WASP WINTER CONFERENCE 2022

Poster Catalogue

SOFTWARE

WASP | WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

WASP WINTER CONFERENCE 2022
POSTER CATALOGUE 4/4

SOFTWARE

Andersson, Pontus
Lund University / NVIDIA

WASP | WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

Evaluating Differences Between Rendered Images and Videos

In rendering research and development, it is important to have a formalized way of visualizing and communicating how and where errors occur when rendering with a given algorithm. Such evaluation is often done by comparing the test image or video to a ground-truth reference. We have presented a tool for comparing both low and high dynamic range images. Our tool is based on a perception-motivated image metric. Now, we are exploring how to extend that metric to also convey the differences in rendered videos, an extension that poses several challenges, as the presence of perceptual effects increase significantly when we consider spatiotemporal stimuli.

Andersson, Pontus
Lund University / NVIDIA

WASP WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

Evaluating Differences Between Rendered Images and Videos

Pontus Andersson, Lund University
Centre for Mathematical Sciences

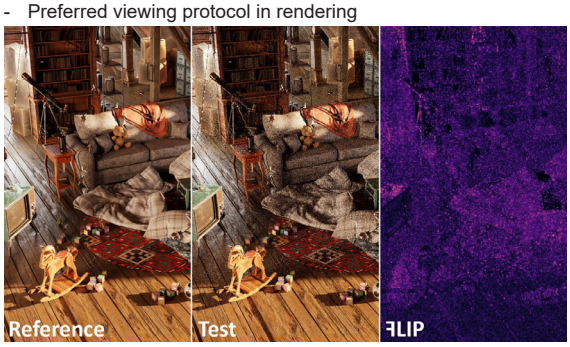
Abstract

In rendering research and development, it is important to have a formalized way of visualizing and communicating how and where errors occur when rendering with a given algorithm. Such evaluation is often done by comparing the test image or video to a ground-truth reference. We have presented a tool for comparing both low and high dynamic range images. Our tool is based on a perception-motivated image metric. Now, we are exploring how to extend that metric to also convey the differences in rendered videos, an extension that poses several challenges, as the presence of perceptual effects increase significantly when we consider spatiotemporal stimuli.

Image Differences: ꟻLIP [1, 2]

Evaluates and visualizes the perceived differences observed when alternating between images

- Removes unperceivable details (spatial filtering)
- Perceptually uniform color space
- Enhancing edge and point errors
- Preferred viewing protocol in rendering



User study: How well does the error map correspond to the errors you perceive? Results showed that ꟻLIP corresponds significantly better to the perceived error than any of the other metrics in the study.

Through an extension, ꟻLIP handles both high and low dynamic range imagery [2]

ꟻLIP @ GitHub

Video Differences: ?

Sequences of images are the main focus in rendering. Importantly, errors that are imperceptible in static images can be noticeable in videos, and vice versa.

Use cases for a video metric:

- Replace user studies and other expensive video comparisons
- Objective assessment of video generation algorithms
- Cost functions in, e.g., deep-learning-based techniques

Challenges in developing a video metric:

- *Motion*: How do we perceive errors on moving objects?
- *Flicker*: How can we determine if flickering is present?
- *Complexity*: A second of video could contain 240+ images
- *Evaluation*: Flipping not possible. What can we do instead?

Our research targets a new perception-based video metric:

- Current subproject: Temporal Edge Detection (TED):
  - Estimate the temporal edge detection filters in human vision
  - This has been done for spatial edge detection filters [3]
  - Result used to compare perceptible flicker between videos

References

1. Andersson, P., Nilsson, J., Akenine-Möller, T., Oskarsson, M., Åström, K., & Fairchild, M. D. (2020). ꟻLIP: A Difference Evaluator for Alternating Images. *Proceedings of the ACM on Computer Graphichs and Interactive Techniques*, *3(2)*, 15:1-15:23.
2. Andersson, P., Nilsson, J., Shirley, P., & Akenine-Möller, T. (2021). Visualizing Errors in Rendered High Dynamic Range Images. In *Eurographics Short Papers*.
3. McIlhagga, W. (2018). Estimates of Edge Detection Filters in Human Vision. *Vision Research*, *153*, 30-36.

Couderc, Noric
Lund University

WASP WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

Building an data-structure selection tool for Java programs

Writing programs require using both algorithms and data-structures. Most programming languages provide implementations for some ”classical” data-structures (lists, maps, and sets). The developer chooses which implementations to use. Unfortunately, programmers are not that good at picking the data-structure that minimizes runtime.
There is existing work which uses machine learning to provide suggestions to C++ developers. Adapting this work to Java poses new challenges. In this poster, we evaluate how effective this tool is on Java programs.

Couderc, Noric
Lund University

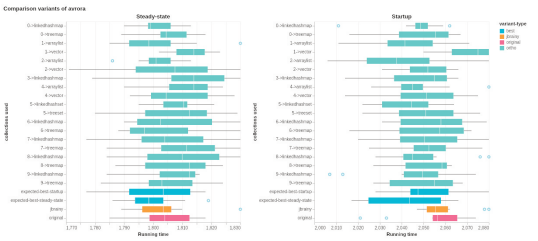
WASP WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

Noric Couderc, Lund University

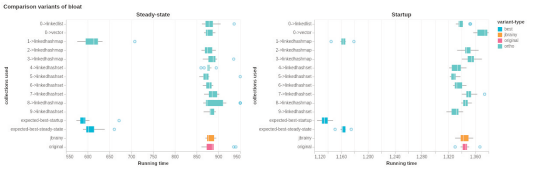
LUNDS UNIVERSITET

I used machine learning to make Java programs faster,
It didn't really work

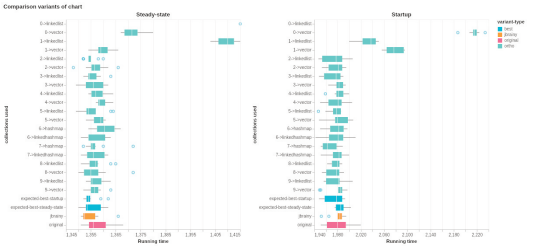
My tool doesn't improve much over the original, exhaustive search does slightly better



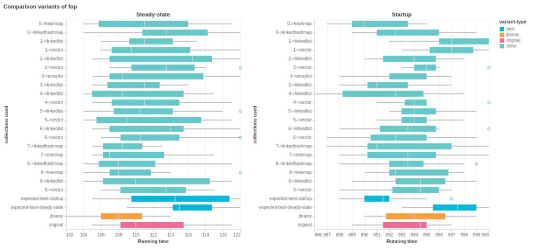
Exhaustive search finds a massive optimization, but my tool misses it



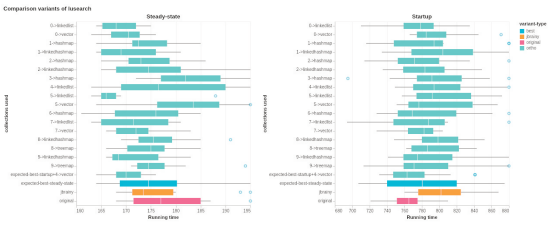
My tool doesn't improve over the original, neither does exhaustive search



My tool does slightly better than exhaustive search on steady state performance



My tool does worse on startup than the original program



This is ok
It's a reproduction study!
Paper with details coming soon!

