EMPIRICAL EVALUATION OF FORWARD AND BACKWARD STATIC TAINT ANALYSIS

Master's Thesis

Amit Kumar

Paderborn | 29.06.2023

HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN

Fraunhofer
IEM

AGENDA

1. Motivation
2. Introduction
3. Research questions
4. Approaches and findings
5. Conclusion
6. Limitations and future work

Motivation

Taint analysis can detect many security vulnerabilities:

- 17 out of 25 SANS/CWE-25
- 6 out of the TOP 10 OWASP

Taint analysis needs to be fast and efficient

- Quick feedback to developers
- Restricted resources (developer machines)
- Optimal configuration for the analysis: self-adaption

Sources: Codebase-Adaptive Detection of Security-Relevant Methods: Piskachev et al.
Fluently specifying taint-flow queries with fluentTQL: Piskachev et al.

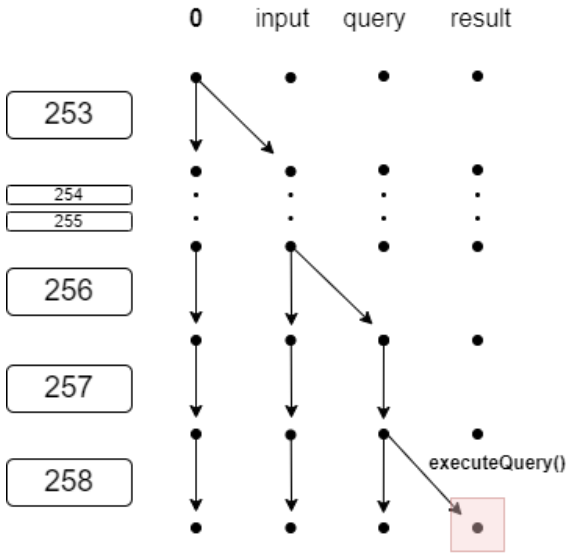
Introduction: Taint Analysis

Tracks the flow of sensitive or untrusted data (a taint)

from sources to sinks.

- a source can be user input, untrusted or sensitive data.
- a sink can be a security-sensitive method or a public channel.

```
250  .
251  .
252  .
253      public List<Message> searchMessage(String input) {   "abc;' DROP TABLE records;"
254          try{
255              Connection connection= DriverManager.getConnection(DB_URL, USER, PASS);
256              String query = "SELECT * FROM MESSAGE WHERE TEXT LIKE '".concat(input);   → Source
257              Statement statement = connection.createStatement();
258              ResultSet result = statement.executeQuery(query);   → Sink
259          } catch (SQLException e){
260  .
261  .
262  .
263  .
```



Exploded super graph

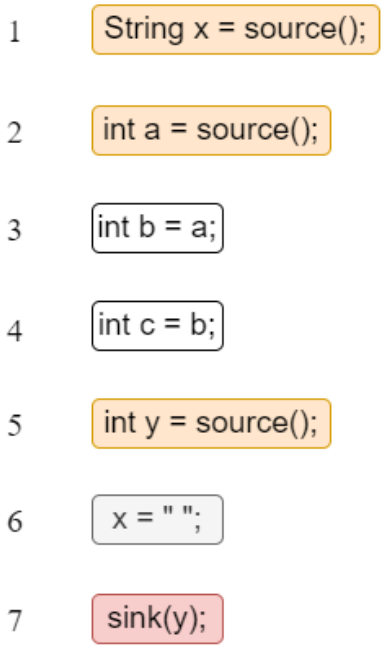
HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN

