GPLMT: A Lightweight Experimentation and Testbed Management Framework

Matthias Wachs[1]([✉]), Nadine Herold[1], Stephan-A. Posselt[1], Florian Dold[2], and Georg Carle[1]

[1] Technical University of Munich (TUM), Boltzmannstr. 3, 85748 Garching, Germany
{wachs,herold,posselt,carle}@net.in.tum.de
https://www.net.in.tum.de
[2] Chair for Network Architectures and Services, Department for Informatics, Technical University of Munich (TUM), Munich, Germany
dold@in.tum.de

Abstract. Conducting experiments in federated, distributed, and heterogeneous testbeds is a challenging task for researchers. Researchers have to take care of the whole experiment life cycle, ensure the reproducibility of each run, and the comparability of the results. We present GPLMT, a flexible and lightweight framework for managing testbeds and the experiment life cycle. GPLMT provides an intuitive way to formalize experiments. The resulting experiment description is portable across varying experimentation platforms. GPLMT enables researchers to manage and control networked testbeds and resources, and conduct experiments on large-scale, heterogeneous, and distributed testbeds. We state the requirements and the design of GPLMT, describe the challenges of developing and using such a tool, and present selected user studies along with their experience of using GPLMT in varying scenarios. GPLMT is free and open source software and can be obtained from the project's GitHub repository.

Keywords: Testbed management · Experimentation

1 Introduction

Network testbeds are an invaluable tool for researchers developing network protocols and networked systems to test a novel approach and existing, already deployed solutions "in the wild". A large variety of testbeds is available to researchers. Many of them focus on a specific domain (e.g. wireless experimentation, high-precision measurements, real-world network testbeds), and most of them use a non-standardized and domain-specific approach to how the testbed is designed, accessed, managed, and experiments are controlled, requiring manual adaptation for every experiment. When trying to transfer such an experiment to a different testbed, the experimenter has to adapt—and most of the time rewrite—the experiment to be able to transfer the experiment to a different platform. This makes it difficult to reproduce and confirm experiment results for both the researcher as well as the research community.

A testbed may be heterogeneous with respect to the hardware and the operating system, and may be physically distributed across more than one location.

This allows the researcher to evaluate reliability and portability under close to real-world conditions. However, an experiment is challenging to manage in a complex testbed. The life cycle of a network experiment comprises tasks such as testbed configuration, resource allocation, experiment definition, and deployment. Many testbed environments, for example PlanetLab, Emulab, or GENI, focus on testbed configuration and resource allocation but do not consider executing the experiment itself. The experiment's execution plan may require assigning different tasks to subsets of nodes in a precise timely manner to control the execution. At the end, the results need to be collected from all nodes. Monitoring and error handling also have to be considered, as resources may become unavailable, or a sub-task may fail. At worst, an experiment lasting several days has to be repeated.

Experiment runs often share similarities, but are still set up manually, or with the help of ad hoc scripts which are rarely reusable. Instead of implementing ad hoc solutions specific to our particular problems, we decided to realize a flexible and extensible testbed and experimentation tool, supporting us in our work and to make it available to the public.

With this work, we present GPLMT, a flexible, lightweight experimentation and testbed management tool. GPLMT provides an intuitive way for users to define experiments, supports the full experimentation life cycle, and allows experiments to be transferred between different testbeds and platforms, ensuring reproducibility and comparability of experiment results. GPLMT is free software and its source code is publicly available on the GPLMT website[1]. In the remainder of this paper we will give an overview of GPLMT, state the requirements and challenges for such a tool, and describe the design and implementation. We also describe the experiences of users working with GPLMT in various scenarios.

2   GPLMT Features

GPLMT is started on a control node and executes a user-supplied XML-based experiment description. GPLMT provides an experiment definition language to define the resources participating in the experiment, the tasks to execute and including specific order and parallelism, and to assign such tasks to resources. In addition, it allows the inclusion of files to reuse experiment definitions and to group resources. GPLMT connects to the nodes via SSH and can be extended with additional communication backends, runs tasks on the nodes, i.e. platform-specific binaries or executable scripts, and can transfer files between the controller and the nodes. To simplify the use of PlanetLab, GPLMT supports importing information about available and assigned nodes from the user's PlanetLab account using the PlanetLab-API. Testbed specific functionality, e.g. setting link properties in the testbed, can be accessed in GPLMT using external scripts if the testbed provides an API. GPLMT offers additional features focusing on handling the intricacies of testbeds: the user can annotate commands with different modes of failure and register arbitrary cleanup actions to, for example, kill processes and delete temporary files.