WASP WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

Etemadi, Khashayar
KTH

Estimating the Potential of Program Repair Search Spaces with Commit Analysis

The most natural method for evaluating program repair systems is to run them on bug datasets, such as Defects4J. Yet, usingthis evaluation technique on arbitrary real world software projects requires heavy configuration. In this paper, we propose a newmethod, which is purely static, to evaluate the breadth of the search space of repair approaches. Our key insight is to encode thesearch spaces of repair approaches by specifying the repair strategies they employ. Next, we use the specifications to check whetheror not past commits lie in repair search spaces. For a repair approach, including many human-written past commits in its searchspace indicates its potential to generate useful patches. We implement our evaluation method in a tool, called LighteR. LighteRgets a Git repository as input and outputs a list of commits whose corresponding source code changes lie in the search spaces ofrepair approaches. Using LighteR, we conduct a study on 55,309 commits from the history of 72 Github repositories and showthat the precision and recall of LighteR are 77% and 92%, respectively. Overall, our experiments show that our novel method isboth lightweight and effective to study the search space of program repair approaches.

Etemadi, Khashayar
KTH

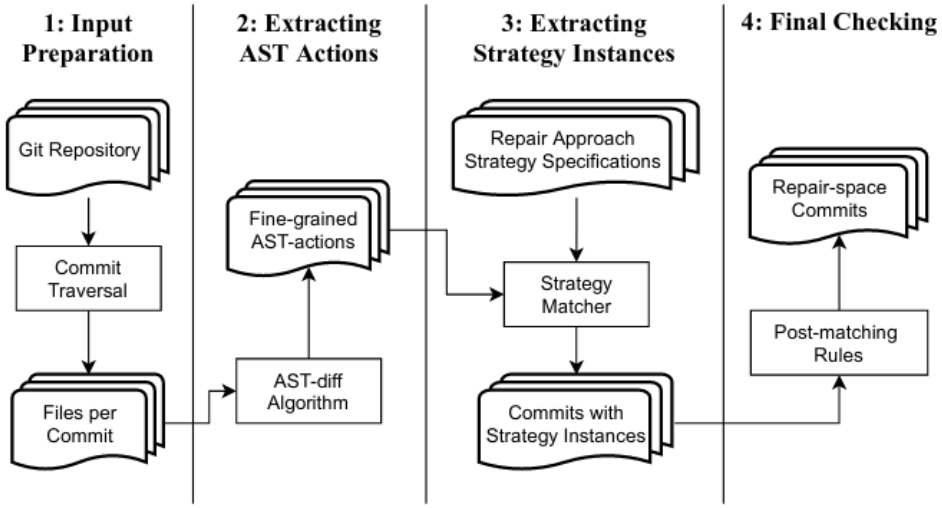
WASP WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

Estimating Program Repair Potential with Commit Analysis

Khashayar Etemadi, khaes@kth.se
Theoretical Computer Science Department @ EECS

Context: Running program repair tools is usually very costly [1]. Therefore, it is not an efficient way to assess the strength of program repair tools.
Contribution: We propose a new purely static method, which is purely static, to evaluate the breadth of the search space of repair approaches. Our key insight is to encode the search spaces of repair approaches by specifying the repair strategies they employ. Next, we use the specifications to check whether or not past human-made commits lie in repair search spaces. Our method is implemented in LighteR, with precision and recall of 77% and 92%, respectively. We find that 1.35% of 55,309 commits from 72 projects lie in search spaces of eight considered tools.

Overview of LighteR's Approach



Identifying Repair-space Commits:

- Uses GumTree [2] to extract AST actions.
- Uses Coming [3] to match actions with specification.
- Uses post-matching rules to discard non-synthesizable patches.

Considered Tools:

Arja, Cardumen, Elixir, GenProg, jMutRepair, Kali, Nopol, NPEfix

Actionable Implications:

- Prototyping of New Repair Approaches by Researchers
- Evaluation of the Potential Value of Using Program Repair by Practitioners

Sample Strategy Specification for GenProg

```
<pattern name="genprog_simple">
    <entity id="1" type="*" role="Statement"/>
    <action entityId="1" type="*"/>
</pattern>
```

References

1- Durieux, Thomas, et al. "Empirical review of Java program repair tools: A large-scale experiment on 2,141 bugs and 23,551 repair attempts." Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2019.
2- Falleri, Jean-Rémy, et al. "Fine-grained and accurate source code differencing." Proceedings of the 29th ACM/IEEE international conference on Automated software engineering. 2014.
3- Martinez, Matias, and Martin Monperrus. "Coming: A tool for mining change pattern instances from git commits." 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion). IEEE, 2019.

WASP WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

Gissurarson, Matthías Páll
Chalmers

WASP | WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

Using Typed-Holes in Haskell for Property-Based Automatic Program Repair

This poster presents the use of typed-hole synthesis in PropR, a property-based automatic repair tool for Haskell that uses genetic programming to automatically repair Haskell programs. Haskell programs are often annotated with very specific types and come with a large suite of property-based tests in addition to unit tests. Using those properties and unit tests, we can isolate the parts of the code involved in a failing test case, and then we can leverage the available type information by integrating with the valid hole-fit synthesis in the GHC compiler to do accurate synthesis of well-typed programs as possible repairs for the fault-involved expressions.

Gissurarson, Matthías Páll
Chalmers

WASP WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

Using Typed-Holes in Haskell for Property-Based Automatic Program Repair

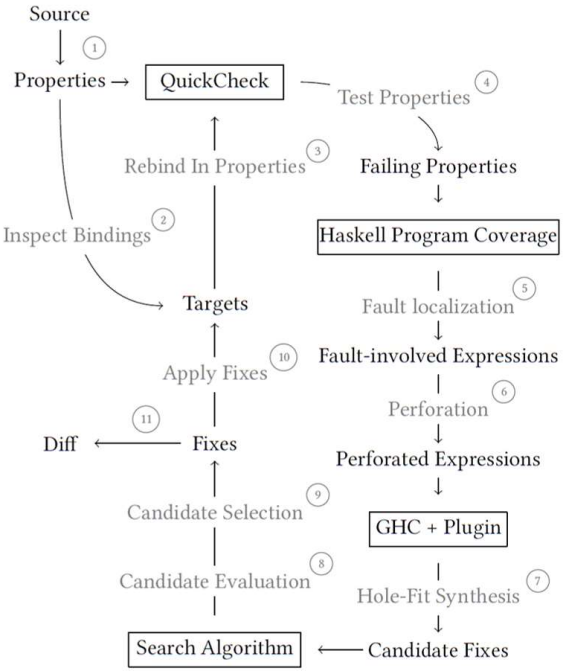
Matthías Páll Gissurarson,
Chalmers University of Technology
Department of Computer Science and Engineering

Abstract

We present the use of typed-hole synthesis in PropR, a property-based automatic repair tool for Haskell that uses genetic programming to automatically repair Haskell programs.