Fraunhofer IEM

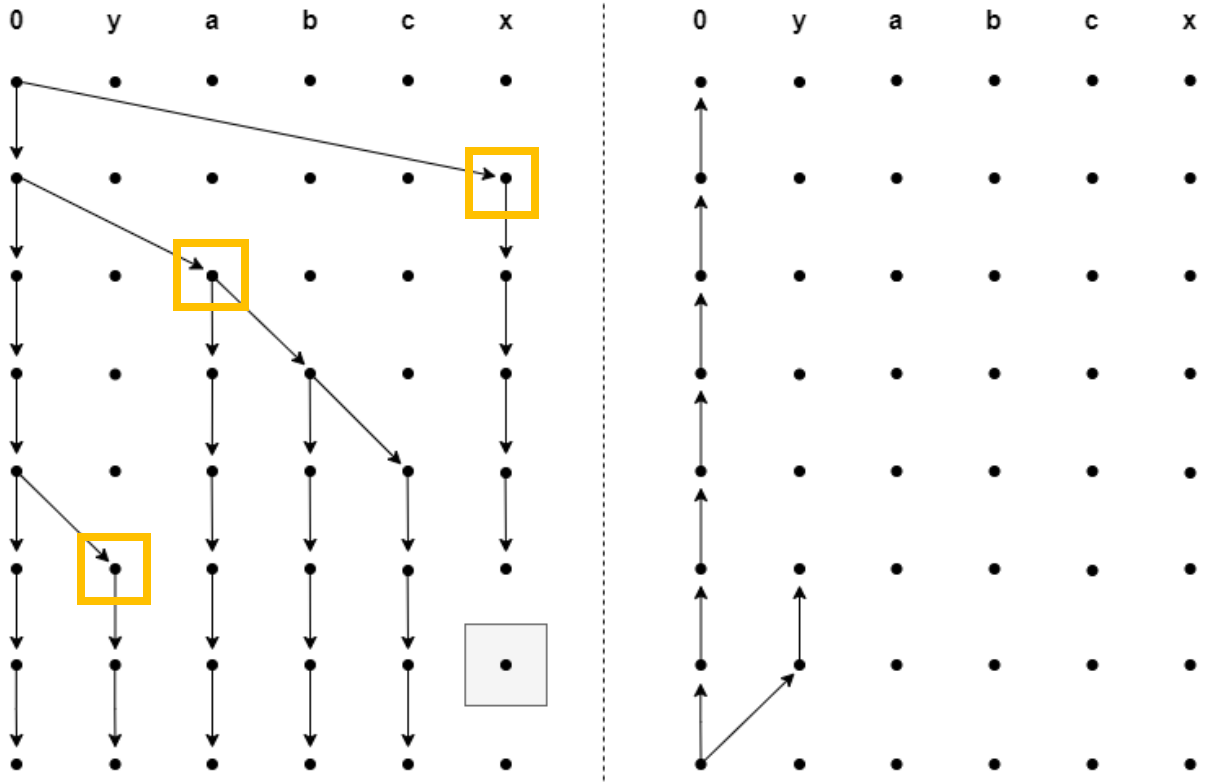
Taint Analysis Direction

Forward

Backward



Sample program

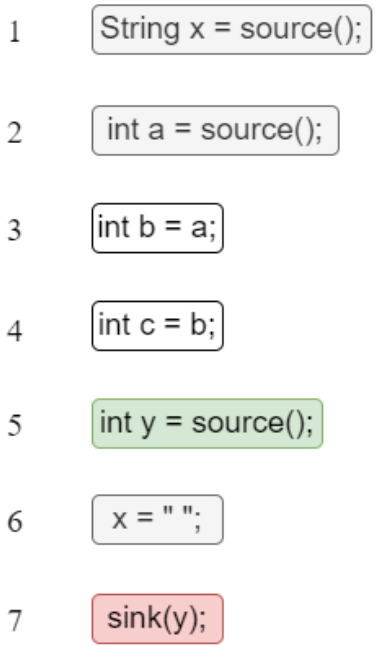


Exploded super graph

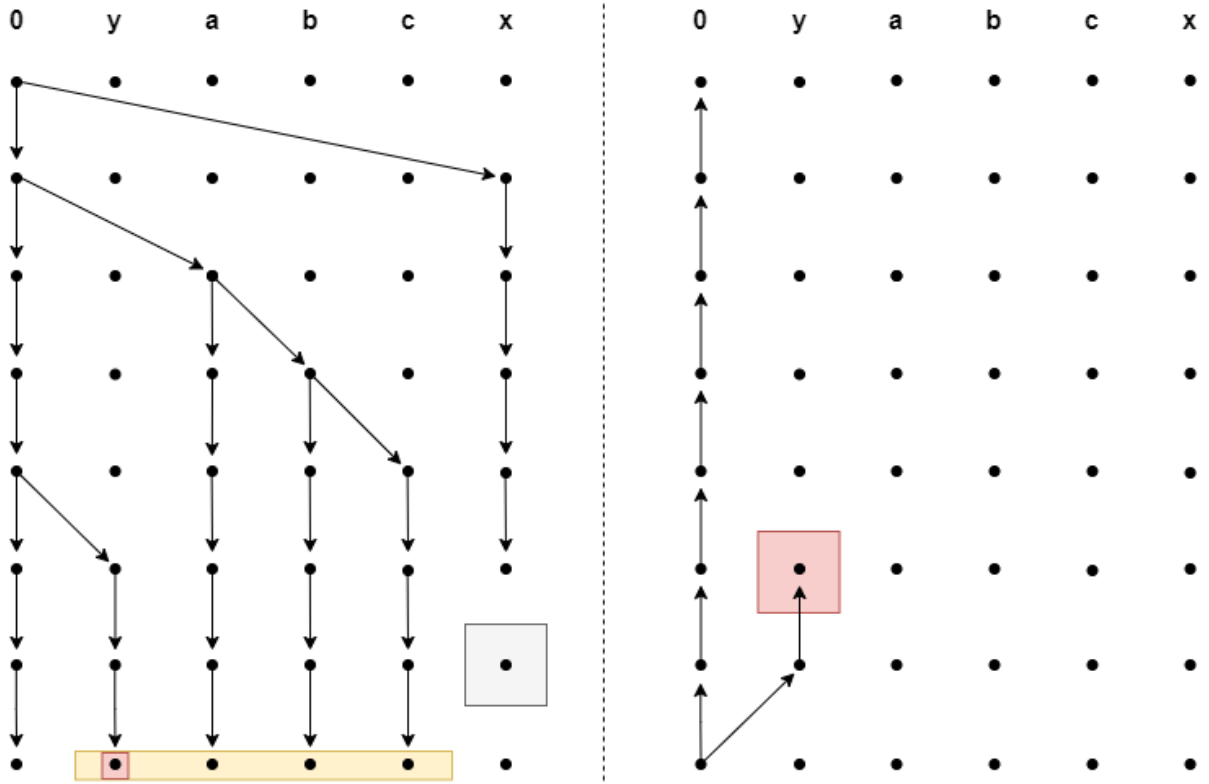
Taint Analysis Direction

Forward

Backward



Sample program



Exploded super graph

Research Questions

1. **How do the number of sources and sinks affect the efficiency of forward and backward analysis, and how do two types of taint analysis compare in terms of their overall efficiency?**

2. **How can we predict efficient taint analysis direction for a given application?**
   - does the number of sources and sinks play a role?
   - are there other source code properties that affect the analysis?

Solution Approach: RQ1

Evaluations requires applications

- varying number of sources and sinks
- different source code properties and taint flows

Created a benchmark suite: Taint Analysis Benchmark Suite (TABS)

- used benchmark case generator: GenBenchDroid
- 114 applications
- varying number of sources and sinks ratio in range 20:200 and 200:20
- 6 different taint flow patterns

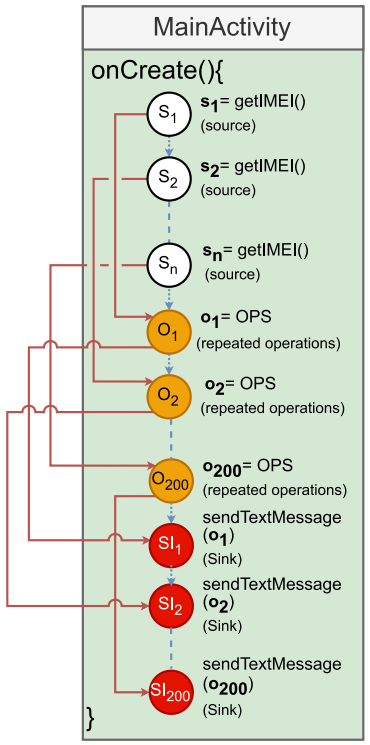
GenBenchDroid extension

Limitations addressed

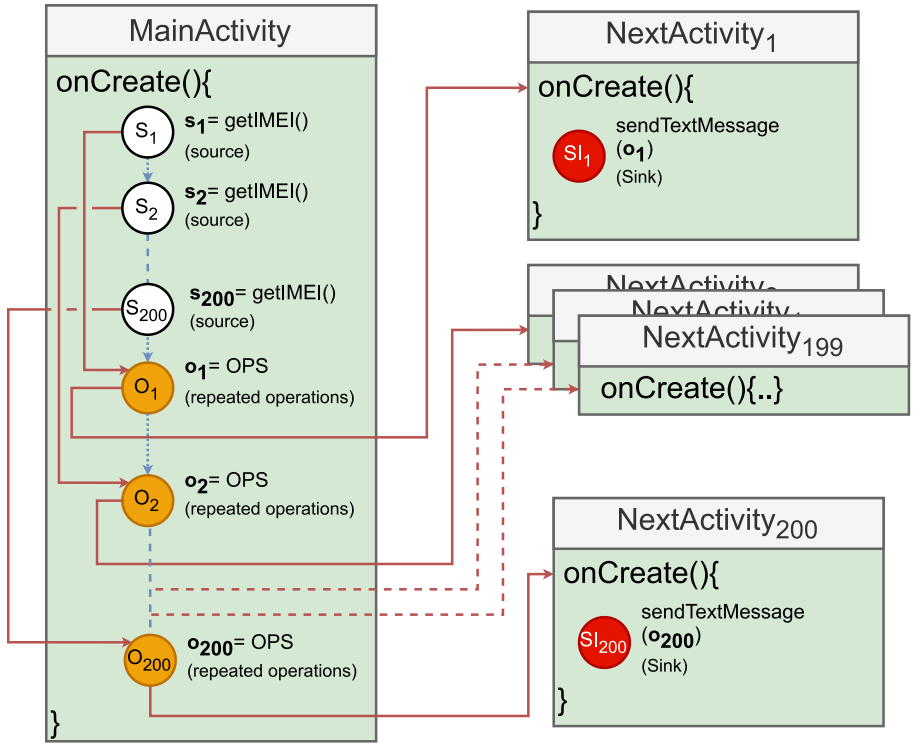
- Multi-variable taint flows
- Parallel taint flow and its ground truth
- Flexible application design