---

[1] https://github.com/docmalloc/gplmt.

3   Requirements and Challenges

In this section, we highlight the requirements for the design of an experimentation and management tool realizing the features described in Sect. 2 and based on experiences obtained from conducting different types of experiments with various testbeds, exchange with the research community and an analysis of possible use cases varying from managing large scale and unreliable to small virtualization based testbeds.

Self-Containment. GPLMT is intended as a lightweight tool for researchers and experimenters. The tool should neither require a complex experimentation infrastructure, rely on client software like agents installed on testbed nodes nor have requirements for external services like a database server. The tool shall be realized as a portable, platform independent stand-alone tool.

Scalability is important for the experimentation tool to support large-scale testing and experimentation. When conducting experiments with many participants, orchestration and controlling of a large number of different nodes is a challenging task since large delays and setup times have to be prevented.

Resource Restrictions. Experimentation with GPLMT may be limited due to restrictions in the surrounding environment. Establishing a large number of connections to a large number of nodes has to be realized efficiently. Therefore, GPLMT has to be aware of resource restrictions in the host environment and reuse connections and provide rate limiting for new connections being established.

Heterogeneous Testbeds and Nodes. GPLMT has to make experimentation independent from the testbed platform and the participating nodes. Experiments have to be executable in heterogeneous environments with different operating systems and different versions of the operating system.

Fault Tolerance in Unreliable Environments. In real-world and large-scale network testbeds availability of resources cannot always ensured: not all assigned nodes and resources may be available or can fail during an experiment and become available again. GPLMT, therefore, has to cope with unreliable resources and has to provide automatic error handling and recovery transparent to the experiment.

High-level Experiment Definition. With GPLMT experiment definition shall be done on a high level of abstraction, to allow the experimenter to focus on essential aspects of experiment design and control flow without getting distracted by implementation details.

Experiment Reproducibility. Experiment reproducibility is essential for confirmability of experimental results. GPLMT has to support an experiment flow making execution independent from participants, resources, testbeds, external dependencies and state based on a high-level definition of experiments.

Experiment Portability, Reusability and Extensibility. Experiments shall be transferable to other testbeds infrastructures and allow researchers to share

experiment definitions. Employing an abstraction over the testbed infrastructure and using a high-level description of an experiment allows an experiment definition to be reused and to be varied in different scenarios speeding up the testing process.

Grouping Entities in Experiments. In an experiment, tasks and resources may be assigned to different groups of nodes. GPLMT shall provide the functionality to group nodes and resources and to assign tasks to such a group.

Nested Task Execution and Synchronization. Within an experiment, tasks often have to be executed in a specific order or can be executed in parallel. GPLMT shall provide constructs to allow experimenters to specify the execution order of tasks. Tasks may also be nested and grouped in such sequential and parallel constructs. Additional synchronization barriers between the tasks have to be provided.

Repeatable, Periodic and Scheduled Tasks for Experiments. Often tasks inside an experiment have to be executed repeatedly or triggered periodically or at a certain point in time (e.g. for periodic measurements). GPLMT has to provide constructs to express a looping functionality and to schedule tasks to be executed at certain point in time or after a certain duration without adding high complexity.

Error Condition Handling in Experiments. In many cases the experiment control flow depends on successful or failed execution of tasks, making subsequent operations useless or the whole experiment fail. Therefore, GPLMT has to allow the experimenter to define the expected result of a task and how an error condition has to be handled. In addition, functionality to define a clean up and tear down task—executed before the experiment is terminated—is beneficial.

4   GPLMT Design and Implementation

GPLMT is designed as a stand-alone tool running on the so-called *GPLMT controller*. The GPLMT controller is responsible for orchestrating the whole experiment, i.e. scheduling tasks on the hosts of a testbed, from now on called *nodes*. GPLMT manages a connection from the controller to each node. GPLMT does not require any original services on the nodes, but relies on SSH, and possibly other protocols in the future. In addition, GPLMT can use the PlanetLab-API to obtain information about available nodes in the experimenter's PlanetLab slice. An experiment is conducted by passing an experiment description in a high-level description language to GPLMT. The description tells GPLMT which nodes to connect to, which files to exchange, and which tasks to run.