PropR: Property-Based Automatic Program Repair

Haskell programs are often annotated with very specific types and come with a large suite of property-based tests in addition to unit tests. Using those properties and unit tests, we can isolate the parts of the code involved in a failing test case, and then we can leverage the available type information by integrating with the valid hole-fit synthesis in the GHC compiler to do accurate synthesis of well-typed programs as possible repairs for the fault-involved expressions [1].



The Test-Localize-Synthesize-Rebind Loop

PropR is based on a Test-Localize-Synthesize-Rebind loop (TLSR), shown in the figure to the right [1]. We begin by parsing the source code and discovering ***properties***[1], as defined by the name and/or type of the functions in the provided source ①. By inspecting these properties ②, we determine top-level *targets* for repair. We rebind these to mutable expressions (making no changes initially, keeping the original target intact) ③. We test the properties using QuickCheck [2] to determine which ones fail ④, and use the generated counter-examples to determine the *fault-involved sub-expressions* in the targets ⑤. Then we *perforate* the targets by replacing each fault-involved sub-expression with a ***typed-hole***[2] ⑥. By using GHCs built-in ***valid hole-fit***[3] synthesis in conjunction with a hole-fit plugin, we synthesize candidate fixes in place of these holes based on their type ⑦ and mined expressions from the source code. We then evaluate the candidates by replacing the holes in the targets with candidate fixes ⑧ and apply genetic search to *select* ⑨ potential fixes based on how well the program would perform on the test-suite if they were to be applied. After selection, we apply the selected fixes to the program ⑩ by replacing sub-expressions in the targets using the selections. If all properties are now satisfied, we've found a repair, and output it as a diff ⑪, otherwise we apply the current fixes ③ to the program and repeat until we succeed or run out of search budget [1].

[1] ***Properties*** *are testable assertions in the code. They can either be unit-tests (i.e. an input-output pair) or more generic, e.g.* prop_flsPositive n = (f n) > 0, *is tested by generating random n and checking that the property holds. When QuickCheck finds a value for which the property does not hold, it* shrinks *it (i.e. minimizes it) and returns it as a counter-example [2].*

[2] *A* ***typed-hole*** *is a placeholder value for unknowns in code, and include a type inferred from the context and other constraints based on where it is placed in the source code. A typed-hole is represented by an underscore* (_) *in Haskell source-code.*

[3] *A* ***valid hole-fit*** *is an expression that matches the type of the hole. In GHCs, these can be synthesized by the compiler, either a simple fit (e.g.* sum *for* (_ :: [Int] -> Int)) *or a more complex* refinement hole-fit *(e.g. (*foldl (+) :: [Int] -> Int) [3].

References

1. Gissurarson, M. P., Leonhard, A., Panichella, A., Deursen, A., Sands, D. 2022. PropR: Property-Based Automatic Program Repair. To be published at the 44th International Conference on Software Engineering (ICSE 2022), Pittsburgh, PA, USA.
2. Claessen, K., & Hughes, J. 2000. QuickCheck: a lightweight tool for random testing of Haskell programs. In *Proceedings of the fifth ACM SIGPLAN international conference on Functional programming* (ICFP '00), Montreal, Canada.
3. Gissurarson, M. P. 2018. Suggesting Valid Hole Fits for Typed-Holes (Experience Report). In Proceedings of the 11th ACM SIGPLAN International Haskell Symposium (Haskell '18), St. Louis, MO, USA.

Hrusto, Adha
Lund University

Closing the Feedback Loop in DevOps Through Autonomous Monitors in Operations

DevOps represent the tight connection between development and operations. To address challenges that arise on the borderline between development and operations, we conducted a study in collaboration with a Swedish company responsible for ticket management and sales in public transportation. The aim of our study was to explore and describe the existing DevOps environment, as well as to identify how the feedback from operations can be improved, specifically with respect to the alerts sent from system operations. Therefore, we design a solution to improve the alert management by optimizing when to raise alerts and accordingly introducing a new element in the feedback loop, a smart filter. Moreover, we implemented a prototype of the proposed solution design using a hybrid method that combines rule-based and unsupervised machine learning for operations data analysis.

Hrusto, Adha
Lund University

WASP WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

Closing the Feedback Loop in DevOps Through Autonomous Monitors in Operations

Adha Hrusto, Lund University
Department of Computer Science

LUND UNIVERSITY LTH FACULTY OF ENGINEERING

SYSTEM VERIFICATION

Abstract

DevOps represent the tight connection between development and operations. To address challenges that arise on the borderline between development and operations, we conducted a study in collaboration with a Swedish company responsible for ticket management and sales in public transportation. The aim of our study was to explore and describe the existing DevOps environment, as well as to identify how the feedback from operations can be improved, specifically with respect to the alerts sent from system operations. Therefore, we design a solution to improve the alert management by optimizing when to raise alerts and accordingly introducing a new element in the feedback loop, a smart filter. Moreover, we implemented a prototype of the proposed solution design using a hybrid method that combines rule-based and unsupervised machine learning for operations data analysis.

Research approach

Our study is a problem-driven design science approach as shown in *Figure 1*. We explored how the general problem, of incorporating feedback from **operations** in the **development**, manifests as a problem instance in the industrial context under study. For that purpose, we conducted *interviews* and performed *observations* in the case company to identify and articulate the main problems on which to focus further improvements. In the **problem conceptualization** step, we identified *three* problem instances related to *alert flooding*, which is a phenomenon that appears in a case of a high number of alerts that are not properly managed. We provided a **conceptual design** for only one of the problem instances, alert flooding as an optimization problem since it causes the highest information overflow in the feedback loop. Moreover, alongside the proposed solution design, we implemented a **prototype instance** to get a better understanding of the opportunities of the available operations data, its type, and characteristics as well as the constraints of the context. We partially **evaluated** the implemented solution using the limited data set for implementation of the baseline anomaly detection method in a prototype environment.

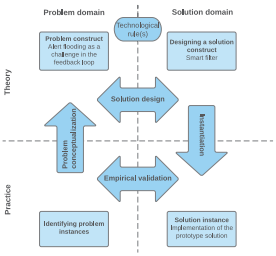


Figure 1 Overview of the Design Science approach

CASE DESCRIPTION

The system under study is a backend system of an **application** for **ticketing** and **payments** used in public transportation. It is a cloud-based system with a microservice architecture that consists of 20 services, developed using **Microsoft** tools and services. The health status of each service is monitored using the Azure Monitor through which various performance metrics and logs are available to use for alerting and visualization.

Selected Results

Problem conceptualization. We identified alert *targeting*, signal to noise *optimization*, and system *interoperability* as being three important problem instances of the general alert flooding problem in the feedback from operations to development.