Source code pattern: series

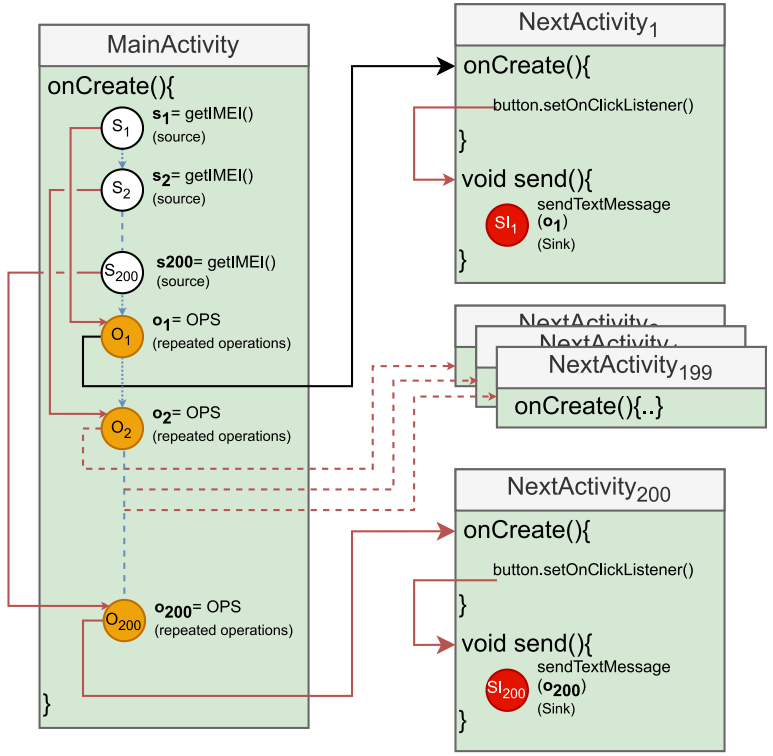
3 taint flow styles



simple



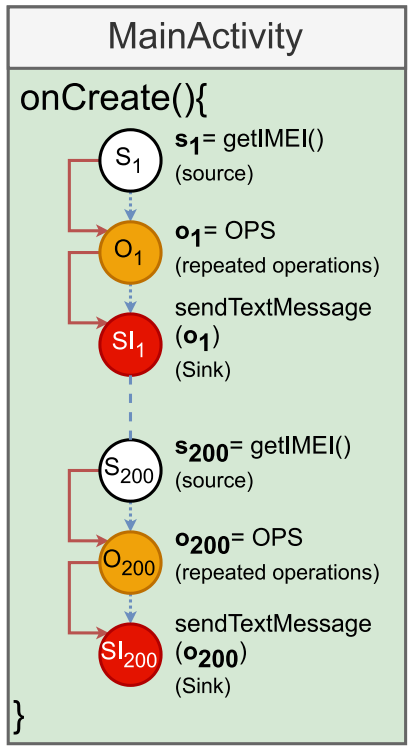
simple_icc



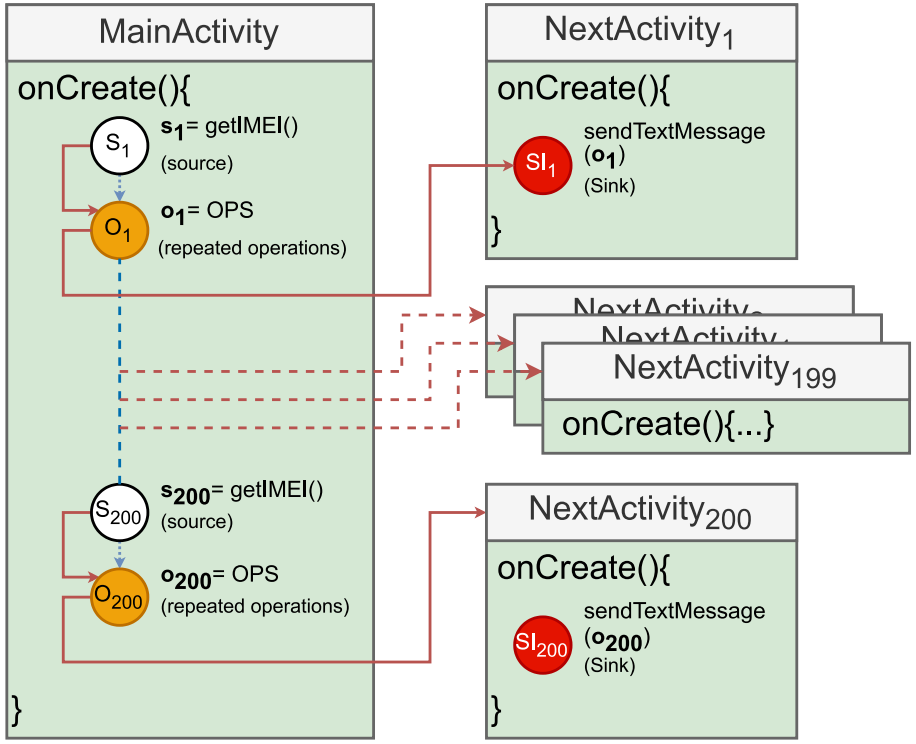
simple_icc_callback

Source code pattern: iteration

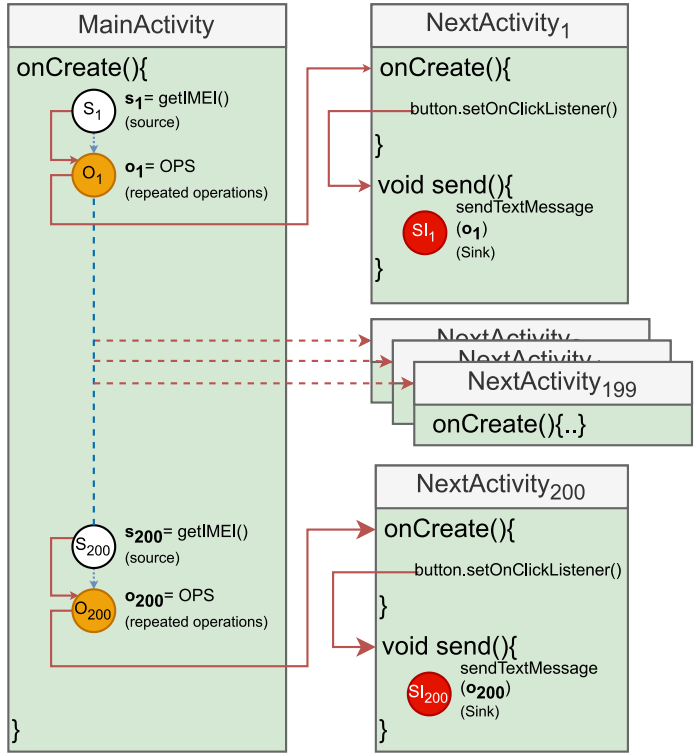
3 taint flow styles



simple



simple_icc



simple_icc_callback

FlowDroid

based on IFDS framework

Taint analysis in Android apps and Java programs

- can run forward and backward analysis independently
- a highly precise, efficient and maintained taint analysis tool
- can analyze large-scale Android applications

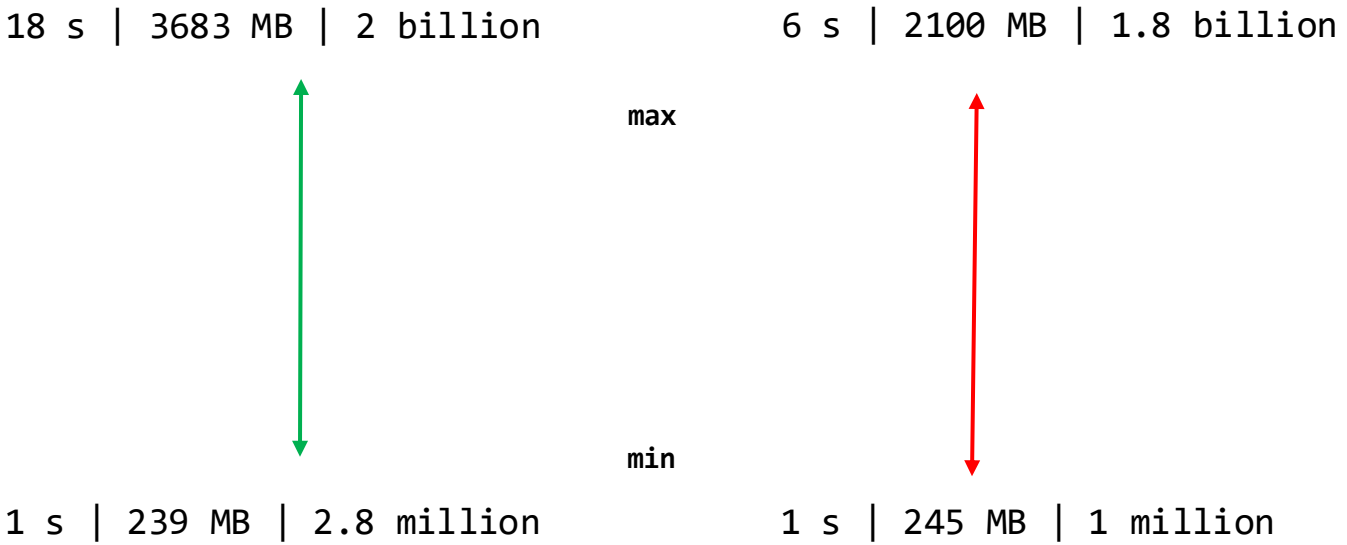
Evaluation Findings

Average results: 114 applications

Metrics	Forward Analysis	Backward Analysis	
Avg. data flow time	7.24 seconds	3.6 seconds	~50%
Avg. memory consumption	1189 MB	1025 MB	~86%
Avg. total edge propagations	1.9 million	0.68 million	~36%

