Schnauzer: Scalable Profiling for Likely Security Bug Sites

William Arthur, Biruk Mammo, Ricardo Rodriguez, Todd Austin, Valeria Bertacco

MichiganEngineering

CGO, Shenzhen, China
February 26, 2013

Goal of this work

MAKE **SOFTWARE** MORE **SECURE**

Goal of this work

MAKE **SOFTWARE** MORE **SECURE**

Leveraging **Limited** Test Resources

Goal of this work

MAKE **SOFTWARE** MORE **SECURE**

Leveraging **Limited** Test Resources

Goal of this work

MAKE **SOFTWARE** MORE **SECURE**

Leveraging **Limited** Test Resources

Importance of Path

- Vast majority of security attacks are enabled by software bugs

Importance of Path

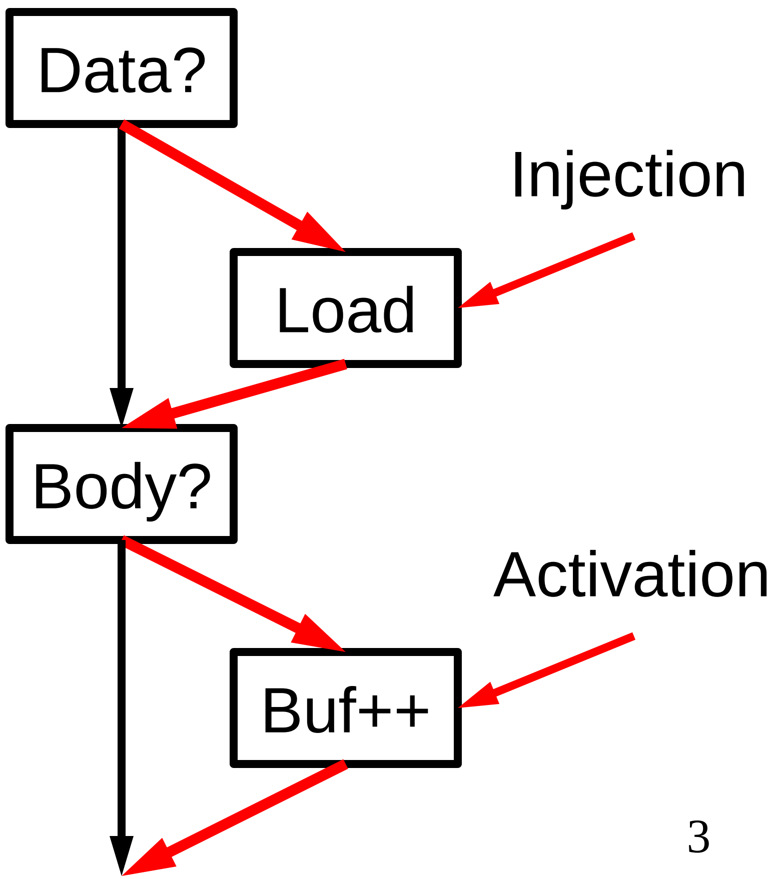
- Vast majority of security attacks are enabled by software bugs

