02u>, ", t_tag.mux,
 ↪  t_tag.sec, t_tag.typ);
            fprintf(stderr, "o_tag=<%02u, %02u, %02u>\n", o_tag.mux,
 ↪  o_tag.sec, o_tag.typ);
            if (*error == -1){
                tries_remaining--;
                continue;
            }
#else
            *error = 0;
            fprintf(stderr, "REQU get_a: ReqId=%d error=%d tries=%d ",
 ↪  request_get_a.trailer.seq, *error, tries_remaining);
            if (*error > 0) fprintf(stderr, "ResId=%d Reserr=%d ",
             ↪  response_get_a.trailer.seq >> 2,
             ↪  (response_get_a.trailer.seq >> 1) & 0x01);
            fprintf(stderr, "t_tag=<%02u, %02u, %02u>, ", t_tag.mux,
 ↪  t_tag.sec, t_tag.typ);
            fprintf(stderr, "o_tag=<%02u, %02u, %02u>\n", o_tag.mux,
 ↪  o_tag.sec, o_tag.typ);
#endif /* __ONEWAY_RPC__ */
            break;  /* Reach here if received a response or
             ↪  __ONEWAY_RPC__ */
        }
#ifndef __ONEWAY_RPC__
        if (*error >= 0) req_counter = 1 + (response_get_a.trailer.seq
         ↪  >> 2);
#else
        req_counter++;
#endif /* __ONEWAY_RPC__ */
    }
#endif /* __LEGACY_XDCOMMS__ */
    request_get_a.dummy = 0;
    request_get_a.trailer.seq = req_counter;
#ifndef __LEGACY_XDCOMMS__
    int tries_remaining = 30;
    while(tries_remaining != 0){
        my_xdc_asyn_send(psocket, &request_get_a, &t_tag , mycmap);
#ifndef __ONEWAY_RPC__
        *error = my_xdc_recv(ssocket, &response_get_a, &o_tag , mycmap);
        fprintf(stderr, "REQU get_a: ReqId=%d error=%d tries=%d ",
 ↪  request_get_a.trailer.seq, *error, tries_remaining);
```

```
        if (*error > 0) fprintf(stderr, "ResId=%d Reserr=%d ",
         ↪  response_get_a.trailer.seq >> 2, (response_get_a.trailer.seq
         ↪  >> 1) & 0x01);
        fprintf(stderr, "t_tag=<%02u, %02u, %02u>, ", t_tag.mux,
 ↪  t_tag.sec, t_tag.typ);
        fprintf(stderr, "o_tag=<%02u, %02u, %02u>\n", o_tag.mux,
 ↪  o_tag.sec, o_tag.typ);
        if (*error == -1){
            tries_remaining--;
            continue;
        }
#else
        *error = 0;
        fprintf(stderr, "REQU get_a: ReqId=%d error=%d tries=%d ",
 ↪  request_get_a.trailer.seq, *error, tries_remaining);
        if (*error > 0) fprintf(stderr, "ResId=%d Reserr=%d ",
         ↪  response_get_a.trailer.seq >> 2, (response_get_a.trailer.seq
         ↪  >> 1) & 0x01);
        fprintf(stderr, "t_tag=<%02u, %02u, %02u>, ", t_tag.mux,
 ↪  t_tag.sec, t_tag.typ);
        fprintf(stderr, "o_tag=<%02u, %02u, %02u>\n", o_tag.mux,
 ↪  o_tag.sec, o_tag.typ);
#endif /* __ONEWAY_RPC__ */
        break;  /* Reach here if received a response or __ONEWAY_RPC__
         ↪  */
    }
    zmq_close(psocket);
    zmq_close(ssocket);
    zmq_ctx_shutdown(ctx);
#else /* __LEGACY_XDCOMMS__ */
    int tries_remaining = 30;
    while(tries_remaining != 0){
        xdc_asyn_send(psocket, &request_get_a, &t_tag);
#ifndef __ONEWAY_RPC__
        *error = xdc_recv(ssocket, &response_get_a, &o_tag);
        fprintf(stderr, "REQU get_a: ReqId=%d error=%d tries=%d ",
 ↪  request_get_a.trailer.seq, *error, tries_remaining);
        if (*error > 0) fprintf(stderr, "ResId=%d Reserr=%d ",
         ↪  response_get_a.trailer.seq >> 2, (response_get_a.trailer.seq
         ↪  >> 1) & 0x01);
        fprintf(stderr, "t_tag=<%02u, %02u, %02u>, ", t_tag.mux,
 ↪  t_tag.sec, t_tag.typ);
        fprintf(stderr, "o_tag=<%02u, %02u, %02u>\n", o_tag.mux,
 ↪  o_tag.sec, o_tag.typ);
```

```
        if (*error == -1){
            tries_remaining--;
            continue;
        }
#else
        *error = 0;
        fprintf(stderr, "REQU get_a: ReqId=%d error=%d tries=%d ",
 ↪  request_get_a.trailer.seq, *error, tries_remaining);
        if (*error > 0) fprintf(stderr, "ResId=%d Reserr=%d ",
         ↪  response_get_a.trailer.seq >> 2, (response_get_a.trailer.seq
         ↪  >> 1) & 0x01);
        fprintf(stderr, "t_tag=<%02u, %02u, %02u>, ", t_tag.mux,
 ↪  t_tag.sec, t_tag.typ);
        fprintf(stderr, "o_tag=<%02u, %02u, %02u>\n", o_tag.mux,
 ↪  o_tag.sec, o_tag.typ);
#endif /* __ONEWAY_RPC__ */
        break;  /* Reach here if received a response or __ONEWAY_RPC__
         ↪  */
    }
#endif /* __LEGACY_XDCOMMS__ */
    req_counter++;
#ifndef __ONEWAY_RPC__
    result = response_get_a.ret;
    return (result);
#else
    return 0;
#endif /* __ONEWAY_RPC__ */
}

#pragma cle begin ERR_HANDLE_RPC_GET_A
double _err_handle_rpc_get_a(){
#pragma cle end ERR_HANDLE_RPC_GET_A
    int err_num;
    double res = _rpc_get_a(&err_num);
    // err handling code goes here
    return res;
}
```

On the orange side, the `main` function is replaced with one which waits for requests from purple repeatedly:

```
int main(int argc, char *argv[]) {
  return _slave_rpc_loop();
}
```

The slave rpc loop is defined as follows in a multithreaded environment. It will create a separate thread for handling requests for `get_a`