Evaluation Findings

Min and max: 114 applications

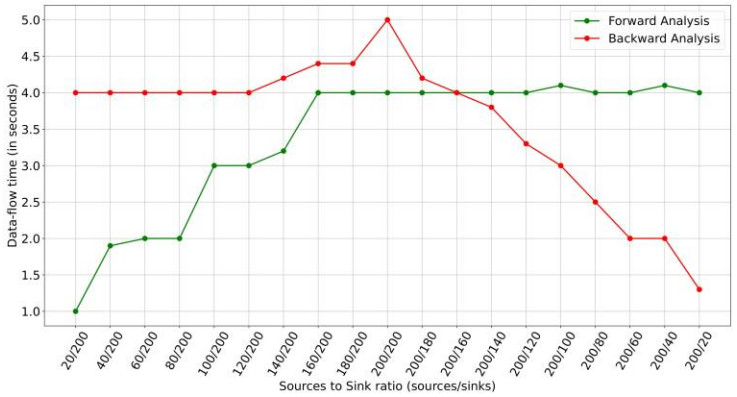


Evaluation Findings

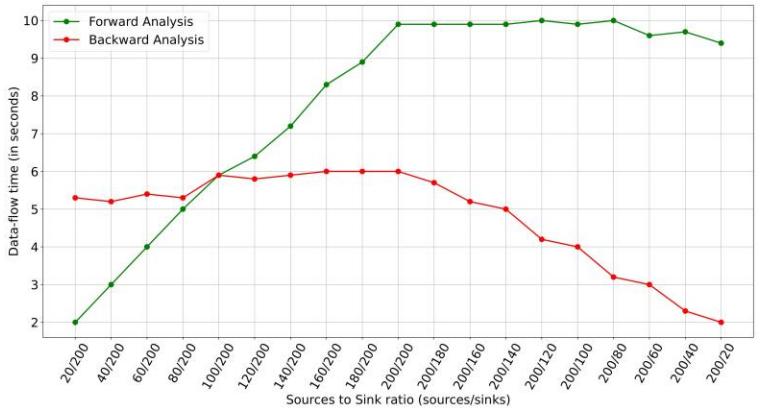
Data flow time

Forward Analysis

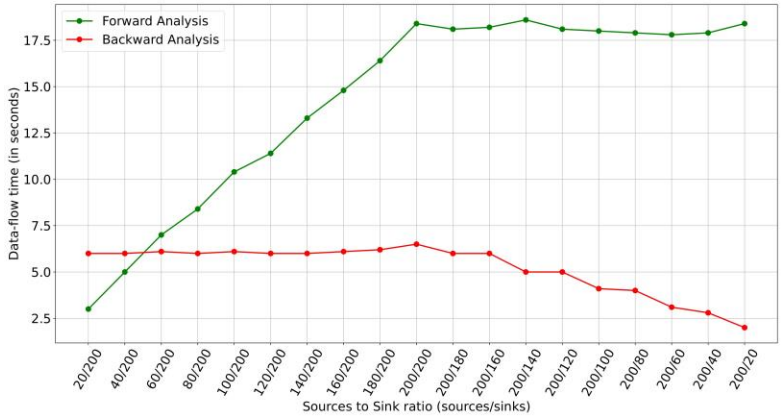
Backward Analysis



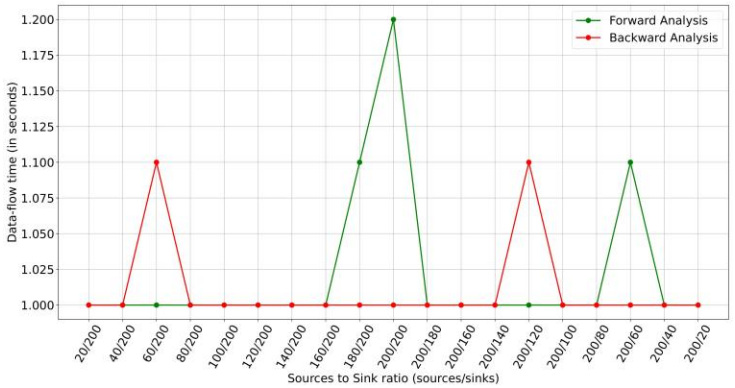
series_simple



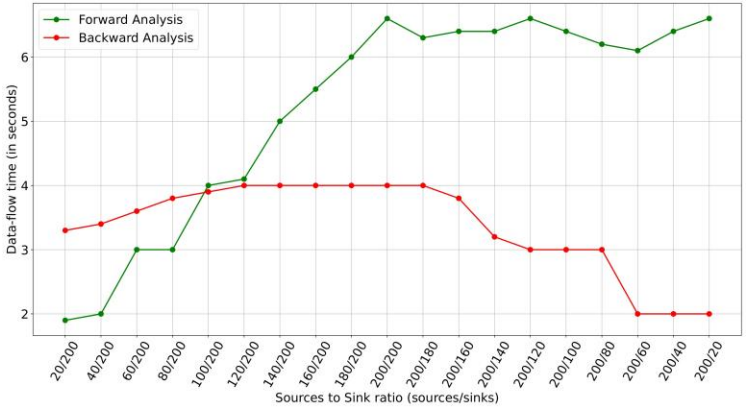
series_simple_icc



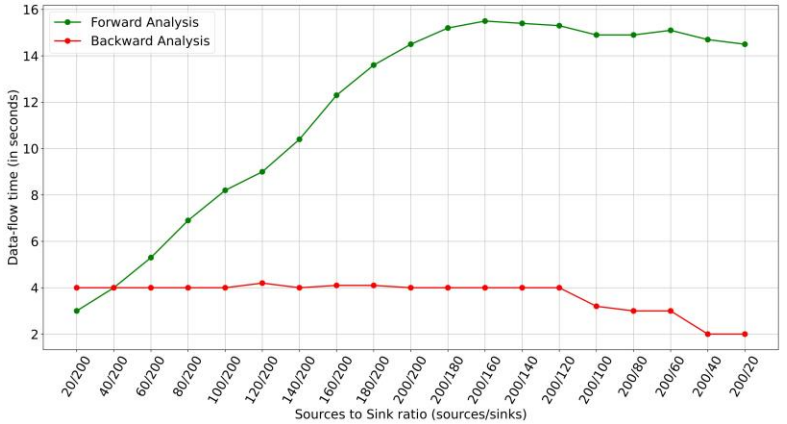
series_simple_icc_callback



iteration_simple



iteration_simple_icc



iteration_simple_icc_callback

HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN

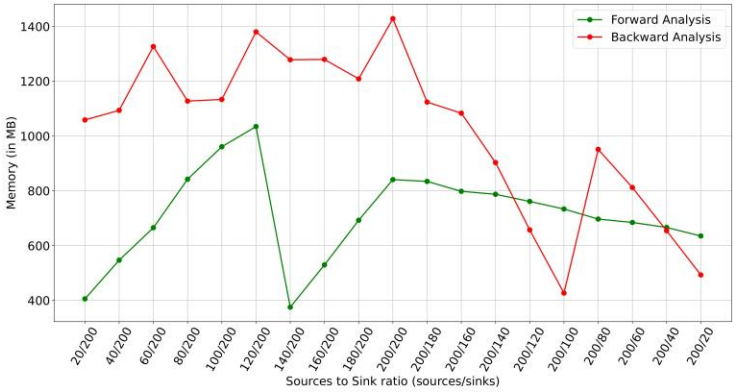
Fraunhofer IEM

Evaluation Findings