  Often, hidden bugs only appear when sensitized by the proper path

Importance of Path

- Vast majority of security attacks are enabled by software bugs

Often, hidden bugs only appear when sensitized by the proper path



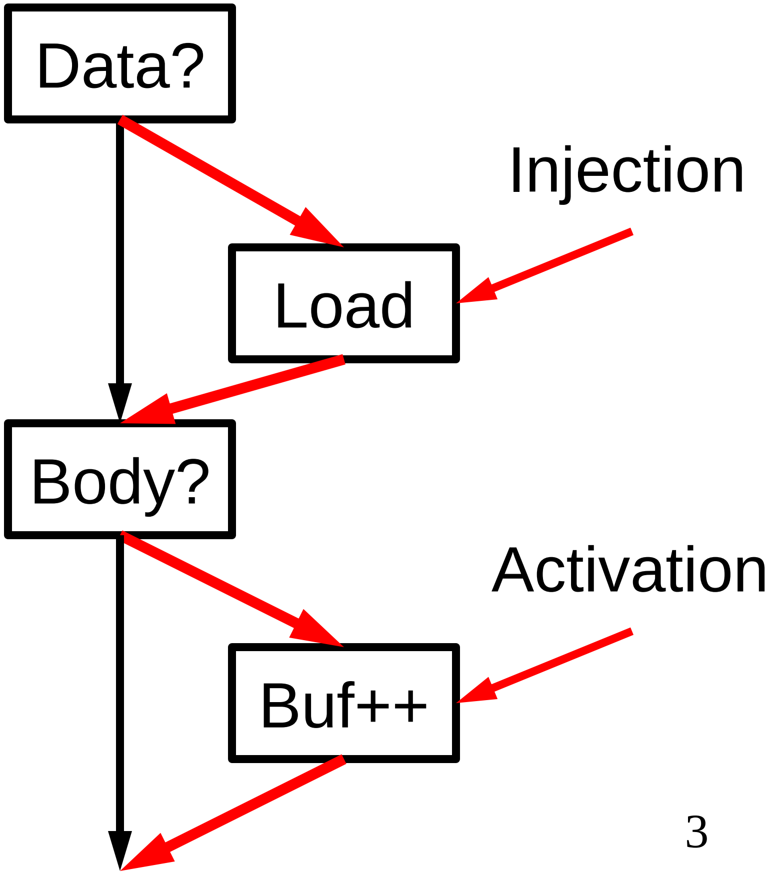
Importance of Path

- Vast majority of security attacks are enabled by software bugs

Often, hidden bugs only appear when sensitized by the proper path

Bugs escape Code/Branch coverage

Attackers will seek out code paths not tested



Path Test Complexity

- Path Explosion
  - Path space is exponential with length
  - Heavyweight test methods are slow
- Path coverage remains beyond reach
- Attackers seek to discover untested paths
- Necessitates new approach to achieve path testing

Dynamic Control Frontier

Dynamic Control Frontier

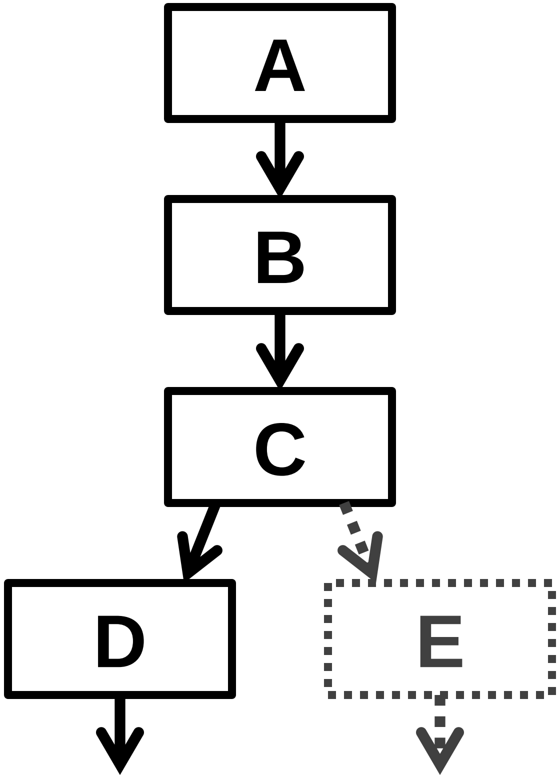
- Line of demarcation between dynamically seen paths of execution and those which are unseen

Dynamic Control Frontier

- Line of demarcation between dynamically seen paths of execution and those which are unseen
  - Frontier of path space explored by an application

Dynamic Control Frontier

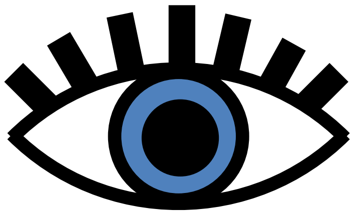
- Line of demarcation between dynamically seen paths of execution and those which are unseen