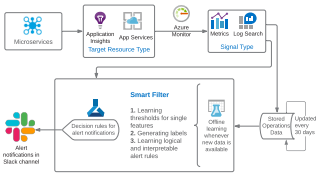


Figure 2 Overview of the proposed solution design

Solution design. We designed a solution for more effective processing of data available through the monitoring system in operations by introducing a smart filter in the feedback loop as shown in *Figure 2*. The smart filter is a unique technical solution that combines various systems' and applications' metrics for learning advanced alert rules (*see Table 1*).

Prototype implementation. We performed a pilot implementation of the proposed solution in the case environment as a proof of concept for further work. In the implementation of the prototype solution, we used unsupervised anomaly detection throughout the labeling process of unlabeled operations data while also considering the service vulnerability and observed metrics frequency (*see Table 1*). Further, for generating new advanced alert rules, a supervised tree-based machine learning technique was used.

Table 1 Overview of the selected data, service vulnerabilities and desired decision rules

Selected metrics	CPU Time	Num. of failed requests	Num. of exceptions	Num. of dep. failures	Http 4xx errors	Http 5xx errors	Num. of requests	Response time
Services with known vulnerabilities	Service B – buying tickets on vending machines		Service G – service for validating selected locations		Service M –> main service for ticketing		Service P –> bridge to an external payment service	
Example of a decision rule	**IF** num_of_failed_requests_SG > threshold_1 **AND** response_time_SB > threshold_2 **AND** num_of_Http500_SB > threshold_3 **THEN** send_notification							

Evaluation. We also implemented multivariate anomaly detection (MAD) to validate our prototype by comparing it with the pure unsupervised ML technique for detecting outliers, representing alerts, in multivariate unlabeled data set. The results revealed that the **MAD** trained model **does not scale** very well the number of predicted alerts, thus, producing the same level of noise and several alert floods. On the other hand, the **smart filter** produces **less noise** around actual failures and more accurately predicts isolated alerts in case of short system's glitches.

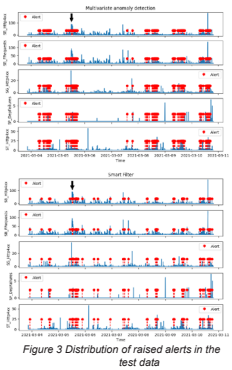


Figure 3 Distribution of raised alerts in the test data

References

1. Adha Hrusto, Per Runeson, and Emelie Engström (2021). **Closing the Feedback Loop in DevOps Through Autonomous Monitors in Operations.** SN Computer Science 2, 6 (Aug. 2021). https://doi.org/10.1007/s42979-021-00826-y

Nilsson, Alexander
Lund University

Timing-Attacks on Post-Quantum Cryptographic Primitives

Vulnerabilities found in crypto algorithms “HQC” and “BIKE”
Next-generation public-key encryption algorithms “HQC” and “BIKE”, which are code-based key encapsulation mechanisms, share a vulnerability due to the use of Rejection Sampling in their decapsulation mechanisms.
An attacker can use this vulnerability to craft special messages by which a complete secret-key recovery can be achieved.
The time-complexity of this attack is low enough to be practically managed within a couple of days, in an ideal lab scenario.

Nilsson, Alexander
Lund University

WASP WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

Timing-Attacks on Post-Quantum Cryptographic Primitives

Alexander Nilsson, Lund University

Dept. Electrical and Information Technology, Faculty of Engineering LTH

advenica

Vulnerabilities found in crypto algorithms “HQC” and “BIKE”

Next-generation public-key encryption algorithms “HQC” and “BIKE”, which are code-based *key encapsulation mechanisms,* share a vulnerability due to the use of *Rejection Sampling* in their decapsulation mechanisms.

An attacker can use this vulnerability to craft special messages by which a complete secret-key recovery can be achieved. The time-complexity of this attack is low enough to be practically managed within a couple of days, in an ideal lab scenario.

Post-Quantum Cryptography

Quantum computers threaten to break most of the encryption in use over the internet today. Therefore, new algorithms are needed.

NIST is currently in an open process to standardize a small number of new public-key primitives for encryption and digital signatures. Among the few remaining candidates are the code-based key encapsulation mechanisms “HQC” and “BIKE”.

KEM/PKE Scheme	Type	Vulnerable to Rejection Sampling Timing Attack
Finalists		
Classic McEliece	*Code-based*	NO
Kyber	Lattice	NO
NTRU	Lattice	NO
SABER	Lattice	NO
Alternates		
BIKE	*Code-based*	YES
FrodoKEM	Lattice	NO
HQC	*Code-based*	YES
NTRU Prime	Lattice	NO
SIKE	Supersingular elliptic curve isogeny	NO

Rejection Sampling

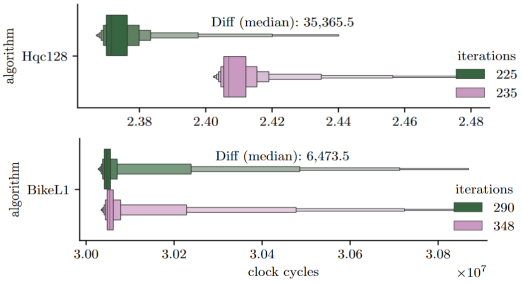
The Hamming Weight $hw(v)$ denotes the number of set bits in the bit vector v.

Both BIKE and HQC need to generate uniformly random error vectors e, where $hw(e) = T$, where T is a constant parameter.

Rejection sampling does this by iteratively generating random bit positions and rejecting invalid positions. This is an efficient way of ensuring uniform randomness of specific weights.

It is, however, very hard to implement it such that the run-time, or number of *samplings,* of the algorithm does not depend on its input (such as the seed for the random number generator).

Because of this the input to the rejection sampling algorithm must be comprised entirely of public values, or be derived from entirely public values.



The Vulnerability [ref]

An important property in code-based schemes is the malleability of the ciphertexts. This means that it is inherently possible to slightly modify the ciphertext by a small amount and still decrypt to the very same plaintext.

This is an undesired property of highly secure KEM/PKE schemes, and the most common way to resolve the issue has been shown to open up PKE/KEM schemes to hitherto unknown timing attacks. [1]

In the current work [2] we show how to adapt the known attack to apply also to the Rejection Sampling algorithm employed by BIKE and HQC. In this work we rely on techniques from [3] and [4].

References

1. Qian Guo, Thomas Johansson, and **Alexander Nilsson**. "A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM." *Annual International Cryptology Conference*. Springer, Cham, 2020.
2. Clemens Hlauschek, Norman Lahr, Robin Leander Schröder, Qian Guo, Thomas Johansson, and **Alexander Nilsson,** "Key recovery attacks on BIKE and HQC, due to Rejection Sampling”
3. **Alexander Nilsson**, Thomas Johansson, and Paul Stankovski. "Error Amplification in Code-based Cryptography." IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES) 2019.1 (2018): 238-258.
4. Qian Guo, Thomas Johansson, and Paul Stankovski. "A key recovery attack on MDPC with CCA security using decoding errors". ASIACRYPT 2016, Springer, Heidelberg, December 2016

Riouak, Idriss
Lund University

WASP | WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

A Precise Framework for Source-Level Control-Flow Analysis

Static program analysis plays a fundamental role in software development and may help developers detect subtle bugs such as null pointer exceptions or security vulnerabilities. We present IntraCFG, a language-independent framework for constructing precise intraprocedural control-flow graphs (CFGs) superimposed on the Abstract Syntax Tree (AST). Source-level dataflow analysis permits easier integration with the IDEs and Cloud tools since the reports can be directly linked to the source code and do not require producing the Intermediate Representation (IR).