4.1   Resource Management

In large-scale experiments with many nodes, GPLMT will open a large number of connections. SSH is particularly resource-intense. The SSH connection setup is

computationally expensive due to cryptography and may overload a low-powered controller or the physical host of a virtualized testbed. A high rate of connection attempts may stress IDS systems, and may trigger IDS alerts for alleged SSH scanning.

GPLMT offers two solutions to limit its resource usage: *connection reuse* and *rate limiting* of connection attempts. GPLMT will tunnel all commands to the same node through a single control connection, but will still try to reconnect when the connection is lost. GPLMT optionally delays connection attempts, including reconnects, to not exceed a configurable number of attempts per interval.

4.2   Implementation

The GPLMT controller is implemented in Python 3. Besides a few Python libraries and the Python interpreter itself, GPLMT only depends on the external tools which are needed to connect to nodes. Notably, GPLMT wraps OpenSSH, so all features of OpenSSH are available via a local OpenSSH configuration file. GPLMT directly uses OpenSSH's *control master* feature to reuse connections to the same node.

5   GPLMT's Experiment Definition Language

GPLMT provides a domain-specific language to describe the experiment setup and execution. Its syntax is defined in an *XML Schema* obtained from a *relax-ng* definition. Therefore, terms such as *element* and *attribute* refer to the respective XML objects.

The `experiment` root element may contain multiple `include`, `targets`, and `tasklist` elements and a single `steps` element. A `targets` element names the nodes and can also be used to group nodes. `tasklist` defines a set of commands to be run. Both definitions are tied together with the `steps` element, which states which tasklist is to be executed on which targets and at what time.

Target and tasklist definitions are optional and may also be imported from other documents. Targets and tasklists are distinguished and referenced by unique names.

5.1   Targets

A `target` element names a member node, and specifies how to access the node. The following types of targets are currently supported:

- `local` specifies execution on the GPLMT controller itself.
- `ssh` states that the nodes can be accessed using SSH. The child elements `username` and `password` may provide credentials.
- `planetlab` specifies a PlanetLab node and accepts the PlanetLab-API-URL, the slice, and the user name as attributes.
- `group` specifies a nested target definition, creating a set of nodes (and other groups) addressable as a single target.

To support parameterization per target, each target definition can contain multiple `export-env` elements, which declare an environment variable to be exported. The value of this variable is then available to tasks on the target.

5.2 Tasklists

The `tasklist` binds a list of *tasks* to a name. A *task* is one of the following predefined commands:

– `get` and `put` are used to exchange files between the controller and the targets.
– `run` accepts a command to be executed. When a target defines additional environment variables, those are passed to the command using `export-env`.
– The `par` and `seq` elements contain nested lists of tasks. `seq` will run those tasks in order, whereas `par` will immediately start all sub-tasks in parallel.
– `call` is used to reference a tasklist to be executed.

`tasklist` accepts the optional attributes `cleanup`, `timeout`, and `error`, controlling the tasklist's behavior in case of an error condition. `cleanup` references another tasklist to be executed after the current tasklist, even if the current tasklist aborts due to an error. This can be used to kill stale processes and delete temporary files or to save intermediate results. `timeout` specifies the maximum amount of time the tasklist is allowed to execute before it is aborted. This guarantees progress in case a command loops infinitely or dead-locks. `on-error` determines how GPLMT continues when a task fails. The following fail modes are available:

– `abort-tasklist` aborts the current tasklist and continues with the tasklist specified by the surrounding context.
– `abort-step` aborts the current *step* and continues with the next *step*. Steps are explained in Sect. 5.3.
– `panic` aborts the whole experiment.

5.3 Steps

The language requires exactly one `steps` element. It may contain multiple `step`, `synchronize`, `register-teardown`, and `repeat` elements.

The `step` element determines which tasklists run on which target. A start and a stop time can be added to schedule a task for later execution. Times are either relative to the start of the experiment or absolute wall clock times, allowing to defer a step until night-time when resources are available. Thus, `step` elements form the basic building block for orchestrating the experiment.

Consecutive `step` elements run in parallel. A `synchronize` element represents barrier synchronization, and execution can only continue after all currently running steps have finished.