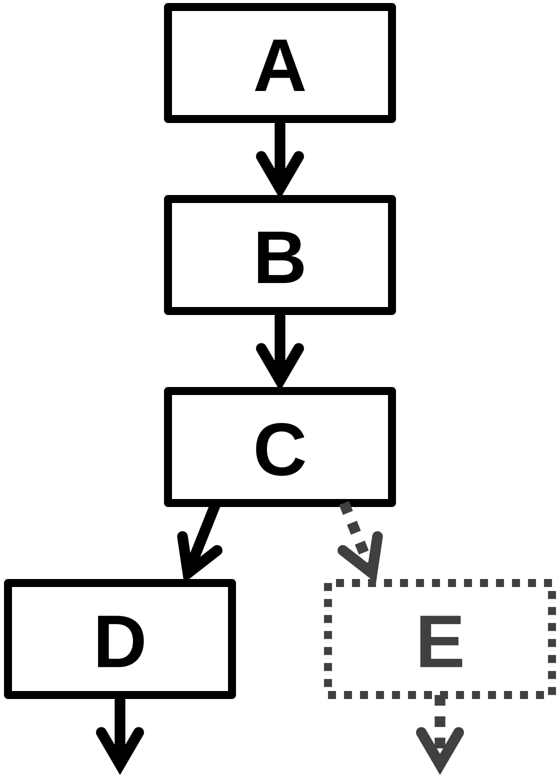


Dynamic Control Frontier

- Line of demarcation between dynamically seen paths of execution and those which are unseen