Riouak, Idriss
Lund University

WASP WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

A Precise Framework for Source-Level Control-Flow Analysis

Idriss Riouak*, Christoph Reichenbach*, Görel Hedin*, and Niklas Fors
*idriss.riouak, christoph.reichenbach, gorel.hedin, and niklas.fors (@cs.lth.se)
Department of Computer Science, Lund University, Sweden

LUND UNIVERSITY

DESCRIPTION

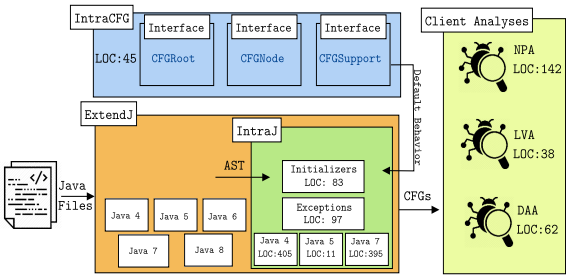
Static program analysis plays a fundamental role in software development and may help developers detect subtle bugs such as null pointer exceptions or security vulnerabilities. We present IntraCFG, a language-independent framework for constructing precise intraprocedural control-flow graphs (CFGs) superimposed on the Abstract Syntax Tree (AST). Source-level dataflow analysis permits easier integration with the IDEs and Cloud tools since the reports can be directly linked to the source code and do not require producing the Intermediate Representation (IR).

OUR APPROACH

We build the CFGs on top of the AST using Reference Attribute Grammars (RAGs). Highlights of our approach:

- Handles implicit control flow
- Fully declarative specification using JastAdd2
- Overcomes the limitations of an earlier RAG framework, eliminating *misplaced* and *redundant* nodes in the constructed CFGs.

FRAMEWORK



INTERFACE	ASTNODE
CFGRoot	MethodDecl, ConstructorDecl, ...
CFGSupport	WhileStmt, IfStmt, ...
CFGNode	All the ASTNodes that might appear in the CFGs.

The **IntraCFG** interfaces provide client APIs for the successor and predecessor relations, and default behaviour that simplifies constructing CFGs for a specific language. We used **IntraCFG** to construct high-precision CFGs for Java 7, extending the ExtendJ Java compiler.

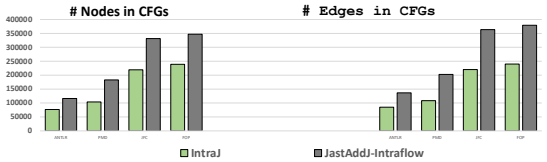
EXPERIMENTS

We compared the results of IntraJ with:

- JastAddJ–Intraflow (JJI): a RAG based framework
- SonarQube: a highly tuned static analyser

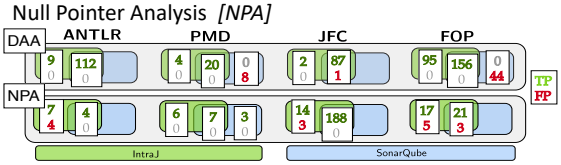
We used as benchmarks:

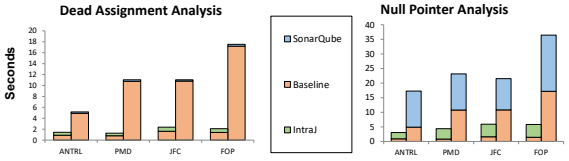
- CFG size reduced by 30-40%



We compared the precision and the performance of IntraJ against SonarQube by implementing two dataflow analyses:

- Dead Assignment Analysis *[DAA]*
- Null Pointer Analysis *[NPA]*





- Higher precision and better overall performance

CONCLUSIONS & FUTURE WORK

IntraCFG is a language-independent RAGs framework that overcomes the limitation of the earlier approaches:

- High-Precision
- ≥30% fewer nodes
- Concise CFG specification
- Competitive to SonarQube

We plan to:

- extend the support of IntraJ to Java 8
- extend **IntraCFG** to construct inter-procedural CFGs

LTH FACULTY OF ENGINEERING LUND UNIVERSITY

WASP WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

Rizwan, Momina
Lund University

WASP | WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

A Domain-Specific language to express dynamic functional safety rules

Ensuring functional safety has become more challenging as robots work in a dynamic and unpredictable environment. A functionally safe autonomous system performs correctly given a set of inputs and if it receives an unknown input, then it fails predictably. While making a robot safe, sometimes we over-constrain the system that makes the robot incapable of doing anything useful. For example, turning off the robot whenever something unexpected happens is not a good recovery strategy. Specifying static safety constraints is a conservative approach. We develop a domain-specific language (DSL) that facilitates the user to specify dynamic safety specifications. A DSL allows us to reason at an abstract level making it easier to handle abstract domain-specific concerns like functional safety. Domain-specific modeling also allows us to validate if a system behaves as expected.

Rizwan, Momina
Lund University

WASP WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

A ***static*** safety constraint hinders a robot to do anything ***useful***!

A Domain-Specific Language to express dynamic robot safety rules

Momina Rizwan, Christoph Reichenbach and Volker Krueger

Motivation & Research Goal

Ensuring functional safety has become more challenging as robots work in a dynamic and unpredictable environment.

"A functionally safe autonomous system performs correctly given a set of inputs and if it can't, it fails predictably."

Static safety rules over-constrain the system that makes the robot incapable of doing anything useful. Turning off the robot whenever something unexpected happens is not a good strategy. In our research, we try to replicate the work by [S.Adam et. al.] [1] and add dynamic safety rules.

Safety scenarios

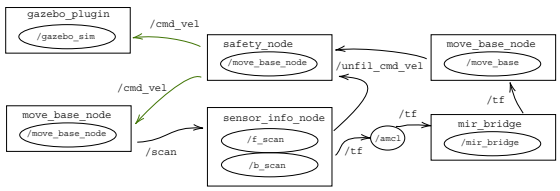
1. Avoid damaging jerks while crossing uneven terrain (IMU)
2. Handle ramps in an industrial setting (IMU)
3. Handle partially closed doors (LIDAR)
4. Robot arm should never hit anything (Force Sensor)

Dynamic recovery strategy follow the rule:

"As soon as the input sensor reading is in safe-range, continue the task."

- When the ramp slope is gentle, slow down, go back and find a new path
- When the ramp slope is steep, speed up a bit so it can cross the ramp

Safety Node As a Filter



Why use a Domain Specific Language?

- To reason about domain specific concepts that are easier to understand by non-experts [2].
- One can do domain specific static analysis and report errors early.
- Better error reporting.