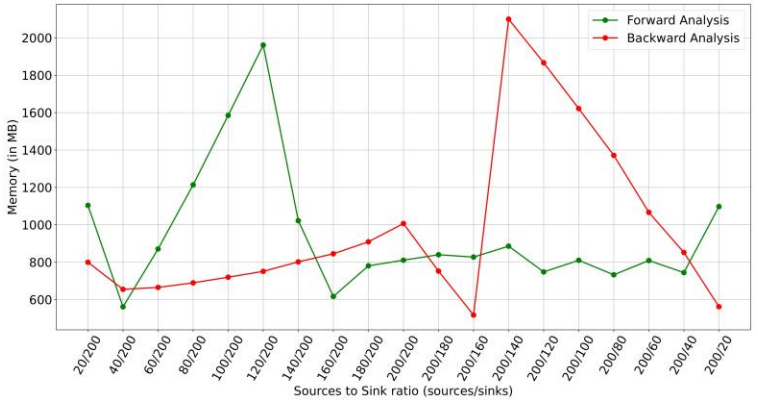
Memory consumption

Forward Analysis

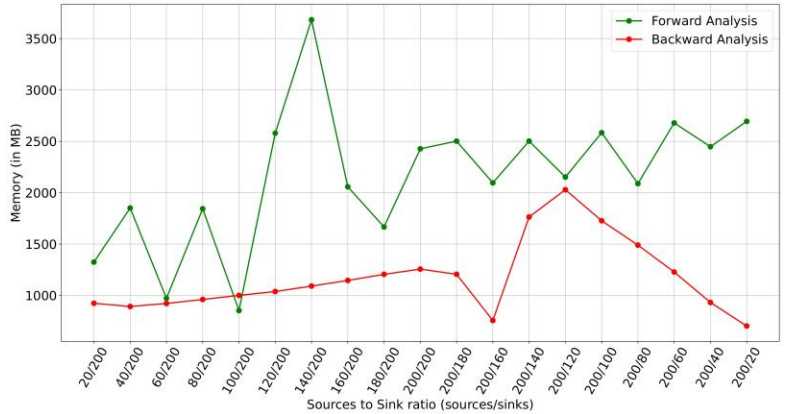
Backward Analysis



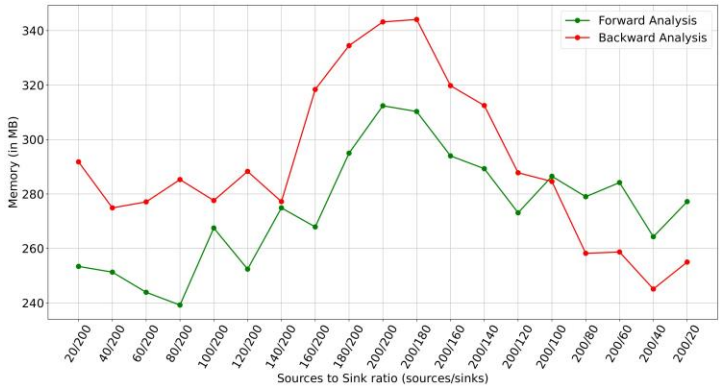
series_simple



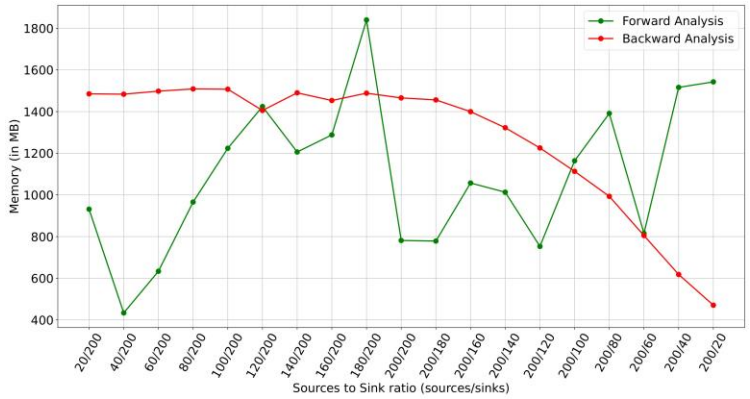
series_simple_icc



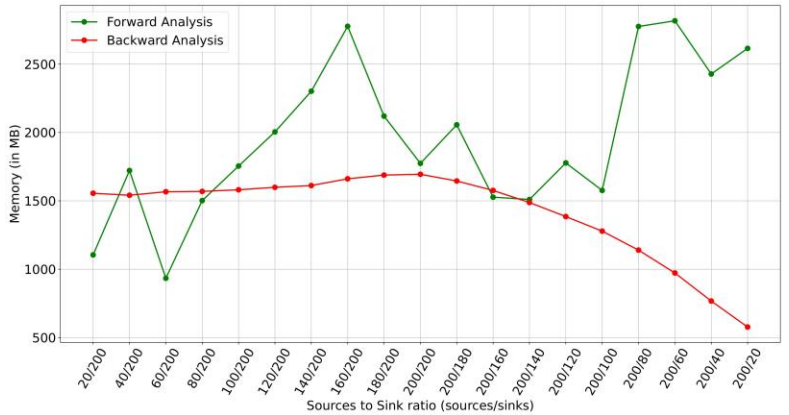
series_simple_icc_callback



iteration_simple



iteration_simple_icc



iteration_simple_icc_callback

HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN

Fraunhofer IEM

Solution Approach: RQ2

Resource consumption comparison respective to application metrics

- present number of sources and sinks 
- number of lines of code
- number of assignment operations
- number of methods

