Automatically Assessing and Extending Code Coverage for NPM Packages

Haiyang Sun[1], Andrea Rosà[1], Daniele Bonetta[2], Walter Binder[1]

[1] Università della Svizzera italiana (USI)
[2] Oracle Labs

Agenda

- Background
- Motivation and Challenges
- TESA
- Evaluation

- An open-source, cross-platform, back-end JavaScript runtime environment
- Enables both browser and server applications to be written in JavaScript


PayPal


LinkedIn


NETFLIX


Uber


node
JS

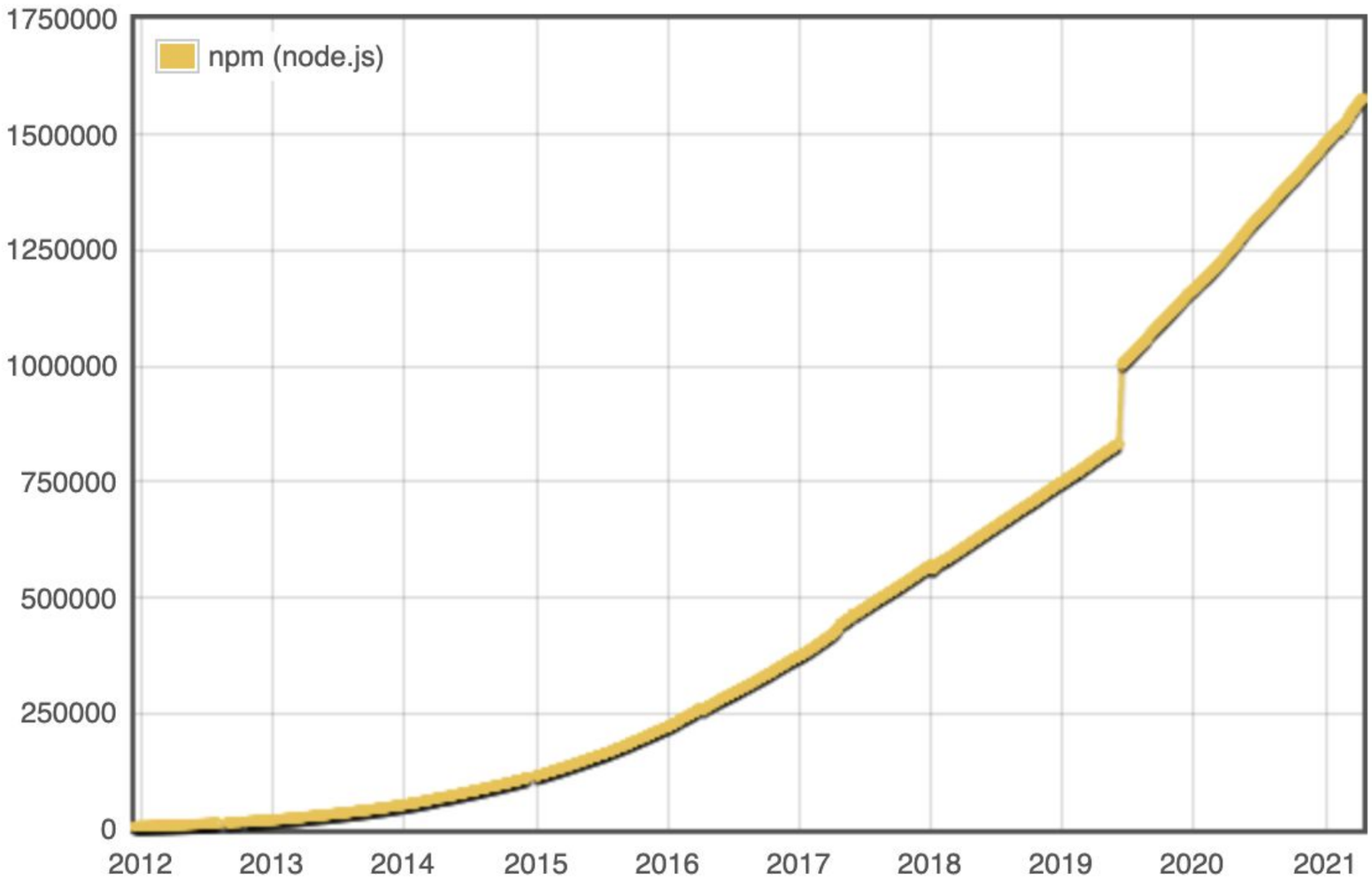

npm


ebay


GoDaddy


Medium

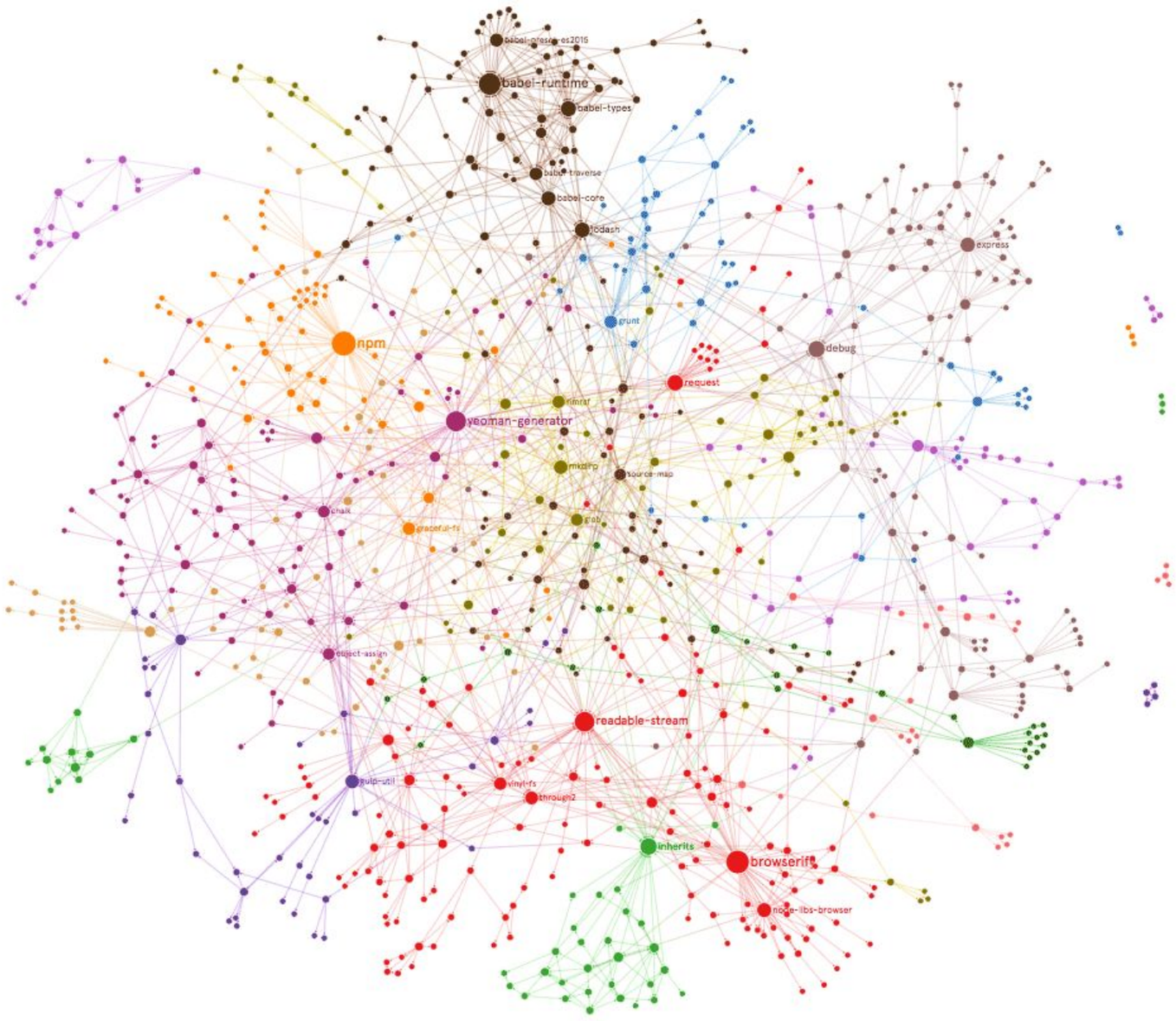
Node Package Manager (NPM)