A code snippet (Syntax is in the style of Ulrik's work [1])

```
action moveBack ;
action lowSpeed ;
action increaseSpeed ;
const maxSpeed = 0.5 m/s
const reasonabletilt = 10 deg
const maxtilt = 20 deg
input orientation = topic
imu_information
entity imuSensorSystem
{
  gentleSlope :
    orientation.pitch() not in reasonabletilt
    for 4.0 sec;
  steepRamp :
    orientation.pitch() not in maxtilt
}
entity driveSystem {
  maxspeedExceeded :
    linearSpeed > maxSpeed for 2.0 sec ;
}
if imuSensorsystem.gentleSlope and driveSystem.moving
then { increaseSpeed ;} ;
if imuSensorsystem.steepRamp and driveSystem.moving
then { lowSpeed ; moveBack; } ;
```

*

References

[1] S. Adam, M. Larsen, K. Jensen, and U. P. Schultz, "Rule-based dynamic safety monitoring for mobile robots," *Journal of Software Engineering for Robotics*, vol. 7, no. 1, pp. 121–141, 2016.

[2] A. Nordmann, N. Hochgeschwender, and S. Wrede, "A survey on domain-specific languages in robotics," in *International conference on simulation, modeling, and programming for autonomous robots*, pp. 195–206, Springer, 2014.

WASP WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

Spanghero, Marco
KTH

WASP | WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

Attacking and protecting GNSS receivers

Precise time and position obtained by GNSS receivers is an integral part of a wide gamut of strategic infrastructure. Demonstrations of attacks highlight the vulnerability of current civilian GNSS signals. Advanced countermeasures using external information and receiver properties can be used to detect advanced spoofer and recover from attacks. The poster describes both aspects, with a specific outlook on time focused receivers.

Spanghero, Marco

KTH

WASP WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

Attacking and protecting GNSS receivers

Marco Spanghero, KTH Royal Institute of Technology

Networked Systems Security (NSS) group, www.eecs.kth.se/nss

Main advisor: Panos Papadimitratos

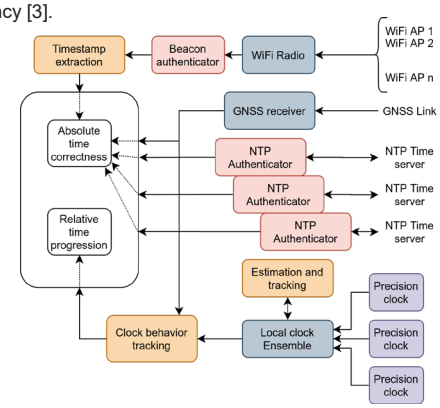
Motivation & Research Goals

Precise time and position obtained by Global Navigation Satellite System (GNSS) receivers are an integral part of a wide gamut of strategic infrastructure. Demonstrations of attacks highlight the vulnerability of current civilian GNSS signals. Advanced countermeasures using external information and receiver properties can be used to detect advanced spoofer and recover from attacks. Specifically, integration of different time sources can tackle various spoofing scenarios, from simple cases of simulation to advanced signal lift-off.

Time based attack detection

We seek a general-purpose framework that enables validation of the GNSS time information [1]. **Fusion of multiple (secure) network time sources** [2] provides online verification, using different off-the-shelf technologies.

Local high-precision clocks can be leveraged as an ensemble to guarantee enhanced holdover under attack even when connectivity is not available, reducing the remote timing service polling frequency [3].



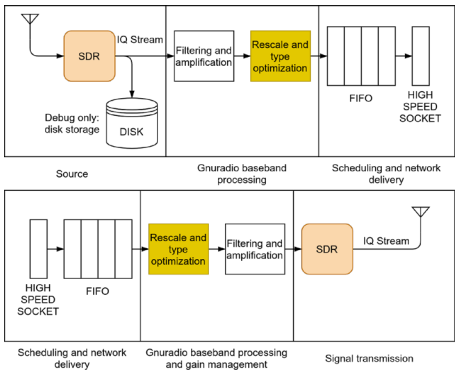
Combination of multiple time sources is not trivial and requires **knowledge of the source** characteristics: fusion can be possible with **stochastic filtering,** providing statistical indicators of GNSS time misbehavior (i.e. Kalman filtering).

Selected results:

- Detection based on opportunistic WiFi beacons with a 25µs threshold
- Multi-technology fusion framework for GNSS attack detection
- Local oscillator ensemble-based detection with 0.3µs threshold and dual frequency/phase indicator for advanced attack detection

GNSS Replay/Relay attacks

Replay of GNSS signals will be of increasing importance with the upcoming shift to authenticated signals [4]. Extending from simple meaconing to **over-the-network-meaconing** allows to **break free of the limitations** imposed by a physical connection.



Selected results:
Meaconing of mobile and static targets was successful over **consumer 4G networks**, by either replaying the entire spectrum (bandwidth intensive) or by **surgically replaying navigation messages** using meacon and recreate strategies (future proof against authenticated navigation messages).

We developed a **future-proof, flexible and versatile GNSS testing platform**. Our modular prototype will enable research on security enhanced signals, allowing mobile scenario testing without requiring any regulatory permission.

References

[1] K. Zhang, M. Spanghero, and P. Papadimitratos, “Protecting GNSS-based Services using Time Offset Validation,” in 2020 IEEE/ION Position, Location and Navigation Symposium (PLANS), Portland, Oregon, April 2020.

[2] M. Spanghero, and P. Papadimitratos, “Detecting GNSS misbehaviour with high-precision clocks. WiSec 2021”, in Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks, Virtual.

[3] M. Spanghero, K. Zhang, and P. Papadimitratos, “Authenticated time for detecting GNSS attacks,” in Proceedings of the 33rd International Technical Meeting of the Satellite Division of the Institute of Navigation, ION GNSS+ 2020, Virtual

[4] M. Lenhart, M. Spanghero, and P. Papadimitratos, “Relay/replay attacks on gnss signals,” in Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks, Virtual.

Tiwari, Deepika
KTH

WASP WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

Tests from Production Traces

Context: End users may interact with software in ways that are not well-tested.
Contribution: We propose to monitor software in production, in order to automatically improve the effectiveness of test suites. The generated tests can complement developer-written tests, and represent real usages of the application in production.