```
#define WRAP(X) void *_wrapper_##X(void *tag) { while(1) {
 ↪  _handle_##X(); } }
WRAP(request_get_a)

int _slave_rpc_loop() {
    _hal_init((char *)INURI, (char *)OUTURI);
    pthread_t tid[NXDRPC];
    pthread_create(&tid[0], NULL, _wrapper_request_get_a, NULL);
    for (int i = 0; i < NXDRPC; i++) pthread_join(tid[i], NULL);
    return 0;
}
```

These requests are handled by the `_handle_rpc_get_a` function, which calls `get_a` internally and sends the response back to purple over the wire. The definition of `_handle_rpc_get_a` is as follows:

```
#pragma cle begin HANDLE_REQUEST_GET_A
void _handle_request_get_a() {
#pragma cle end HANDLE_REQUEST_GET_A
    gaps_tag t_tag;
    gaps_tag o_tag;
#ifndef __LEGACY_XDCOMMS__
    my_tag_write(&t_tag, MUX_REQUEST_GET_A, SEC_REQUEST_GET_A,
 ↪  DATA_TYP_REQUEST_GET_A);
#else
    tag_write(&t_tag, MUX_REQUEST_GET_A, SEC_REQUEST_GET_A,
 ↪  DATA_TYP_REQUEST_GET_A);
#endif /* __LEGACY_XDCOMMS__ */
#ifndef __LEGACY_XDCOMMS__
    my_tag_write(&o_tag, MUX_RESPONSE_GET_A, SEC_RESPONSE_GET_A,
 ↪  DATA_TYP_RESPONSE_GET_A);
#else
    tag_write(&o_tag, MUX_RESPONSE_GET_A, SEC_RESPONSE_GET_A,
 ↪  DATA_TYP_RESPONSE_GET_A);
#endif /* __LEGACY_XDCOMMS__ */
    static int res_counter = 0;
    static double last_processed_result = 0;
    static int last_processed_error = 0;
    static int inited = 0;
#ifndef __LEGACY_XDCOMMS__
    void *psocket;
```

```
    void *ssocket;
#else
    static void *psocket;
    static void *ssocket;
#endif /* __LEGACY_XDCOMMS__ */
    #pragma cle begin TAG_REQUEST_GET_A
    request_get_a_datatype request_get_a;
    #pragma cle end TAG_REQUEST_GET_A
    #pragma cle begin TAG_RESPONSE_GET_A
    response_get_a_datatype response_get_a;
    #pragma cle end TAG_RESPONSE_GET_A
#ifndef __LEGACY_XDCOMMS__
    codec_map  mycmap[DATA_TYP_MAX];
    for (int i=0; i < DATA_TYP_MAX; i++)  mycmap[i].valid=0;
    my_xdc_register(request_get_a_data_encode,
 ↪  request_get_a_data_decode, DATA_TYP_REQUEST_GET_A, mycmap);
    my_xdc_register(response_get_a_data_encode,
 ↪  response_get_a_data_decode, DATA_TYP_RESPONSE_GET_A, mycmap);
#endif /* __LEGACY_XDCOMMS__ */
#ifndef __LEGACY_XDCOMMS__
    void * ctx = zmq_ctx_new();
    psocket = my_xdc_pub_socket(ctx, (char *)OUTURI);
    ssocket = my_xdc_sub_socket(t_tag, ctx, (char*)INURI);
    sleep(1); /* zmq socket join delay */
#else
    if (!inited) {
        inited = 1;
        psocket = xdc_pub_socket();
        ssocket = xdc_sub_socket(t_tag);
        sleep(1); /* zmq socket join delay */
    }
#endif /* __LEGACY_XDCOMMS__ */
#ifndef __LEGACY_XDCOMMS__
    int proc_error = 1;
    while (proc_error == 1) {
        my_xdc_blocking_recv(ssocket, &request_get_a, &t_tag, mycmap);
        int req_counter = request_get_a.trailer.seq;
        if(req_counter > res_counter){
            proc_error = 0;
            res_counter = req_counter;
            last_processed_result = get_a();
            response_get_a.ret = last_processed_result;
            last_processed_error = proc_error;
        }
```

```
#ifndef __ONEWAY_RPC__
        response_get_a.trailer.seq = res_counter << 2 |
 ↪  last_processed_error << 1;
        my_xdc_asyn_send(psocket, &response_get_a, &o_tag, mycmap);
#else /* __ONEWAY_RPC__ */
        res_counter = req_counter;
#endif /* __ONEWAY_RPC__ */
        fprintf(stderr, "RESP get_a: ReqId=%d ResId=%d err=%d (seq=0x%x)
 ↪  ", req_counter, res_counter, proc_error, last_processed_error);
        fprintf(stderr, "t_tag=<%02u, %02u, %02u>, ", t_tag.mux,
 ↪  t_tag.sec, t_tag.typ);
        fprintf(stderr, "o_tag=<%02u, %02u, %02u>\n", o_tag.mux,
 ↪  o_tag.sec, o_tag.typ);
    }
    zmq_close(psocket);
    zmq_close(ssocket);
    zmq_ctx_shutdown(ctx);
#else
    int proc_error = 1;
    while (proc_error == 1) {
        xdc_blocking_recv(ssocket, &request_get_a, &t_tag);
        int req_counter = request_get_a.trailer.seq;
        if(req_counter > res_counter){
            proc_error = 0;
            res_counter = req_counter;
            last_processed_result = get_a();
            response_get_a.ret = last_processed_result;
            last_processed_error = proc_error;
        }
#ifndef __ONEWAY_RPC__
        response_get_a.trailer.seq = res_counter << 2 |
 ↪  last_processed_error << 1;
        xdc_asyn_send(psocket, &response_get_a, &o_tag);
#else /* __ONEWAY_RPC__ */
        res_counter = req_counter;
#endif /* __ONEWAY_RPC__ */
        fprintf(stderr, "RESP get_a: ReqId=%d ResId=%d err=%d (seq=0x%x)
 ↪  ", req_counter, res_counter, proc_error, last_processed_error);
        fprintf(stderr, "t_tag=<%02u, %02u, %02u>, ", t_tag.mux,
 ↪  t_tag.sec, t_tag.typ);
        fprintf(stderr, "o_tag=<%02u, %02u, %02u>\n", o_tag.mux,
 ↪  o_tag.sec, o_tag.typ);
    }
#endif /* __LEGACY_XDCOMMS__ */
```

```
}
```

5.11 HAL Configuration Files

5.11.1 `devices.json`

`devices.json` provides the interfaces for reading/writing to the employed GAPS devices. Note that different devices can be used in the forward and reverse direction between a pair of enclaves. Device settings for BITW(MIND), BKND (ILIP), and Emulator shown.

Bump-in-the-Wire (MIND device)