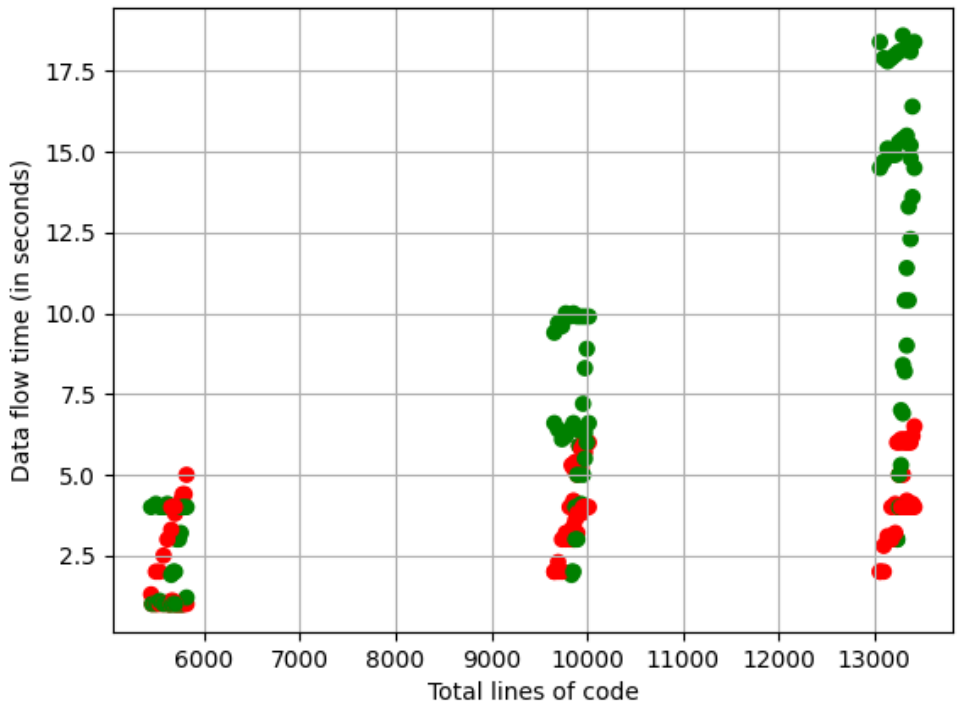
Solution Approach: RQ2

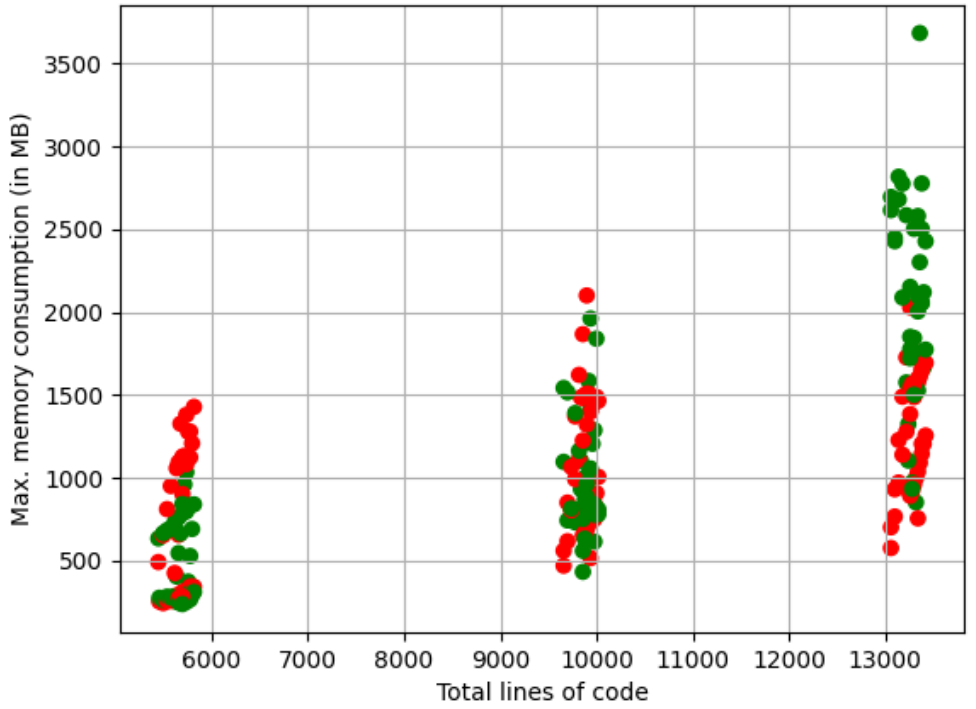
Resource consumption comparison respective to application metrics

- number of lines of code

Forward Analysis

Backward Analysis





HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN

Fraunhofer IEM

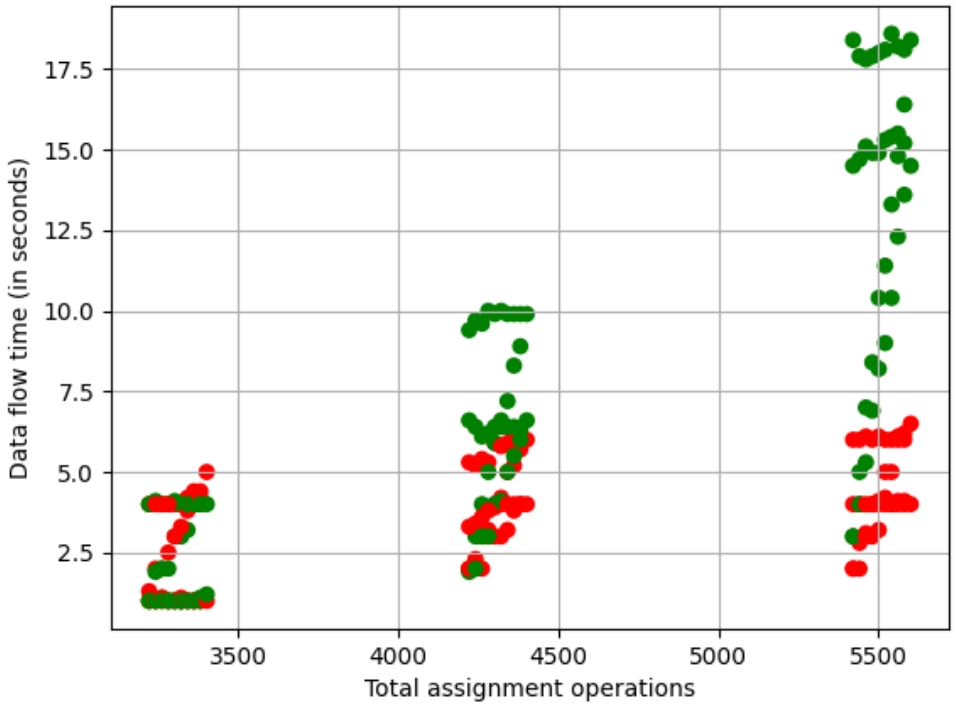
Solution Approach: RQ2

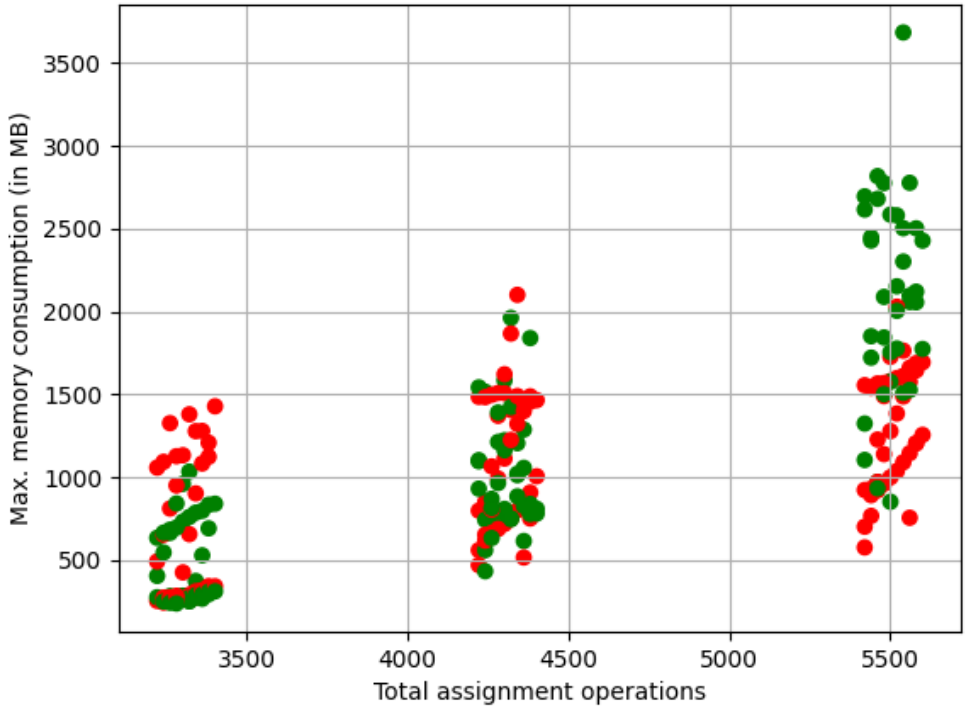
Resource consumption comparison respective to application metrics