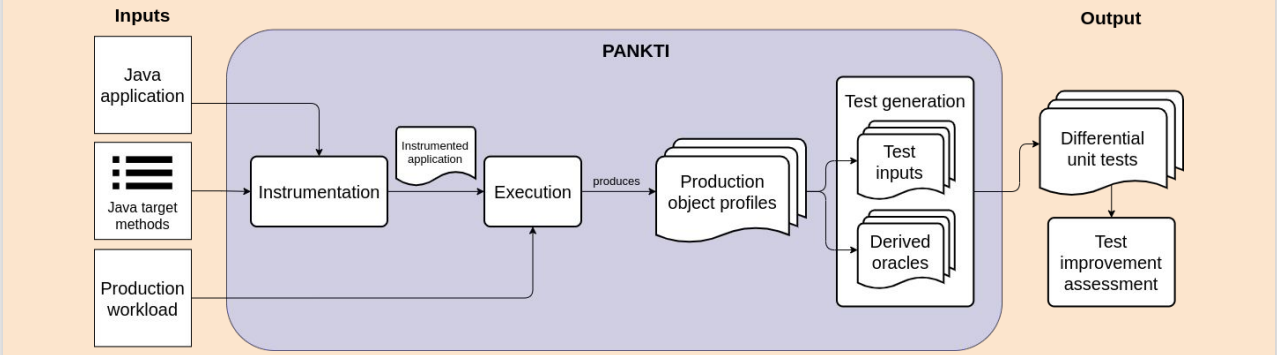
Tiwari, Deepika
KTH

WASP WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

Tests From Production Traces

Deepika Tiwari deepikat@kth.se
Division of Software and Computer Systems,
School of Electrical Engineering and Computer Science

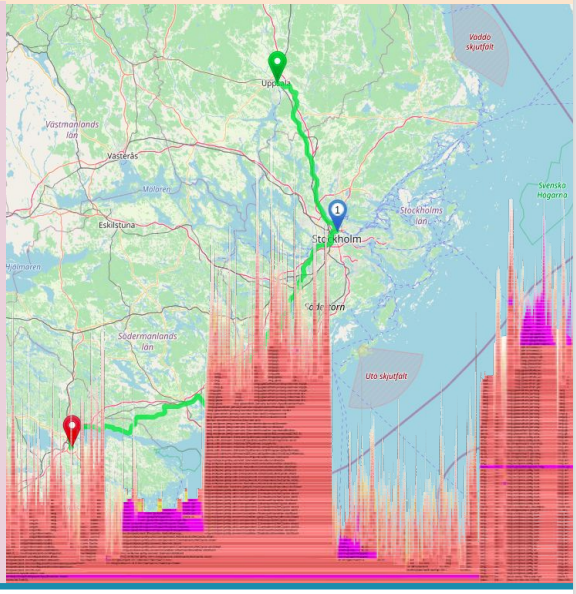
Context: End users may interact with software in ways that are not well-tested [1].
Contribution: We propose to monitor software in production, in order to automatically improve the effectiveness of test suites. The generated tests can complement developer-written tests [2], and represent real usages of the application in production [3].



PANKTI: Automated unit test generation [2]
- Generate tests that improve the test quality of 53 weakly-tested methods in 3 open-source Java applications

AutoGraphQL: Tests from GraphQL requests [3]
- Find 8 schema faults in 1 industrial application
- Improve coverage in 1 open-source application

Future work
- Monitor the invocation of methods within target methods in production [4]
- Use the collected data for the automated generation of mocks



References
1. Wang, Q., Brun, Y., & Orso, A. (2017, March). Behavioral execution comparison: Are tests representative of field behavior?. In 2017 IEEE International Conference on Software Testing, Verification and Validation (ICST). IEEE.
2. Tiwari, D., Zhang, L., Monperrus, M., & Baudry, B. (2021). Production Monitoring to Improve Test Suites. IEEE Transactions on Reliability.
3. Zetterlund, L., Tiwari, D., Monperrus, M., & Baudry, B. (2022, April). Harvesting Production GraphQL Queries to Detect Schema Faults. In 2022 IEEE International Conference on Software Testing, Verification and Validation (ICST). IEEE.
4. Zhang, L., Tiwari, D., Morin, B., Baudry, B., & Monperrus, M. (2021). Automatic Observability for Dockerized Java Applications. Submitted to IEEE Transactions on Dependable and Secure Computing.

WASP WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

Waldemarson, Gustaf
Lund University / Arm

WASP | WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

Efficient GPU Programming for Visual and Autonomous Software Systems

This project aims to improve the efficiency of parallel programming of GPUs in heterogeneous architectures and environments, potentially leading to new frameworks and algorithms for, e.g., more realistic lighting in graphics systems or higher performance computing in real-time software systems. The aim is to enhance or produce programming tools that can be used to process large datasets from e.g. computer vision, deep learning, data visualization, or other graphics rendering or systems by using the potential available in modern platforms that contain GPUs and other accelerators.

Waldemarson, Gustaf
Lund University / Arm

WASP WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

Efficient GPU Programming for Visual and Autonomous Software Systems

Gustaf Waldemarson Ind. PhD, Arm Ltd and Lund University
Dept. of Computer Science, Lund University Graphics Group
Supervisors: Michael Doggett (LU) and Simone Pellegrini (Arm Ltd)

LUND UNIVERSITY

arm

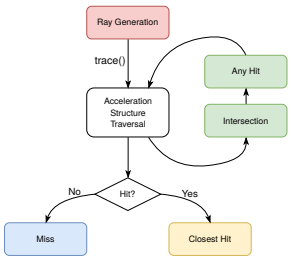
Motivation & Research Goals

This project aims to improve the efficiency of parallel programming of GPUs in heterogeneous architectures and environments, potentially leading to new frameworks and algorithms for, e.g., more realistic lighting in graphics systems or higher performance computing in real-time software systems. The aim is to enhance or produce programming tools that can be used to process large datasets from e.g. computer vision, deep learning, data visualization, or other graphics rendering or systems by using the potential available in modern platforms that contain GPUs and other accelerators.

Methods

When devising high-performance algorithms, it is often beneficial to decouple the algorithm itself from scheduling and performance aspects. As an example, the Halide [3] framework (see below) has successfully done just this for the domain of image filters and as a part of this research we are investigating similar DSLs for generating 3D content, such as rasterization or ray tracing.

```
Func blur_3x3(Func input) {
  Func blur_x, blur_y;
  Var x, y, xi, yi;
  // The algorithm - no storage or order
  blur_x(x,y) = (input(x-1,y) + input(x,y) + input(x+1,y))/3;
  blur_y(x,y) = (blur_x(x,y-1) + blur_x(x,y) + blur_x(x,y+1))/3;
  // The schedule - defines order, locality; implies storage
  blur_y.tile(x, y, xi, yi, 256, 32)
        .vectorize(xi, 8).parallel(y);
  blur_x.compute_at(blur_y, x).vectorize(x, 8);
  return blur_y;
}
```

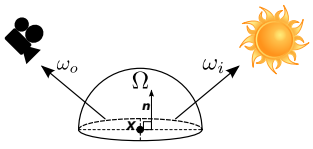


As seen above, one of the core components in ray tracing is the acceleration structure. It is the single greatest thing that determines how well ray tracing algorithms perform [2]. Thus, a lot of research has been devoted to understanding how to make this better. As a part of our research, we are investigating how to specialize this structure for particular use cases.

Physically Based Rendering

Nature is a complicated beauty with many areas that we still do not fully understand. Even when we limit ourselves to only light transport, there are many things that we are not modeling yet. Thus, another part of this research is to extend the toolkit of physical phenomena we can simulate using ray-tracing or rasterization based methods.