```
{
  "devices": [
    {
      "model":    "sdh_ha_v1",
      "comms":    "zmq",
      "mode_in":  "sub",
      "mode_out": "pub"
    },
    {
      "model":          "sdh_bw_v1",
      "path":           "lo",
      "comms":          "udp",
      "enclave_name_1": "orange",
      "listen_addr_1":  "10.0.0.1",
      "listen_port_1":  6788,
      "enclav_name_2":  "purple",
      "listen_addr_2":  "10.0.1.1",
      "listen_port_2":  6788
    }
  ]
}
```

Bookends (ILIP Device)

```
{
  "devices": [
    {
      "model":    "sdh_ha_v1",
      "comms":    "zmq",
      "mode_in":  "sub",
```

```
      "mode_out": "pub"
    },
    {
      "model":          "sdh_be_v3",
      "path":           "/dev/gaps_ilip_0_root",
      "comms":          "ilp",
      "enclave_name_1": "green",
      "path_r_1":       "/dev/gaps_ilip_2_read",
      "path_w_1":       "/dev/gaps_ilip_2_write",
      "from_mux_1":     11,
      "init_at_1":      1,
      "enclave_name_2": "orange",
      "path_r_2":       "/dev/gaps_ilip_2_read",
      "path_w_2":       "/dev/gaps_ilip_2_write",
      "from_mux_2":     12,
      "init_at_2":      1
    }
  ]
}
```

Emulator (socat device)

```
{
  "devices": [
    {
      "model":    "sdh_ha_v1",
      "comms":    "zmq",
      "mode_in":  "sub",
      "mode_out": "pub"
    },
    {
      "model":    "sdh_socat_v1",
      "path":     "/dev/vcom",
      "comms":    "tty"
    }
  ]
}
```

5.11.2 `xdconf.ini`

`xdconf.ini` is automatically generated during the automagic stages of the project build by the rpc generator. The file appears in `partitioned/{single, multi}threaded`

directory. The file includes the mux, sec, typ mappings for each data type. The data types are oraganized by enclave with from/to enclave clearly specified so the direction is apparent. Appropriate in/out uri interfaces for where the message is read or written to and from HAL are also specified. `xdconf.ini` content ultimately populates the map portion of the HAL config file.

```
{
  "enclaves": [
    {
      "enclave": "orange",
      "inuri": "ipc:///tmp/sock_suborange",
      "outuri": "ipc:///tmp/sock_puborange",
      "halmaps": [
        {
          "from": "purple",
          "to": "orange",
          "mux": 2,
          "sec": 2,
          "typ": 1,
          "name": "nextrpc_purple_orange"
        },
        {
          "from": "orange",
          "to": "purple",
          "mux": 1,
          "sec": 1,
          "typ": 2,
          "name": "okay_purple_orange"
        },
        {
          "from": "purple",
          "to": "orange",
          "mux": 2,
          "sec": 2,
          "typ": 3,
          "name": "request_get_a"
        },
        {
          "from": "orange",
          "to": "purple",
          "mux": 1,
          "sec": 1,
          "typ": 4,
```

```
        "name": "response_get_a"
      }
    ]
  },
  {
    "enclave": "purple",
    "inuri": "ipc:///tmp/sock_subpurple",
    "outuri": "ipc:///tmp/sock_pubpurple",
    "halmaps": [
      {
        "from": "purple",
        "to": "orange",
        "mux": 2,
        "sec": 2,
        "typ": 1,
        "name": "nextrpc_purple_orange"
      },
      {
        "from": "orange",
        "to": "purple",
        "mux": 1,
        "sec": 1,
        "typ": 2,
        "name": "okay_purple_orange"
      },
      {
        "from": "purple",
        "to": "orange",
        "mux": 2,
        "sec": 2,
        "typ": 3,
        "name": "request_get_a"
      },
      {
        "from": "orange",
        "to": "purple",
        "mux": 1,
        "sec": 1,
        "typ": 4,
        "name": "response_get_a"
      }
    ]
  }
]
```

```
}
```

5.11.3 hal_orange configuration

The HAL configuration tool combines the `xdconf.ini` with selected devices (see closure_env.sh). Example orange enclave configuration for example1 shown below.

```
maps =
(
    {
        from_mux = 2;
        to_mux = 2;
        from_sec = 2;
        to_sec = 2;
        from_typ = 1;
        to_typ = 1;
        codec = "NULL";
        to_dev = "xdd0";
        from_dev = "xdd1";
    },
    {
        from_mux = 1;
        to_mux = 1;
        from_sec = 1;
        to_sec = 1;
        from_typ = 2;
        to_typ = 2;
        codec = "NULL";
        to_dev = "xdd1";
        from_dev = "xdd0";
    },
    {
        from_mux = 2;
        to_mux = 2;
        from_sec = 2;
        to_sec = 2;
        from_typ = 3;
        to_typ = 3;
        codec = "NULL";
        to_dev = "xdd0";
        from_dev = "xdd1";
    },
    {
```

```
        from_mux = 1;
        to_mux = 1;
        from_sec = 1;
        to_sec = 1;
        from_typ = 4;
        to_typ = 4;
        codec = "NULL";
        to_dev = "xdd1";
        from_dev = "xdd0";
    }
);
devices =
(
    {
        enabled = 1;
        id = "xdd0";
        model = "sdh_ha_v1";
        comms = "zmq";
        mode_in = "sub";
        mode_out = "pub";
        addr_in = "ipc:///tmp/sock_puborange";
        addr_out = "ipc:///tmp/sock_suborange";
    },
    {
        enabled = 1;
        id = "xdd1";
        path = "/dev/vcom";
        model = "sdh_socat_v1";
        comms = "tty";
    }
);
```

5.12 EMU configuration

5.12.1 `enclaves.json`

`enclaves.json` is a *HAL-Daemon* configuration file that specifies the cross-domain elments of the scenario nodes and associated topology. Example1 enclaves.json shown below. Key elements include:

- qname: scenario name
- enclave: the enclave, set of nodes running at the same level
- xdhost: cross-domain host, node from enclave with access to the guard

- inthost: internal enclave node (future use)
- halconf: HAL configuration file that should be used to run HAL automatically on xdhost node
- xdgateway: specifies node between the enclaves, used to manage cross-domain communication and filters in BITW config
- xdlink: configures the cross domain link between xdhost and xdgateway, BITW or BKND specified for guard type