`register-teardown` references a tasklist by name that is executed when `steps` finishes. This tasklist is always executed, even if errors lead to the abortion

of the experiment. The registered tasklist is intended to contain cleanup tasks and to transfer experiment results to the controller. The `register-teardown` cleanup tasklist only needs to be registered right before the `step` that allocates the corresponding resources is issued.

GPLMT's experiment definition language offers basic loops within `steps`: The `repeat` element loops over the enclosed steps until at least one of the following conditions is satisfied:

– a given number of iterations (`iterations`)
– a given amount of time has passed (`during`)
– a given point in time was passed (`until`)

These are deliberately simple conditions that only allow for decidable loops, so it can be easily verified by manual inspection (or programmatically) whether a loop terminates.

5.4   Example

In this section, we present a brief example for a GPLMT experiment to illustrate how experiments are defined. In this experiment, we use GPLMT, running on the controller, to generate network traffic on two nodes and capture this traffic using a third monitoring node. Therefore, nodes *A* (IP 10.0.0.16) and *B* (IP 10.0.0.17) ping each other. The *monitor* collects all network traffic using *tcpdump*. At the end of the experiment, the resulting capture file is transferred to the controller. Listing 1.1 shows a (slightly abbreviated) description for this experiment.

First of all, an external experiment description containing teardown functionality is included (l. 4). Separating functionality in different files eases reuse of frequently used targets and tasklists.

The definition for the three nodes *A* and *B* and *monitor* is done in the `targets` element (ll. 6–23): nodes *A* and *B* are grouped into a target named `pingGroup`. To ping each other, these hosts have to know the partner's IP address which is provided in the environment variable `host`.

The experiment workflow is defined in the `steps` element (ll. 37–45). The different `step` elements reference tasklists from the `tasklists` element (ll. 25–35). The experiment starts with instructing the monitor node to capture network traffic using `tcpdump` (l. 38) using tasklist `createPCAP` (l. 26). To ensure `tcpdump` is terminated at the end of the experiment, the experiment registers tasklist `stopMonitoring` (l. 39), imported from a file (l. 4). Both tasklists, `createPCAP` and `stopMonitoring`, are executed in parallel.

The `synchronize` statement (l. 41) ensures monitoring is started before the nodes in group `pingGroup` (ll. 11–22) begin to ping each other (l. 42). Both nodes execute the same tasklist `doPing` (ll. 29–31). The shell on respective node expands the variable `host` (on l. 27) set to the other host's IP address (ll. 15,20).

The `synchronize` statement (l. 43) blocks until the `doPing` tasklists have finished (l. 30). The final step (l. 44) copies the captured traffic from the monitor node to the controller.

Listing 1.1. Example: Generate and Monitor Network Traffic with GPLMT

```xml
1  <?xml version="1.0" encoding="utf-8" ?>
2  <experiment>
3
4    <include file="include/teardowns.xml" />
5
6    <targets>
7      <target name="monitor" type="ssh">
8        <user>testaccount</user>
9        <host>monitor.example</host>
10     </target>
11     <target name="pingGroup" type="group">
12       <target name="A" type="ssh">
13         <user>testaccount</user>
14         <host>10.0.0.16</host>
15         <export-env var="host" value="10.0.0.17" />
16       </target>
17       <target name="B" type="ssh">
18         <user>testaccount</user>
19         <host>10.0.0.17</host>
20         <export-env var="host" value="10.0.0.16" />
21       </target>
22     </target>
23   </targets>
24
25   <tasklists>
26     <tasklist name="createPCAP">
27       <run>tcpdump -i eth0 -w testrun.pcap &</run>
28     </tasklist>
29     <tasklist name="doPing">
30       <run>ping $host -c 10</run>
31     </tasklist>
32     <tasklist name="getData">
33       <get>testrun.pcap</get>
34     </tasklist>
35   </tasklists>
36
37   <steps>
38     <step tasklist="createPCAP" targets="monitor" />
39     <register-teardown ref="stopMonitoring"
40       targets="monitor" />
41     <synchronize />
42     <step tasklist="doPing" targets="pingGroup" />
43     <synchronize />
44     <step tasklist="getData" targets="monitor" />
45   </steps>
46  </experiment>
```

6   User Studies

In the following section, we present an overview of projects using GPLMT to show the various different use cases and purposes GPLMT can be used for and highlight the challenges emerging with respect to both experimentation as well as using the GPLMT framework. Based on these experiences, we modified GPLMT in the current version to cope with this challenges.