- number of assignment operations

Forward Analysis

Backward Analysis





HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN

Fraunhofer IEM

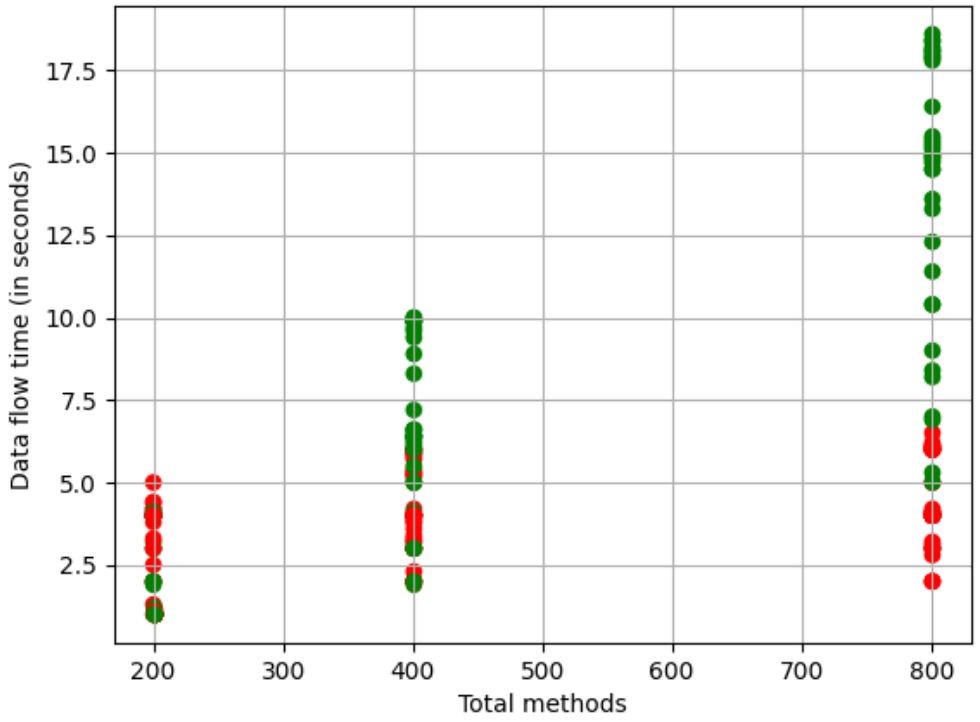
Solution Approach: RQ2

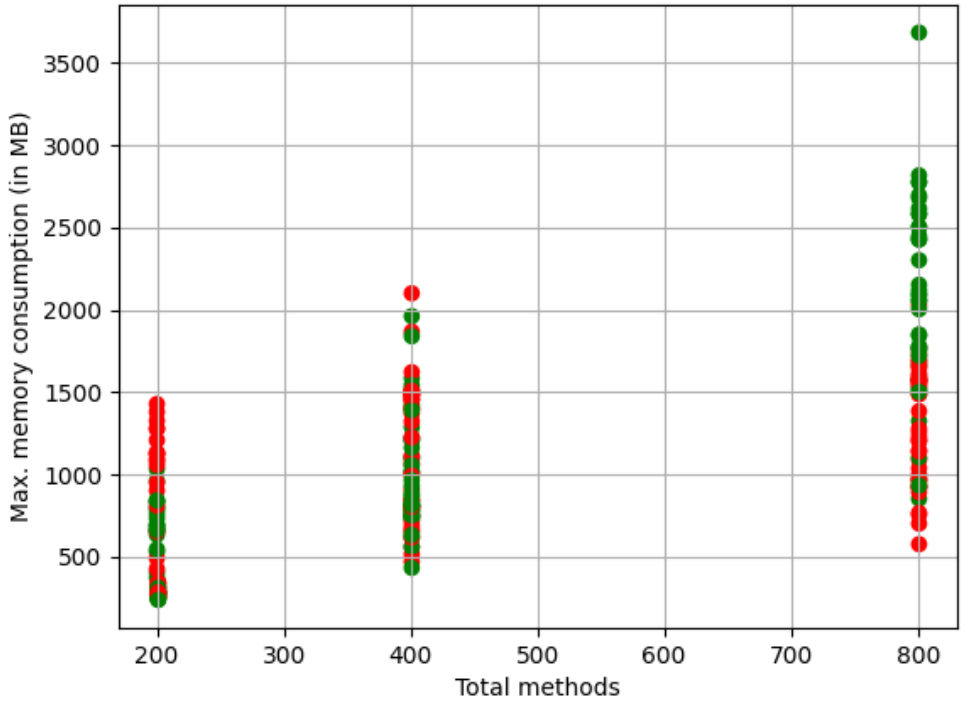
Resource consumption comparison respective to application metrics

- number of methods

Forward Analysis

Backward Analysis





HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN

Fraunhofer IEM

Conclusion

- Backward taint analysis is efficient (in many cases) in terms of data flow time while consuming comparable amount of memory compared to forward taint analysis

- Forward taint analysis is only efficient in cases where the number of sources is significantly smaller than sinks (sources<=1/3 sinks)

- Source code properties like LOC, assignment operations and methods, show no correlation to resource consumption or efficiency prediction

Limitations

- Generated benchmark applications in TABS do not account for all real-world application properties
- Empirical evaluation with a single analysis tool is not enough to validate the results

Future work

Benchmarking

- Further evaluations with real world applications
- Extension of TABS to include different taint flow patterns and application properties

Analysis tools

- Similar evaluations in other tools, for example: SecuCheck
- Extensions in SecuCheck to support Android application analysis

Many thanks
for your attention

Q & A

TABS

- ***src***: Java code and resources present in the built Android application
- ***target***: All compiled java files
- ***app-config.txt***: GenBenchDroid TMC configuration
- *.***build-log***: Build log from GenBenchDroid
- ***build.gradle***: Gradle build tool configuration
- ***full-ground-truth.xml*** and ***ground-truth.xml***
- ***generated-app.apk***: Android application.
- ***proguard-rules.pro***: Proguard rules used while building android applications.

Related Work: FlowTwist

based on IFDS

Taint analysis approach: Inside-out

uses the coordination of backward and forward taint analysis

- run on Java Class Library:
  - 45k public methods as **sources**
  - 134 call sites as **sinks**
- analysis completed in 10 minutes

Related Work: SecuCheck

Boomerang(SPDS) and FlowDroid(IFDS)

A configurable taint analysis tool

- can run on multiple IDEs
- analysis configuration possible with Java-DSL FluentTQL
- can adapt to detect different taint-style vulnerabilities

SecuCheck extension

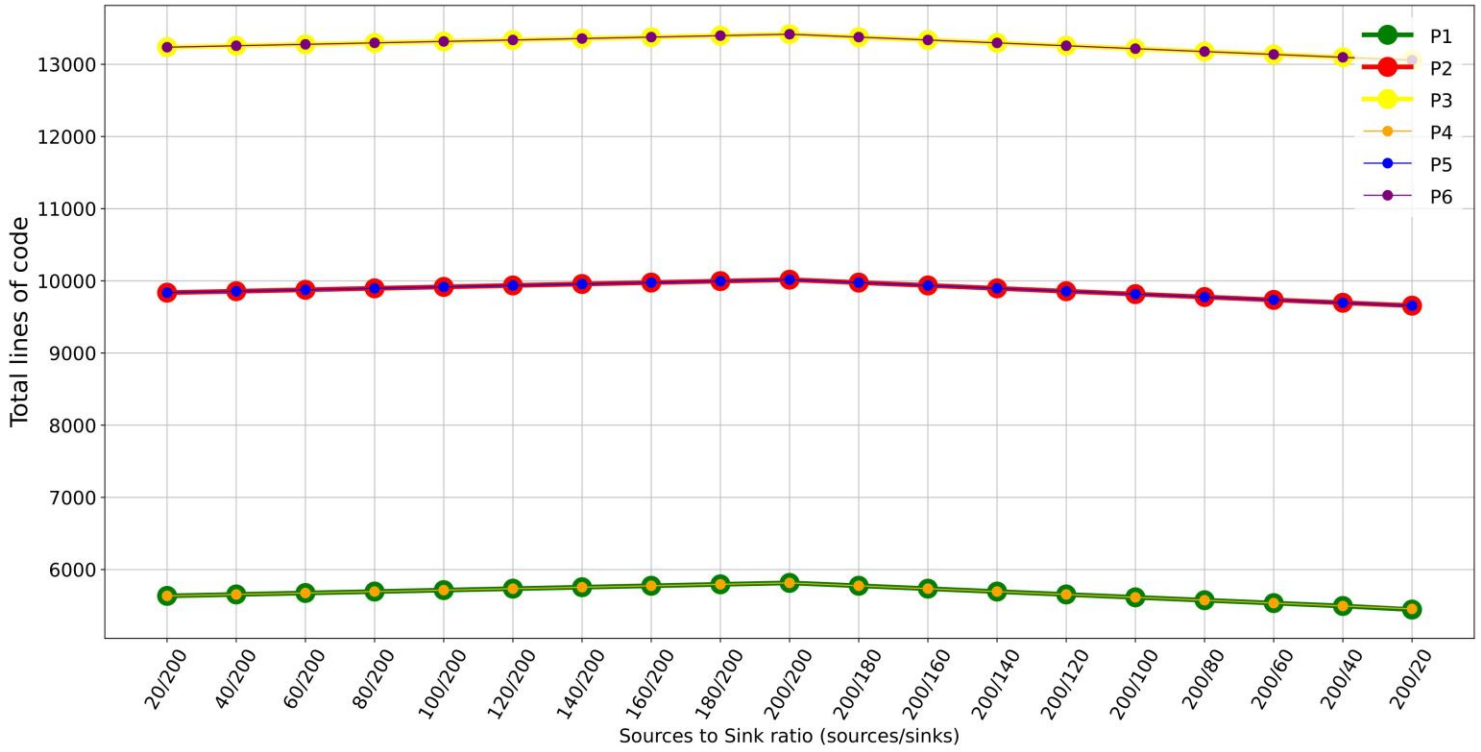
Boomerang(SPDS)