Number of packages available in the past years

modulecounts.com

Node Package Manager (NPM)



Dependencies between packages

[medium.com - graph-commons]

Node Package Manager (NPM)

- Convenience and productivity for developers
- Applications vulnerable to package flaws
  - The *event-stream* incident stealing bitcoins (Sept. 2018)
    - 8 millions downloads in 3 months

https://blog.npmjs.org

Motivation

- Existing study [1]
  - Assessing code coverage of NPM packages is non-trivial
  - Many projects have no tests or tests with low code coverage
- Limitations
  - No automation - requires manual testing and coverage measurement
  - No proposal to extend coverage

[1] A. M. Fard and A. Mesbah, "JavaScript: The (Un)Covered Parts," ICST 2017.

TESA: TEST Automation for NPM

- Automate testing and code coverage measurement
- Identify tests from multiple sources:
  - Original tests from the NPM registry and the package's dev repository
  - Dependent tests from dependent packages
- Assemble test suites based on the code coverage
- An algorithm to compact the suite size
- Integrates with dynamic program analysis (DPA) tools

Limitation of Basic Testing

```
#!/bin/bash

$ npm install
$ npm test
$ exitcode=$?
```

- Not enough
  - Empty / no tests
  - Failed tests due to misconfiguration

Limitation of Code Coverage Measurement

```
#!/bin/bash

$ npm install
$ nyc npm test
```

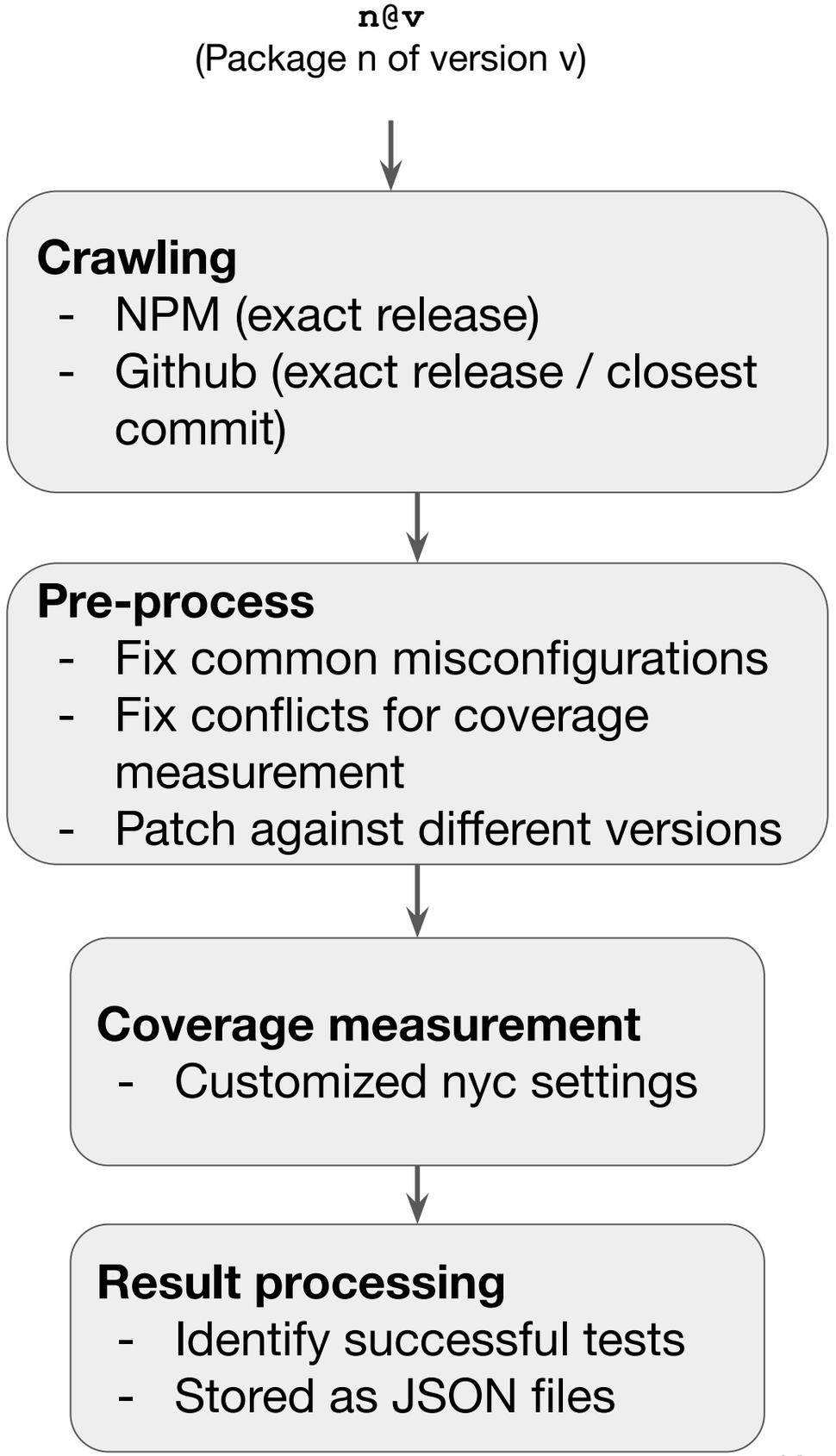
- Conflicting testing harness and linting tools
- Sensitive to package versions
- Cover only executed code by default