```
{
  "qname": "example1",
  "enclave":
  [
    {
      "qname": "orange",
      "xdhost":
      [
    {
          "hostname": "orange-enclave-gw-P",
      "halconf": "example1_hal_orange.cfg",
      "hwconf":{"arch": "amd64"},
      "swconf":{"os": "ubuntu", "distro": "focal", "kernel": "focal",
            "service": [{"s": "UserDefined"}]
      },
      "nwconf":{"interface":
            [{"ifname" :"eth0", "addr":"10.0.101.1/24"},
             {"ifname" :"eth1", "addr":"10.1.2.2/24"}] },
      "ifpeer":[{"ifname": "eth0", "peername": "orange-hub"},
                {"ifname": "eth1", "peername": "orange-purple-xd-gw"}]
    }
      ],
      "inthost":
      [
    {
      "hostname": "orange-1",
      "swconf":{"service": [{"s": "UserDefined"}]},
      "nwconf":{"interface":
            [{"ifname": "eth0", "addr": "10.0.101.2/24"}] },
          "ifpeer":[{"ifname": "eth0", "peername": "orange-hub"}]
    },
    {
      "hostname": "orange-2",
      "swconf":{"service": [{"s": "UserDefined"}]},
      "nwconf":{"interface":
```

```
        [{"ifname": "eth0", "addr": "10.0.101.3/24"}] },
  "ifpeer":[{"ifname": "eth0", "peername": "orange-hub"}]
}
  ],
  "link":
  [
{"f": "orange-hub", "t":"orange-1", "bandwidth": "100000000", "delay": 0},
    {"f": "orange-hub", "t":"orange-2", "bandwidth": "100000000", "delay": 0},
    {"f": "orange-hub", "t":"orange-enclave-gw-P", "bandwidth": "100000000", "delay"
  ],
  "hub":
  [
{ "hostname": "orange-hub",
  "ifpeer": [{"ifname": "e0", "peername": "orange-enclave-gw-P"},
         {"ifname": "e1", "peername": "orange-1"},
         {"ifname": "e2", "peername": "orange-2"}]
}
  ]
},
{
  "qname": "purple",
  "xdhost":
  [
{
  "hostname": "purple-enclave-gw-0",
  "halconf": "example1_hal_purple.cfg",
  "hwconf":{"arch": "amd64"},
  "swconf":{"os": "ubuntu", "distro": "focal", "kernel": "focal",
        "service": [{"s": "UserDefined"}]},
  "nwconf":{"interface":
        [{"ifname" :"eth0", "addr":"10.0.102.1/24"},
         {"ifname" :"eth1", "addr":"10.2.1.2/24"}] },
  "ifpeer":[{"ifname": "eth0", "peername": "purple-hub"},
            {"ifname": "eth1", "peername": "orange-purple-xd-gw"}]
}
  ],
  "inthost":
  [
{
  "hostname": "purple-1",
  "swconf":{"service": [{"s": "UserDefined"}]},
  "nwconf":{"interface":
        [{"ifname": "eth0", "addr": "10.0.102.2/24"}] },
  "ifpeer":[{"ifname": "eth0", "peername": "purple-hub"}]
```

```
  },
  {
    "hostname": "purple-2",
    "swconf":{"service": [{"s": "UserDefined"}]},
    "nwconf":{"interface":
          [{"ifname": "eth0", "addr": "10.0.102.3/24"}] },
    "ifpeer":[{"ifname": "eth0", "peername": "purple-hub"}]
  }
    ],
    "link":
    [
  {"f": "purple-hub", "t":"purple-1", "bandwidth": "100000000", "delay": 0},
      {"f": "purple-hub", "t":"purple-2", "bandwidth": "100000000", "delay": 0},
      {"f": "purple-hub", "t":"purple-enclave-gw-O", "bandwidth": "100000000", "delay"
    ],
    "hub":
    [
  { "hostname": "purple-hub",
    "ifpeer": [{"ifname": "e0", "peername": "purple-enclave-gw-O"},
           {"ifname": "e1", "peername": "purple-1"},
           {"ifname": "e2", "peername": "purple-2"}]
  }
    ]
  }
],
"xdgateway":
[
  {
    "hostname": "orange-purple-xd-gw",
    "swconf":{"service": [{"s": "UserDefined"}, {"s": "IPForward"}]},
    "nwconf":{"interface":
      [{"ifname": "eth0", "addr": "10.1.2.1/24"},
       {"ifname": "eth1", "addr": "10.2.1.1/24"}] },
    "ifpeer":[{"ifname": "eth0", "peername": "orange-enclave-gw-P"},
      {"ifname": "eth1", "peername": "purple-enclave-gw-O"}]
  }
],
"xdlink":
[
  { "model":  "BKND",
    "left":   {"f": "orange-enclave-gw-P", "t":"orange-purple-xd-gw",
           "egress":   {"filterspec": "left-egress-spec", "bandwidth":"100000000", "del
               "ingress":  {"filterspec": "left-ingress-spec", "bandwidth":"100000000"
    "right":  {"f": "orange-purple-xd-gw", "t":"purple-enclave-gw-O",
```

```
            "egress":   {"filterspec": "right-egress-spec", "bandwidth":"100000000", "de
                "ingress":   {"filterspec": "right-ingress-spec", "bandwidth":"10000000
    }
  ]
}
```

5.12.2 `layout.json`

Controls the graphical layout of the scenario elements (nodes, links, colors). Use [2,3,4]enclave directories for boiler plate layouts depending on the network size required for additional scenarios.

```
{
 "canvas": { "name": "Canvas1" },
 "option": {
  "optglobal": {
   "interface_names": "no",
   "ip_addresses": "no",
   "ipv6_addresses": "no",
   "node_labels": "yes",
   "link_labels": "no",
   "show_api": "no",
   "background_images": "no",
   "annotations": "yes",
   "grid": "yes",
   "traffic_start": "0"
  },
  "session": {}
 },
 "nodelayout": [
  {
   "hostname":"orange-enclave-gw-P",
   "canvas":"Canvas1",
   "iconcoords": {"x":265.0, "y":171.0},
   "labelcoords": {"x":265.0, "y":203.0}
  },
  {
   "hostname":"purple-enclave-gw-O",
   "canvas":"Canvas1",
   "iconcoords": {"x":697.0, "y":168.0},
   "labelcoords": {"x":697.0, "y":200.0}
  },
  {
```