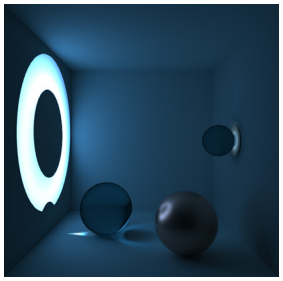
$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_\Omega f_r(\mathbf{x}, \omega_i, \omega_o) L_i(\mathbf{x}, \omega_i)(\omega_i \cdot \mathbf{n})\, \mathrm{d}\omega_i$$

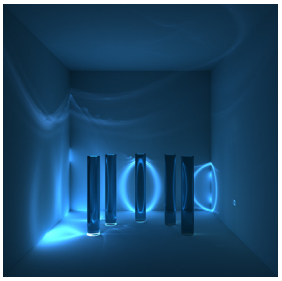


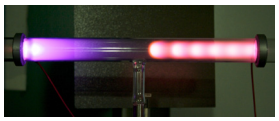
The Rendering Equation, the basis for almost all graphics algorithms.

Selected Results

One of the physical phenomena that is modeled as a part of this project is the transient light known as Cherenkov radiation. A few examples of this phenomenon can be seen below and for more details please see our work in **1**.







Recent research has been focused on creating efficient rendering algorithms for the so called *glow discharge* phenomenon, seen here.

References

[1] G. Waldemarson and M. Doggett, "Photon Mapping Superluminal Particles," in *Eurographics 2020 - Short Papers*, A. Wilkie and F. Banterle, Eds., The Eurographics Association, 2020, ISBN: 978-3-03868-101-4. DOI: 10.2312/egs.20201004

[2] D. Meister, S. Ogaki, C. Benthin, *et al.*, "A Survey on Bounding Volume Hierarchies for Ray Tracing," *Computer Graphics Forum*, 2021, ISSN: 1467-8659. DOI: 10.1111/cgf.142662

[3] J. Ragan-Kelley, A. Adams, D. Sharlet, *et al.*, "Halide: Decoupling algorithms from schedules for high-performance image processing," *Commun. ACM*, vol. 61, no. 1, pp. 106–115, Dec. 2017, ISSN: 0001-0782. DOI: 10.1145/3150211. [Online]. Available: https://doi.org/10.1145/3150211

Zhang, Long
KTH

WASP WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

Application-level Chaos Engineering

Chaos engineering is a new scientific method within software engineering that consists in specifying and evaluating resilience hypotheses by 1) injecting faults in a production system, 2) observing the impact of such faults, and 3) building new knowledge about the strengths and weaknesses of the resilience of the system. Chaos engineering can be applied at different levels such as network level and infrastructure level. In order to provide more concrete and application-specific insights for developers, our research work focuses on using application-level chaos engineering to address the following challenges: C1-How to evaluate different aspects of resilience, C2-How to automate the chaos experiments, and C3-How to improve the efficiency of the chaos engineering experiments.

Zhang, Long
KTH

WASP WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

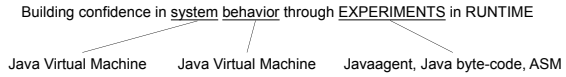
Application-level Chaos Engineering

Long Zhang <longz@kth.se>, KTH
Main advisor: Martin Monperrus

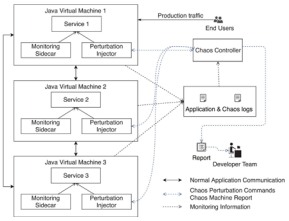
Abstract

Chaos engineering is a new scientific method within software engineering that consists in specifying and evaluating resilience hypotheses by 1) injecting faults in a production system, 2) observing the impact of such faults, and 3) building new knowledge about the strengths and weaknesses of the resilience of the system. Chaos engineering can be applied at different levels such as network level and infrastructure level. In order to provide more concrete and application-specific insights for developers, our research work focuses on using application-level chaos engineering to address the following challenges: **C1-How to evaluate different aspects of resilience, C2-How to automate the chaos experiments, and C3-How to improve the efficiency of the chaos engineering experiments.**

ChaosMachine



In order to address C1, we propose to use different perturbation models at the application level [1,2]. This is because: application-level perturbation models are closer to the application's source code, which helps developers to locate the improvement target, and such models simulate more concrete failure scenarios for this specific application.



- Input
  - Arbitrary software in Java
  - Hypotheses
- Architecture
  - Monitoring sidecars
  - Perturbation injectors
  - Chaos controller
- Output
  - Resilience report

Perturbation model: try-catch block short-circuit testing

- A corresponding exception at the beginning
- The whole try block is made invalid

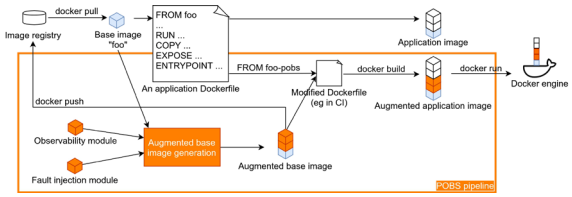
Hypotheses

- RH (Resilience hypothesis)
- OH (Observability hypothesis)
- DH (Debug hypothesis)
- SH (Silence hypothesis)

Evaluation

- 3 large-scale and well-known Java applications totaling 630k lines of code

POBS

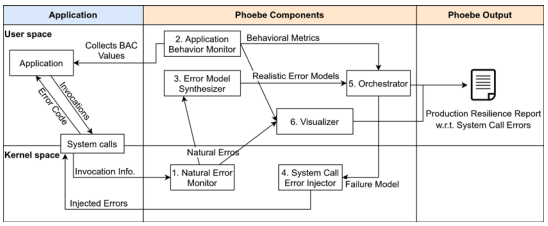


In order to address C2, we propose to design orchestration frameworks to connect monitoring, injection, and analysis. For most of the existing chaos engineering tools, there are several limitations: 1) steady state and hypotheses have to be manually defined, 2) the installation of a tool may be complicated, and 3) the setup of an experiment may be tedious.

Thus we propose a technique called POBS (imProved OBServability) to statically analyze and transform Docker configuration files of Java applications in order to inject observability capabilities [3].

For example, POBS allows developers to observe the JVM memory or CPU usage of their application **with minimal effort: a single line change in the Docker configuration**.

Phoebe



In order to address C3, we present a novel fault injection framework for system call invocation errors, called Phoebe [4]. Phoebe is unique as follows. First, Phoebe enables developers to have full observability of system call invocations. Second, **Phoebe generates error models that are realistic in the sense that they mimic errors that naturally happen in production.** Third, Phoebe is able to automatically conduct experiments to systematically assess the reliability of applications with respect to system call invocation errors in production.

References

1. Zhang et al, *A Chaos Engineering System for Live Analysis and Falsification of Exception-Handling in the JVM,* IEEE TSE, 2019.
2. Zhang et al, *TripleAgent: Monitoring, Perturbation and Failure-Obliviousness for Automated Resilience Improvement in Java Applications,* IEEE 30th International Symposium on Software Reliability Engineering (ISSRE), 2019.
3. Zhang et al, *Automatic Observability for Dockerized Java Applications,* arXiv preprint:1912.06914, 2019.
4. Zhang et al, *Maximizing Error Injection Realism for Chaos Engineering with System Calls,* IEEE TDSC, 2021.