6.1   The GNUnet Project - Large-Scale Software Deployment in Heterogeneous Testbeds

GNUnet[2] is a GNU free software project focusing on a future, decentralized Internet. GNUnet develops the GNUnet peer-to-peer (P2P) framework to allow developers to realize decentralized networking applications.

GNUnet employs GPLMT to deploy the GNUnet framework to a large number of PlanetLab nodes to be able to test the software under real-world conditions and to support bootstrapping of the network. GNUnet's requirement was to compile the latest GNUnet version on PlanetLab nodes directly.

GNUnet used GPLMT to provide the nodes with all software dependencies required. While running, GNUnet was monitored to analyze the behavior of the software and the P2P network and to obtain log files in case of a crash. With GPLMT detailed information for every node could be obtained.

For GNUnet, the major challenge was the unreliability and heterogeneity of the PlanetLab testbed. With a large number of nodes only a fraction were accessible and working correctly. PlanetLab nodes only provide outdated software and are very heterogeneous both with respect to versions of the operating system and version of software installed. Nodes also often get unavailable during operation.

6.2   OpenLab Eclectic - A Holistic Development Life Cycle for P2P Applications

The OpenLab Eclectic Project[3] focused on developing a holistic development life cycle for distributed systems by closing the gap between the testbed and the P2P community.

Eclectic used GPLMT to orchestrate, control and monitor networking, P2P testing, and experimentation on different testbeds. GPLMT's functionality to define experiments and to interact with testbeds using an abstraction layer allowed Eclectic to deploy distributed systems on local systems, HPC systems like the SuperMUC[4] and Internet testbeds like PlanetLab.

The main challenge for Eclectic was to define testbed independent experiments to be able to transfer experiments between different testbeds. GPLMT was

---

[2] https://gnunet.org.

[3] http://www.ict-openlab.eu/experiments-use-cases/experiments.html.

[4] https://www.lrz.de/services/compute/supermuc/.

also used to setup network nodes and collect experimental results. Within this project, GPLMT was integrated with the Zabbix[5] network monitoring solution to provide an integrated approach for infrastructure monitoring and experiment scheduling.

6.3   Testbed Management for Attack and Defense Scenarios

Datasets to train and test Intrusion Detection Systems (IDS) under realistic and reproducible conditions are hard to obtain and generate. Such datasets have to provide a high diversity of attacks with a high packet frequency but also have ensure reproducible results and provide a clear labeled information about the data flows.

At TUM's chair for network architectures and services, researchers used GPLMT to generate such datasets with different attack scenarios. To generate such datasets, a virtualized testbed environment with virtual machines grouped into attackers, victims and monitoring machines was used. These machines were used to execute attacks as well as provide defense mechanisms and obtain the generated network traffic. In addition, this testbed was used to evaluate the quality of port scanners and port scan detection tools with the results being collected and interpreted afterwards.

The main challenge was the grouping of the different entities, as well as the complex interaction and nesting of tasks assigned to the entities. Timing aspects as well as synchronization were crucial to this setting. The monitoring and generation of test datasets during the experiment executions was an additional challenge to be mastered.

6.4   Distributed Internet Security Analysis

In [1], security researchers developed a distributed, PlanetLab-based approach to conduct large-scale scans of today's TLS deployment in the wild. They used PlanetLab nodes to perform distributed scans of large IP ranges and analyzed the TLS certificates found on hosts. To conduct these scans, GPLMT was used to deploy the scanning tool used to the PlanetLab nodes, orchestrate the measurements, and obtain results from the nodes.

A major challenge in this use case was long lasting scan experiments in combination with the large number of parallel SSH connections established to PlanetLab nodes. The organization's intrusion detection system detected these connections as a malicious attack and blocked the control node as the source of these connections on the network as a consequence.

The main challenge was the large number of connections to the PlanetLab nodes. First, those connections had to be throttled during the experiment. Apart from this, the number of connections established had to be managed.

---

[5] http://www.zabbix.com/.