- direction setting instead of hard-code logic
- Tested against two micro-benchmark
- Android analysis test
- can detect vulnerabilities from class files though fails to report many
- For example: list clone, string builder

Solution Approach: RQ2

Resource consumption comparison respective to application metrics

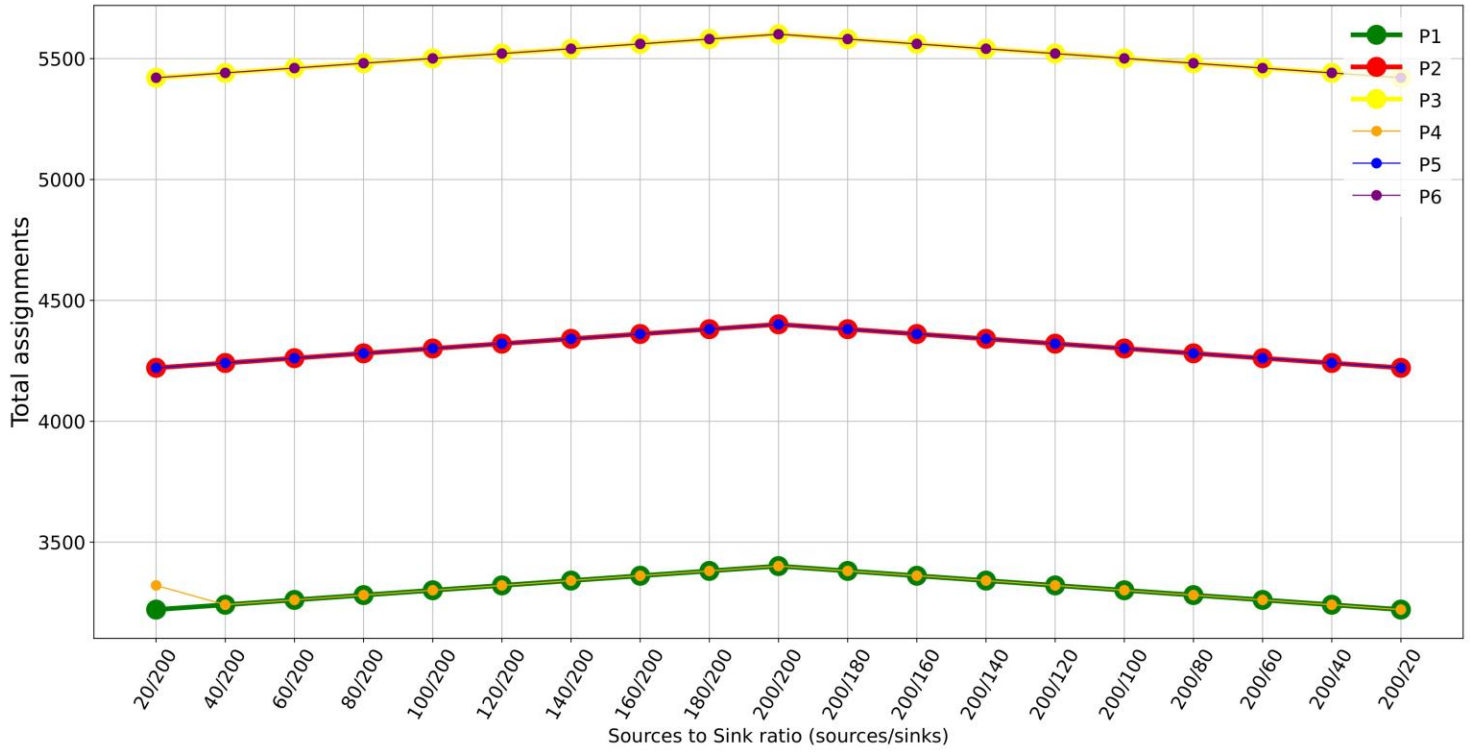
- number of lines of code



Solution Approach: RQ2

Resource consumption comparison respective to application metrics

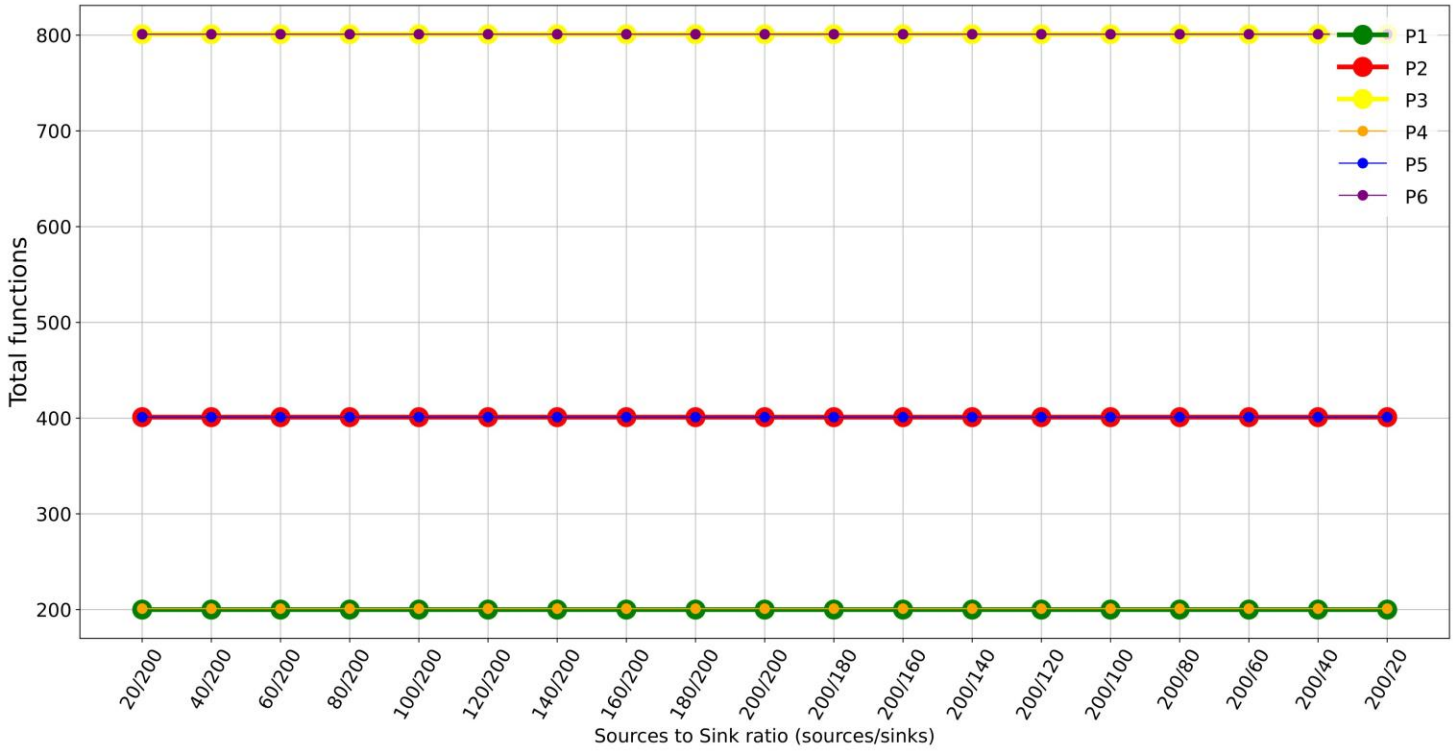
- number of assignment operations



Solution Approach: RQ2

Resource consumption comparison respective to application metrics

- number of methods



Evaluation Findings: Implementation of a Static Backwards Data Flow Analysis in FlowDroid by Tim Lange (2022)

Average results: 200 applications without timeout

Metrics	Forward Analysis	Backward Analysis	
Avg. data flow time	76.91 seconds	35.20 seconds	~46%
Avg. memory consumption	1535.20 MB	1473.21 MB	~96%
Avg. total edge propagations	7.5 million	6.1 million	~81%

Sources: Implementation of a Static Backwards Data Flow Analysis in FlowDroid by Tim Lange (2022)