```
  "hostname":"orange-1",
  "canvas":"Canvas1",
  "iconcoords": {"x":122.0, "y":74.0},
  "labelcoords": {"x":122.0, "y":106.0}
 },
 {
  "hostname":"orange-2",
  "canvas":"Canvas1",
  "iconcoords": {"x":121.0, "y":265.0},
  "labelcoords": {"x":121.0, "y":297.0}
 },
 {
  "hostname":"purple-1",
  "canvas":"Canvas1",
  "iconcoords": {"x":837.0, "y":72.0},
  "labelcoords": {"x":837.0, "y":104.0}
 },
 {
  "hostname":"purple-2",
  "canvas":"Canvas1",
  "iconcoords": {"x":839.0, "y":268.0},
  "labelcoords": {"x":839.0, "y":300.0}
 },
 {
  "hostname":"orange-hub",
  "canvas":"Canvas1",
  "iconcoords": {"x":121.0, "y":171.0},
  "labelcoords": {"x":121.0, "y":195.0}
 },
 {
  "hostname":"purple-hub",
  "canvas":"Canvas1",
  "iconcoords": {"x":838.0, "y":167.0},
  "labelcoords": {"x":838.0, "y":191.0}
 },
 {
  "hostname":"orange-purple-xd-gw",
  "canvas":"Canvas1",
  "iconcoords": {"x":483.0, "y":169.0},
  "labelcoords": {"x":483.0, "y":201.0}
 }
],
"annotation": [
 {
```

```
    "bbox": {"x1":56.0, "y1": 36.0, "x2": 399.0, "y2": 322.0},
    "type": "rectangle",
    "label": "OrangeEnclave",
    "labelcolor": "black",
    "fontfamily": "Arial",
    "fontsize": 12,
    "color": "#ff8c00",
    "width": 0,
    "border": "black",
    "rad": 25,
    "canvas": "Canvas1"
  },
  {
    "bbox": {"x1":607.0, "y1": 41.0, "x2": 918.0, "y2": 327.0},
    "type": "rectangle",
    "label": "PurpleEnclave",
    "labelcolor": "black",
    "fontfamily": "Arial",
    "fontsize": 12,
    "color": "#c300ff",
    "width": 0,
    "border": "black",
    "rad": 25,
    "canvas": "Canvas1"
  }
 ]
}
```

5.12.3 `settings.json`

`settings.json` controls general settings for the emulator. Fields of this json file include:

- core_timeout: if CORE does not start in specified number of seconds, emulator will quit. Make sure core-daemon is properly installed
- instdir: parent directory where emu is installed
- imgdir: directory where QEMU golden images are stored
- mgmt_ip: IP of VMs for ssh purposes
- shadow_directories: comma separated list of directories to be mounted uniquely in CORE BSD containers
- snapdir: location of VM snapshots relative to `${instdir}/emu`
- imndir: location of scenario CORE .imn files relative to `${instdir}/emu`

```
{
  "core_timeout": 30,
  "instdir": "/opt/closure",
  "imgdir": "/IMAGES",
  "mgmt_ip": "10.200.0.1",
  "shadow_directories": "/root;",
  "snapdir": ".snapshots",
  "imndir": ".imnfiles"
}
```

5.13 Autogenerated Data Format Definition Language (DFDL) Schemas

CLOSURE generates DFDL schema specifications for the serialized cross-domain data items (RPC requests and responses) along with the GAPS header and trailer fields for the bump-in-the-wire (MIND) and bookend (ILIP) GAPS devices.

The following is the DFDL schema generated from the `example1 get_a` request and response for both the `bw` and the `be` variants, in that order.

The `bw` variant addresses IP based bump-in-the-wire cross domain devices of which, the GAPS MIND is an example. The `be` variant addresses the bookend style (e.g., PCIe based devices), specifically using the GAPS ILIP formats.

```
<?xml version="1.0" ?><xs:schema
 ↪  xmlns:xs="http://www.w3.org/2001/XMLSchema"
 ↪  xmlns:dfdl="http://www.ogf.org/dfdl/dfdl-1.0/"
 ↪  xmlns:fn="http://www.w3.org/2005/xpath-functions"
 ↪  xmlns:daf="urn:ogf:dfdl:2013:imp:daffodil.apache.org:2018:ext"
 ↪  xmlns:gma="urn:gma:1.0" targetNamespace="urn:gma:1.0">

 <xs:annotation>
  <xs:appinfo source="http://www.ogf.org/dfdl/">
   <dfdl:defineVariable name="ByteOrder" type="xs:string"/>
   <dfdl:defineFormat name="defaults">
```

```
   <dfdl:format alignment="1" alignmentUnits="bits"
↪  binaryBooleanFalseRep="0" binaryBooleanTrueRep="1"
↪  binaryCalendarEpoch="1970-01-01T00:00:00+00:00"
↪  binaryCalendarRep="bcd" binaryDecimalVirtualPoint="0"
↪  binaryFloatRep="ieee" binaryNumberCheckPolicy="lax"
↪  binaryNumberRep="binary" binaryPackedSignCodes="C D F C"
↪  calendarCenturyStart="53" calendarCheckPolicy="lax"
↪  calendarDaysInFirstWeek="4" calendarFirstDayOfWeek="Monday"
↪  calendarLanguage="en-US" calendarObserveDST="yes"
↪  calendarPatternKind="implicit"
↪  calendarPattern="yyyy-MM-dd'T'HH:mm:ss" calendarTimeZone="UTC"
↪  choiceLengthKind="implicit" decimalSigned="yes"
↪  documentFinalTerminatorCanBeMissing="no"
↪  emptyValueDelimiterPolicy="none" encoding="utf-8"
↪  encodingErrorPolicy="replace" escapeSchemeRef="" fillByte="%NUL;"
↪  floating="no" ignoreCase="no" initiatedContent="no" initiator=""
↪  leadingSkip="0" lengthKind="implicit" lengthUnits="bits"
↪  nilKind="literalValue" nilValueDelimiterPolicy="none" nilValue="NIL"
↪  occursCountKind="implicit" outputNewLine="%CR;%LF;"
↪  prefixIncludesPrefixLength="no" representation="binary" separator=""
↪  separatorPosition="infix" sequenceKind="ordered" terminator=""
↪  textBidi="no" textBooleanFalseRep="false"
↪  textBooleanJustification="left" textBooleanPadCharacter="%SP;"
↪  textBooleanTrueRep="true" textCalendarJustification="left"
↪  textCalendarPadCharacter="%SP;" textNumberCheckPolicy="lax"
↪  textNumberJustification="right" textNumberPadCharacter="0"
↪  textNumberPattern="#0" textNumberRep="standard"
↪  textNumberRoundingIncrement="0.0" textNumberRoundingMode="roundUp"
↪  textNumberRounding="pattern" textOutputMinLength="0"
↪  textPadKind="none" textStandardBase="10"
↪  textStandardDecimalSeparator="." textStandardGroupingSeparator=","
↪  textStandardInfinityRep="Inf" textStandardNaNRep="NaN"
↪  textStandardZeroRep="" textStringJustification="left"
↪  textStringPadCharacter="%SP;" textTrimKind="none"
↪  textZonedSignStyle="asciiStandard" trailingSkip="0"
↪  truncateSpecifiedLengthString="no" useNilForDefault="no"
↪  utf16Width="fixed" bitOrder="mostSignificantBitFirst"/>
  </dfdl:defineFormat>
  <dfdl:format ref="gma:defaults" byteOrder="bigEndian"/>
 </xs:appinfo>
</xs:annotation>