7 Related Work

Various different tools exist to manage and control network experiments. A rather extensive list can be found on the PlanetLab website[6]. [1] provides a comprehensive analysis with respect to quality and usability of such tools, finding most of them not usable or suitable to be used with respect to today's network experiments. Many of these tools are outdated and not available anymore (Plush, Nebula, Plman, AppManager) or were not even made publicly available at all (PLACS). Some of these tools provide rather basic functionality to invoke commands on remote nodes (pssh, pshell, vxargs) not supporting error conditions and error handling as well as orchestrating nodes to perform complex and synchronized operations. The Stork project[7] provides a deployment tool for PlanetLab nodes including configuration. This tool lacks fine-grained execution control to setup more complex experiments. Gush (GENI User Shell) [2] claims to be an execution management framework for the GENI testbed. Gush provides extensive methods to define resources but is limited regarding control flow aspects. Parallel or sequential execution is not possible in a straight forward manner. In addition, Gush is not longer supported[8].

Experimentation frameworks like NEPI [3] require the user to do rather complex adaptations in the source code to extend it with new functionalities and add support for new platforms. Approaches like OMF [4] focus on the management and operation of network testbed infrastructures and federation between infrastructures not focusing on the experiment part in the life cycle.

The COCOMA framework [5] focuses on providing an experimentation framework for cloud based services to control and execute tests for cloud based services in a controlled and reproducible manner and to study resource consumption of such services. [6] proposes an emulated testbed for the domain of cyber-physical systems. This work focuses more on the testbed implementation and less on the execution of experiments.

8 Future Work

For future versions, we plan to decouple the GPLMT controller from the experimenter's host and instead run GPLMT as a service on a dedicated control node. Users would then submit experiments to the *experiment queue* of a testbed, which is managed by GPLMT. This would ease the use of shared testbeds. Future versions of GPLMT may support target types other than SSH and PlanetLab, for example *mobile devices.* An intuitive user interface would ease experiment monitoring and control. This feature was provided based on Zabbix in an earlier version of GPLMT but is not available at the moment due to a recent refactoring of the code base.

---

[6] https://www.planet-lab.org/tools.
[7] http://www.cs.arizona.edu/stork/.
[8] http://gush.cs.williams.edu/trac/gush.

9   Conclusion

The focus of GPLMT is to provide a lightweight and convenient way for experimenters to conduct network experiments and manage testbed environments. Instead of using handcrafted onetime scripts for every experiment, we envision GPLMT to be flexible tool usable for different scenarios and use cases. Using a high-level description language GPLMT offers opportunities to share experiment descriptions among researchers and supports closer collaborations between experimenters. Moreover, GPLMT's language was designed to support error handling, nested execution flows and different timing aspects to provide a high level flexibility and adaptability. GPLMT is still under active development and will be extended in the future. With this work, we want to present GPLMT to the community and make it available for a broad audience. GPLMT is free software and can be obtained from the repository[9]. Both feedback as well as contributions from the community are highly appreciated.

References

1. Jaros, M.: Distribution and orchestration of network measurements on the planetlab testbed. Bachelor's thesis, Technische Universität München, Chair for Network Architectures and Services, April 2015
2. Albrecht, J., Huang, D.Y.: Managing distributed applications using gush. In: Magedanz, T., Gavras, A., Thanh, N.H., Chase, J.S. (eds.) TridentCom 2010. LNICST, vol. 46, pp. 401–411. Springer, Heidelberg (2011)
3. Quereilhac, A., Lacage, M., Freire, C., Turletti, T., Dabbous, W.: Nepi: an integration framework for network experimentation. In: 19th International Conference on Software, Telecommunications and Computer Networks (SoftCOM), pp. 1–5, September 2011
4. Rakotoarivelo, T., Ott, M., Jourjon, G., Seskar, I.: OMF: a control and management framework for networking testbeds. ACM Oper. Syst. Rev. (OSR) **43**, 54–59 (2010)
5. Ragusa, C., Robinson, P., Svorobej, S.: A framework for modeling and execution of infrastructure contention experiments. In: 2nd Internation Workshop on Measurement-based Experimental Research, Methodology and Tools (2013)
6. Genge, B., Siaterlis, C., Fovino, I.N., Masera, M.: A cyber-physical experimentation environment for the security analysis of networked industrial control systems. Comput. Electr. Eng. **38**(5), 1146–1161 (2012). Special issue on Recent Advances in Security and Privacy in Distributed Communications and Image processing

---

[9] https://github.com/docmalloc/gplmt.