Test Automation with TESA

- Increase testing successful rate
  - Fix misconfigurations
- Extend usability of nyc
  - Patching conflicting testing harnesses
  - Disabling linting and other conflicting tools
- Accurate coverage results
  - Use checksum to verify code versions
  - Scan all available source code files

Test Automation with TESA

- Increase testing successful rate
  - Fix misconfigurations
- Extend usability of nyc
  - Patching conflicting testing harnesses
  - Disabling linting and other conflicting tools
- Accurate coverage results
  - Use checksum to verify code versions
  - Scan all available source code files



Multiple Sources of Tests

- Original tests
  - Apply the testing workflow on n@v (the target package)
- Dependent tests
  - Get all dependent packages that depend on n@v
  - Apply the testing workflow for each dependant package
    - Special configuration of nyc
    - Prioritizes the ones with higher daily downloads

Multiple Sources of Tests

- What to do if the test suite takes too long to finish?
- Compaction algorithm
  - Minimize time needed to execute the test suite
  - Maximize the code coverage
  - One-time cost

Integration with Dynamic Program Analysis

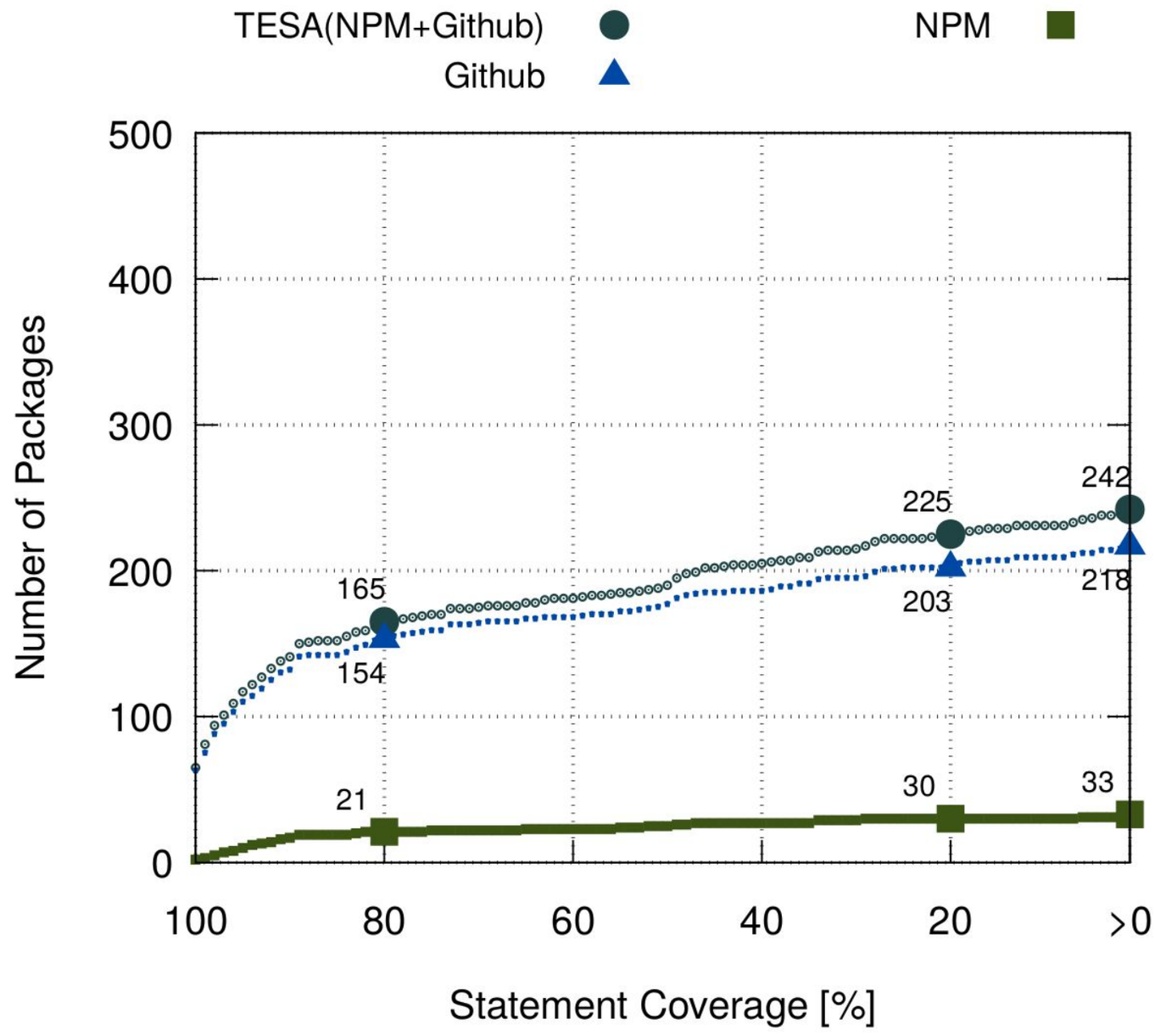
- TESA is integrated with NodeProf [1]
  - Easily port existing DPA tools which NodeProf supports and run them with the test suite found by TESA
  - DPA tools are more effective with better code coverage

[1] Haiyang Sun, Daniele Bonetta, Christian Humer, and Walter Binder. 2018. Efficient Dynamic Analysis for Node.js. CC 2018.