<xs:element name="GapsPDU">
 <xs:complexType>
```

```
  <xs:choice>         <!-- no way to discriminate SDHBW or SDHBE, so
↪  uncomment one -->
    <!-- <xs:element ref="gma:SDHBEPDU" /> -->
    <xs:element ref="gma:SDHBWPDU"/>
  </xs:choice>
 </xs:complexType>
</xs:element>

<xs:element name="SDHBWPDU">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="gma:SDHBWHeader"/>
   <xs:element ref="gma:ApplicationData"/>
   <xs:element ref="gma:GapsTrailer"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>

<xs:element name="SDHBWHeader">
 <xs:complexType>
  <xs:sequence dfdl:byteOrder="bigEndian">
   <xs:element name="tag0" type="gma:gapsuint8"/>
   <xs:element name="tagm" type="gma:gapsuint8"/>
   <xs:element name="tags" type="gma:gapsuint8"/>
   <xs:element name="tagt" type="gma:gapsuint8"/>
   <xs:element name="len" type="gma:gapsuint16"/>
   <xs:element name="crc" type="gma:gapsuint16"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>


<xs:simpleType name="gapsdouble" dfdl:lengthKind="explicit"
↪  dfdl:length="8" dfdl:lengthUnits="bytes"
↪  dfdl:byteOrder="littleEndian">
 <xs:restriction base="xs:double"/>
</xs:simpleType>

<xs:simpleType name="gapsfloat" dfdl:lengthKind="explicit"
↪  dfdl:length="4" dfdl:lengthUnits="bytes"
↪  dfdl:byteOrder="littleEndian">
 <xs:restriction base="xs:float"/>
</xs:simpleType>
```

```
<xs:simpleType name="gapsuint64" dfdl:lengthKind="explicit"
↪  dfdl:length="64" dfdl:lengthUnits="bits">
 <xs:restriction base="xs:unsignedLong"/>
</xs:simpleType>

<xs:simpleType name="gapsuint32" dfdl:lengthKind="explicit"
↪  dfdl:length="32" dfdl:lengthUnits="bits">
 <xs:restriction base="xs:unsignedInt"/>
</xs:simpleType>

<xs:simpleType name="gapsuint16" dfdl:lengthKind="explicit"
↪  dfdl:length="16" dfdl:lengthUnits="bits">
 <xs:restriction base="xs:unsignedInt"/>
</xs:simpleType>

<xs:simpleType name="gapsuint8" dfdl:lengthKind="explicit"
↪  dfdl:length="8" dfdl:lengthUnits="bits">
 <xs:restriction base="xs:unsignedInt"/>
</xs:simpleType>

<xs:simpleType name="gapsint64" dfdl:lengthKind="explicit"
↪  dfdl:length="64" dfdl:lengthUnits="bits">
 <xs:restriction base="xs:long"/>
</xs:simpleType>

<xs:simpleType name="gapsint32" dfdl:lengthKind="explicit"
↪  dfdl:length="32" dfdl:lengthUnits="bits">
 <xs:restriction base="xs:int"/>
</xs:simpleType>

<xs:simpleType name="gapsint16" dfdl:lengthKind="explicit"
↪  dfdl:length="16" dfdl:lengthUnits="bits">
 <xs:restriction base="xs:short"/>
</xs:simpleType>

<xs:simpleType name="gapsint8" dfdl:lengthKind="explicit"
↪  dfdl:length="8" dfdl:lengthUnits="bits">
 <xs:restriction base="xs:byte"/>
</xs:simpleType>

<xs:element name="GapsTrailer">
 <xs:complexType>
  <xs:sequence dfdl:byteOrder="bigEndian">
   <xs:element name="seq" type="gma:gapsuint32"/>
```

```
    <xs:element name="rqr" type="gma:gapsuint32"/>
    <xs:element name="oid" type="gma:gapsuint32"/>
    <xs:element name="mid" type="gma:gapsuint16"/>
    <xs:element name="crc" type="gma:gapsuint16"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="ApplicationData">
  <xs:complexType>
   <xs:choice>
    <xs:element ref="gma:NextRPC">
     <xs:annotation>
      <xs:appinfo source="http://www.ogf.org/dfdl/">
       <dfdl:discriminator test="{../../gma:SDHBWHeader/tagt eq 1}"/>
↪
      </xs:appinfo>
     </xs:annotation>
    </xs:element>
    <xs:element ref="gma:Okay">
     <xs:annotation>
      <xs:appinfo source="http://www.ogf.org/dfdl/">
       <dfdl:discriminator test="{../../gma:SDHBWHeader/tagt eq 2}"/>
↪
      </xs:appinfo>
     </xs:annotation>
    </xs:element>
    <xs:element ref="gma:Request_get_a">
     <xs:annotation>
      <xs:appinfo source="http://www.ogf.org/dfdl/">
       <dfdl:discriminator test="{../../gma:SDHBWHeader/tagt eq 3}"/>
↪
      </xs:appinfo>
     </xs:annotation>
    </xs:element>
    <xs:element ref="gma:Response_get_a">
     <xs:annotation>
      <xs:appinfo source="http://www.ogf.org/dfdl/">
       <dfdl:discriminator test="{../../gma:SDHBWHeader/tagt eq 4}"/>
↪
      </xs:appinfo>
     </xs:annotation>
    </xs:element>
   </xs:choice>
  </xs:complexType>
```

```
 </xs:element>

 <xs:element name="NextRPC">
  <xs:complexType>
   <xs:sequence dfdl:byteOrder="bigEndian">
    <xs:element name="mux" type="gma:gapsint32"/>
    <xs:element name="sec" type="gma:gapsint32"/>
    <xs:element name="typ" type="gma:gapsint32"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>

 <xs:element name="Okay">
  <xs:complexType>
   <xs:sequence dfdl:byteOrder="bigEndian">
    <xs:element name="x" type="gma:gapsint32"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>

 <xs:element name="Request_get_a">
  <xs:complexType>
   <xs:sequence dfdl:byteOrder="bigEndian">
    <xs:element name="dummy" type="gma:gapsint32"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>

 <xs:element name="Response_get_a">
  <xs:complexType>
   <xs:sequence dfdl:byteOrder="bigEndian">
    <xs:element name="ret" type="gma:gapsdouble"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>

</xs:schema>

<?xml version="1.0" ?><xs:schema
 ↪  xmlns:xs="http://www.w3.org/2001/XMLSchema"
 ↪  xmlns:dfdl="http://www.ogf.org/dfdl/dfdl-1.0/"
 ↪  xmlns:fn="http://www.w3.org/2005/xpath-functions"
 ↪  xmlns:daf="urn:ogf:dfdl:2013:imp:daffodil.apache.org:2018:ext"
 ↪  xmlns:gma="urn:gma:1.0" targetNamespace="urn:gma:1.0">
```

```
<xs:annotation>
 <xs:appinfo source="http://www.ogf.org/dfdl/">
  <dfdl:defineVariable name="ByteOrder" type="xs:string"/>
  <dfdl:defineFormat name="defaults">
   <dfdl:format alignment="1" alignmentUnits="bits"
↪  binaryBooleanFalseRep="0" binaryBooleanTrueRep="1"
↪  binaryCalendarEpoch="1970-01-01T00:00:00+00:00"
↪  binaryCalendarRep="bcd" binaryDecimalVirtualPoint="0"
↪  binaryFloatRep="ieee" binaryNumberCheckPolicy="lax"
↪  binaryNumberRep="binary" binaryPackedSignCodes="C D F C"
↪  calendarCenturyStart="53" calendarCheckPolicy="lax"
↪  calendarDaysInFirstWeek="4" calendarFirstDayOfWeek="Monday"
↪  calendarLanguage="en-US" calendarObserveDST="yes"
↪  calendarPatternKind="implicit"
↪  calendarPattern="yyyy-MM-dd'T'HH:mm:ss" calendarTimeZone="UTC"
↪  choiceLengthKind="implicit" decimalSigned="yes"
↪  documentFinalTerminatorCanBeMissing="no"
↪  emptyValueDelimiterPolicy="none" encoding="utf-8"
↪  encodingErrorPolicy="replace" escapeSchemeRef="" fillByte="%NUL;"
↪  floating="no" ignoreCase="no" initiatedContent="no" initiator=""
↪  leadingSkip="0" lengthKind="implicit" lengthUnits="bits"
↪  nilKind="literalValue" nilValueDelimiterPolicy="none" nilValue="NIL"
↪  occursCountKind="implicit" outputNewLine="%CR;%LF;"
↪  prefixIncludesPrefixLength="no" representation="binary" separator=""
↪  separatorPosition="infix" sequenceKind="ordered" terminator=""
↪  textBidi="no" textBooleanFalseRep="false"
↪  textBooleanJustification="left" textBooleanPadCharacter="%SP;"
↪  textBooleanTrueRep="true" textCalendarJustification="left"
↪  textCalendarPadCharacter="%SP;" textNumberCheckPolicy="lax"
↪  textNumberJustification="right" textNumberPadCharacter="0"
↪  textNumberPattern="#0" textNumberRep="standard"
↪  textNumberRoundingIncrement="0.0" textNumberRoundingMode="roundUp"
↪  textNumberRounding="pattern" textOutputMinLength="0"
↪  textPadKind="none" textStandardBase="10"
↪  textStandardDecimalSeparator="." textStandardGroupingSeparator=","
↪  textStandardInfinityRep="Inf" textStandardNaNRep="NaN"
↪  textStandardZeroRep="" textStringJustification="left"
↪  textStringPadCharacter="%SP;" textTrimKind="none"
↪  textZonedSignStyle="asciiStandard" trailingSkip="0"
↪  truncateSpecifiedLengthString="no" useNilForDefault="no"
↪  utf16Width="fixed" bitOrder="mostSignificantBitFirst"/>
  </dfdl:defineFormat>
  <dfdl:format ref="gma:defaults" byteOrder="bigEndian"/>
```

```
 </xs:appinfo>
</xs:annotation>

<xs:element name="GapsPDU">
 <xs:complexType>
  <xs:choice>         <!-- no way to discriminate SDHBW or SDHBE, so
↪  uncomment one -->
   <xs:element ref="gma:SDHBEPDU"/>
   <!-- <xs:element ref="gma:SDHBWPDU" /> -->
  </xs:choice>
 </xs:complexType>
</xs:element>

<xs:element name="SDHBEPDU">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="gma:SDHBEHeader"/>
   <xs:element ref="gma:ApplicationData"/>
   <xs:element ref="gma:GapsTrailer"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>

<xs:element name="SDHBEHeader">
 <xs:complexType>
  <xs:sequence dfdl:byteOrder="bigEndian">
   <xs:element name="stag" type="gma:gapsuint32"/>
   <xs:element name="mtag" type="gma:gapsuint32"/>
   <xs:element name="cnt" type="gma:gapsuint32"/>
   <xs:element name="dtag" type="gma:gapsuint32"/>
   <xs:element name="its" type="gma:gapsuint64"/>
   <xs:element name="hts" type="gma:gapsuint64"/>
   <xs:element name="len" type="gma:gapsuint32"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>


<xs:simpleType name="gapsdouble" dfdl:lengthKind="explicit"
↪  dfdl:length="8" dfdl:lengthUnits="bytes"
↪  dfdl:byteOrder="littleEndian">
 <xs:restriction base="xs:double"/>
</xs:simpleType>
```

```
<xs:simpleType name="gapsfloat" dfdl:lengthKind="explicit"
↪  dfdl:length="4" dfdl:lengthUnits="bytes"
↪  dfdl:byteOrder="littleEndian">
 <xs:restriction base="xs:float"/>
</xs:simpleType>

<xs:simpleType name="gapsuint64" dfdl:lengthKind="explicit"
↪  dfdl:length="64" dfdl:lengthUnits="bits">
 <xs:restriction base="xs:unsignedLong"/>
</xs:simpleType>

<xs:simpleType name="gapsuint32" dfdl:lengthKind="explicit"
↪  dfdl:length="32" dfdl:lengthUnits="bits">
 <xs:restriction base="xs:unsignedInt"/>
</xs:simpleType>