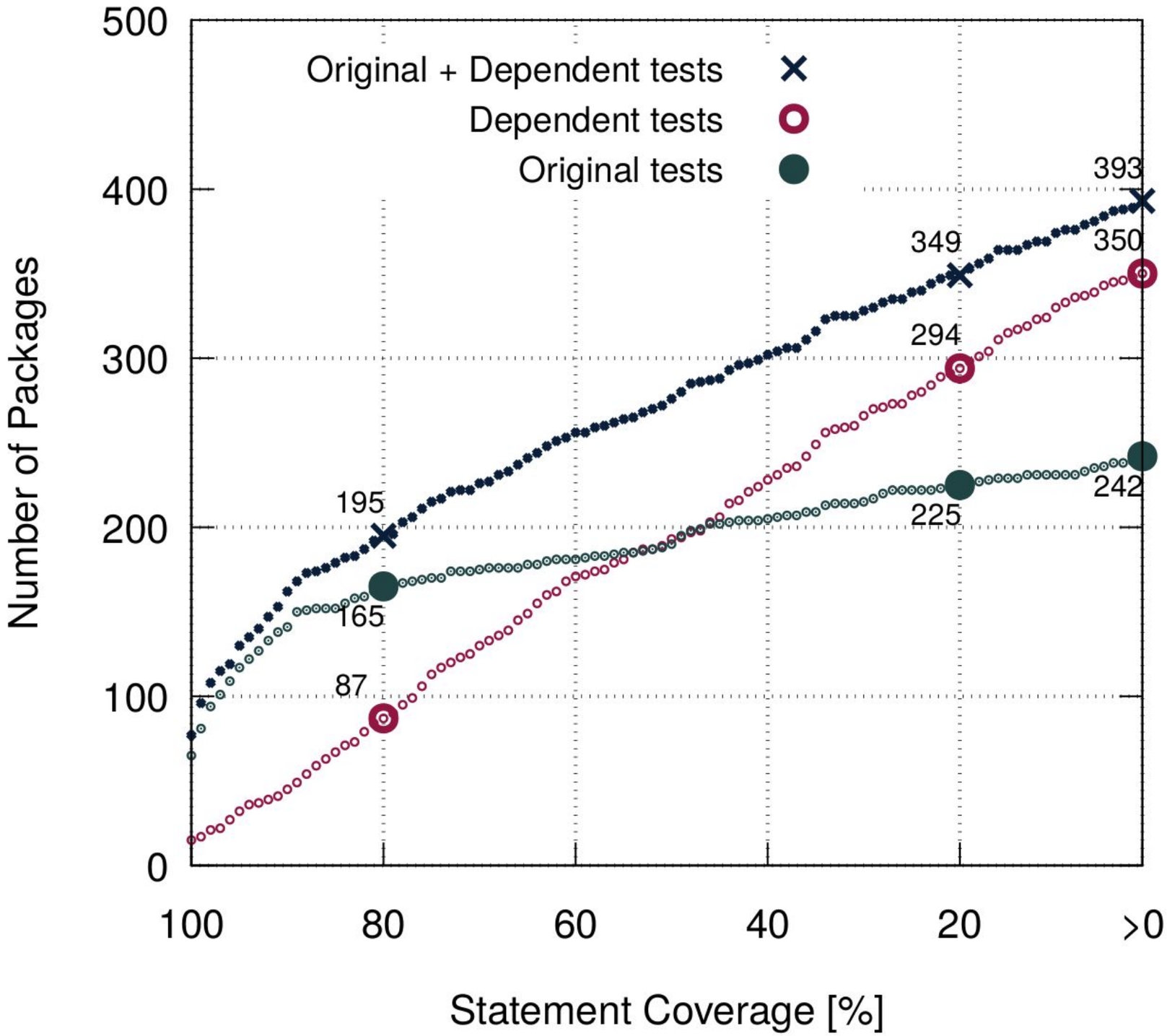
Evaluation

- Target packages: 500 popular NPM packages
  - Ranked by daily downloads
  - With at least 20 dependent packages
  - 20K+ packages tested as dependent packages

Cumulative Distribution of Coverage for Original Tests



Overall Code Coverage



Test Suite Compaction

- Compaction rate for 500 packages
  - Average time needed to finish reduces from 130 seconds to 22 seconds
  - Saved 80% time

DPA Tools from JITProf [1]

- DPA-1: non-contiguous array accesses
  - 2.5x more findings (from 8 to 20) with dependent tests
- DPA-2: typed arrays
  - 2.1x more findings (from 97 to 201) with dependent tests

[1] Liang Gong, Michael Pradel, and Koushik Sen. 2015. JITProf: Pinpointing JIT-unfriendly Javascript code. ESEC/FSE 2015.

Summary

TESA

- Test automation and coverage measurement
  - More scalable
- Original + dependent tests
  - More coverage
- DPA integration
  - More effective DPA tools

Q & A