<xs:simpleType name="gapsuint16" dfdl:lengthKind="explicit"
↪  dfdl:length="16" dfdl:lengthUnits="bits">
 <xs:restriction base="xs:unsignedInt"/>
</xs:simpleType>

<xs:simpleType name="gapsuint8" dfdl:lengthKind="explicit"
↪  dfdl:length="8" dfdl:lengthUnits="bits">
 <xs:restriction base="xs:unsignedInt"/>
</xs:simpleType>

<xs:simpleType name="gapsint64" dfdl:lengthKind="explicit"
↪  dfdl:length="64" dfdl:lengthUnits="bits">
 <xs:restriction base="xs:long"/>
</xs:simpleType>

<xs:simpleType name="gapsint32" dfdl:lengthKind="explicit"
↪  dfdl:length="32" dfdl:lengthUnits="bits">
 <xs:restriction base="xs:int"/>
</xs:simpleType>

<xs:simpleType name="gapsint16" dfdl:lengthKind="explicit"
↪  dfdl:length="16" dfdl:lengthUnits="bits">
 <xs:restriction base="xs:short"/>
</xs:simpleType>

<xs:simpleType name="gapsint8" dfdl:lengthKind="explicit"
↪  dfdl:length="8" dfdl:lengthUnits="bits">
 <xs:restriction base="xs:byte"/>
```

```
</xs:simpleType>

<xs:element name="GapsTrailer">
 <xs:complexType>
  <xs:sequence dfdl:byteOrder="bigEndian">
   <xs:element name="seq" type="gma:gapsuint32"/>
   <xs:element name="rqr" type="gma:gapsuint32"/>
   <xs:element name="oid" type="gma:gapsuint32"/>
   <xs:element name="mid" type="gma:gapsuint16"/>
   <xs:element name="crc" type="gma:gapsuint16"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="ApplicationData">
 <xs:complexType>
  <xs:choice>
   <xs:element ref="gma:NextRPC">
    <xs:annotation>
     <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:discriminator test="{../../gma:SDHBEHeader/dtag eq 1}"/>
↪
     </xs:appinfo>
    </xs:annotation>
   </xs:element>
   <xs:element ref="gma:Okay">
    <xs:annotation>
     <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:discriminator test="{../../gma:SDHBEHeader/dtag eq 2}"/>
↪
     </xs:appinfo>
    </xs:annotation>
   </xs:element>
   <xs:element ref="gma:Request_get_a">
    <xs:annotation>
     <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:discriminator test="{../../gma:SDHBEHeader/dtag eq 3}"/>
↪
     </xs:appinfo>
    </xs:annotation>
   </xs:element>
   <xs:element ref="gma:Response_get_a">
    <xs:annotation>
     <xs:appinfo source="http://www.ogf.org/dfdl/">
```

```
      <dfdl:discriminator test="{../../gma:SDHBEHeader/dtag eq 4}"/>
↪
     </xs:appinfo>
    </xs:annotation>
   </xs:element>
  </xs:choice>
 </xs:complexType>
</xs:element>

<xs:element name="NextRPC">
 <xs:complexType>
  <xs:sequence dfdl:byteOrder="bigEndian">
   <xs:element name="mux" type="gma:gapsint32"/>
   <xs:element name="sec" type="gma:gapsint32"/>
   <xs:element name="typ" type="gma:gapsint32"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>

<xs:element name="Okay">
 <xs:complexType>
  <xs:sequence dfdl:byteOrder="bigEndian">
   <xs:element name="x" type="gma:gapsint32"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>

<xs:element name="Request_get_a">
 <xs:complexType>
  <xs:sequence dfdl:byteOrder="bigEndian">
   <xs:element name="dummy" type="gma:gapsint32"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>

<xs:element name="Response_get_a">
 <xs:complexType>
  <xs:sequence dfdl:byteOrder="bigEndian">
   <xs:element name="ret" type="gma:gapsdouble"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>

</xs:schema>
```

References

[1] "VSCode," *Visual Studio Code - Code editing. Redefined.* Microsoft, 2022. Available: https://code.visualstudio.com/

[2] M. J. Beckerle and S. M. Hanson, "Data format description language (DFDL) v1.0 specification," *Data Format Description Language (DFDL) v1.0 Specification.* The Apache Software Foundation, 2022. Available: https://daffodil.apache.org/docs/dfdl/

[3] "GAPS CLOSURE project." GitHub, 2022. Available: https://github.com/gaps-closure

[4] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[5] P. J. Stuckey, K. Marriott, and G. Tack, "The minizinc handbook," *The MiniZinc Handbook - The MiniZinc Handbook 2.5.5.* 2022. Available: https://www.minizinc.org/doc-2.5.5/en/index.html

[6] S. Liu *et al.*, "Program-mandering: Quantitative privilege separation," in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, 2019, pp. 1023–1040. doi: 10.1145/3319535.3354218.

[7] Y. Huang *et al.*, "KSplit: Automating device driver isolation," in *16th USENIX symposium on operating systems design and implementation (OSDI 22)*, Jul. 2022, pp. 613–631. Available: https://www.usenix.org/conference/osdi22/presentation/huang-yongzhe

[8] S. Liu, G. Tan, and T. Jaeger, "PtrSplit: Supporting general pointers in automatic program partitioning," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 2359–2371. doi: 10.1145/3133956.3134066.

[9] M. Levatich, R. Brotzmann, B. Flin, T. Chen, R. Krishnan, and S. A. Edwards, "C program partitioning with fine-grained security constraints and post-partition verification," *Under Submission.* 2022.

[10] S. Marlow *et al.*, "Haskell 2010 language report," *Available online http://www.haskell.org/(May 2011)*, 2010.

[11] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Proceedings of the theory and practice of software, 14th international conference on tools and algorithms for the construction and analysis of systems*, 2008, pp. 337–340.

[12] "Common open research emulator (CORE)." U.S. Naval Research Laboratory, 2022. Available: https://www.nrl.navy.mil/Our-Work/Areas-of-Research/Information-Technology/NCS/CORE/

[13] F. Bellard, "QEMU, a fast and portable dynamic translator," in *Proceedings of the annual conference on USENIX annual technical conference*, 2005, p. 41.

[14] "LLVM language reference manual," *LLVM Language Reference Manual.* LLVM Project, Jul. 2022. Available: https://llvm.org/docs/LangRef.html

[15] C. Lattner and V. Adve, "LLVM: A compilation framework for lifelong program analysis and transformation," in *International symposium on code generation and optimization, 2004. CGO 2004.*, 2004, pp. 75–86. doi: 10.1109/CGO.2004.1281665.