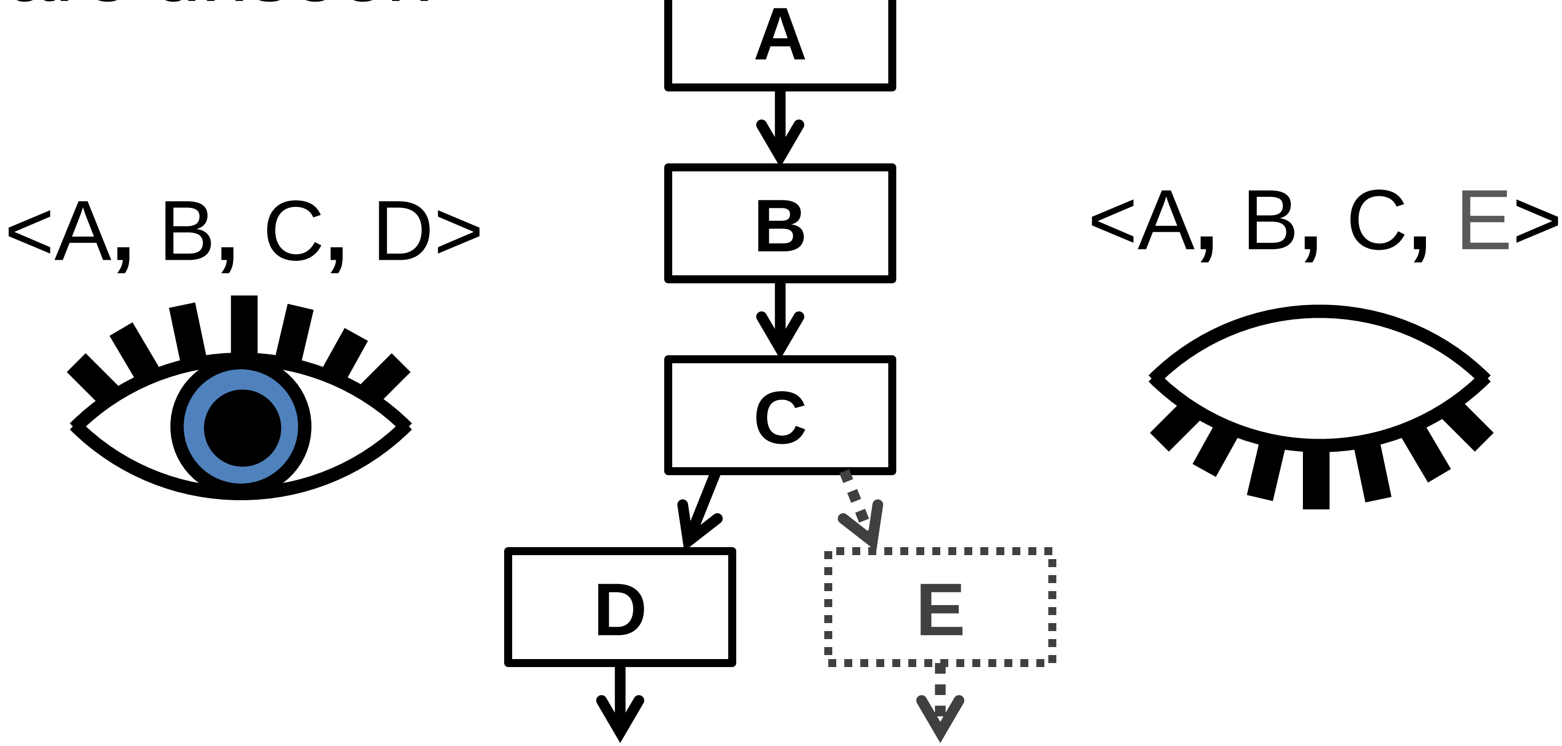
<A, B, C, D>





Dynamic Control Frontier

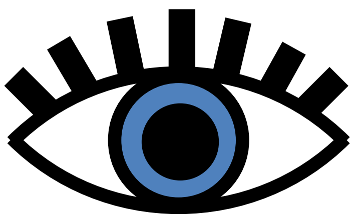
- Line of demarcation between dynamically seen paths of execution and those which are unseen

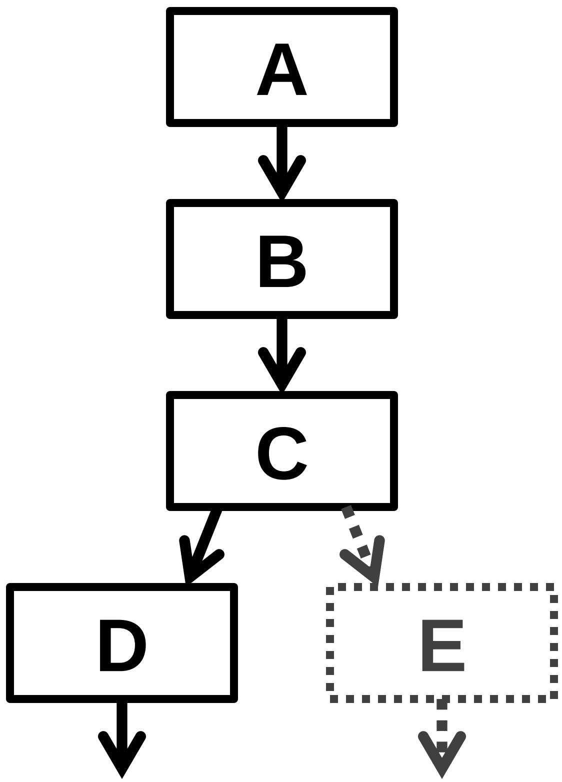


Dynamic Control Frontier

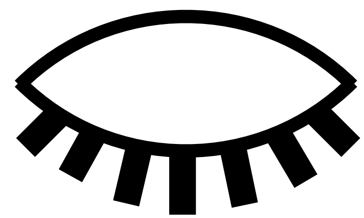
- Line of demarcation between dynamically seen paths of execution and those which are unseen

<A, B, C, D>





<A, B, C, E>



DCF = { <A, B, C, E> ,……., }

Value of the DCF

- Software Test Methodology:
  - Focus on reliability
  - Significant overlap in developer and user test
- Attacker Methodology:
  - Input permutations to deviate slightly from the expected, typical user execution
- Dynamic Control Frontier:
  - Intersection between heavily tested paths, and untested paths which are immediately reachable

Value of Distributed Analysis

- A single user:
  - Profiles an **instance**

Value of Distributed Analysis

- A single user:
  - Profiles an **instance**
- A non-trivial population of users:
  - Represents code paths not tested nor executed with any frequency by any user

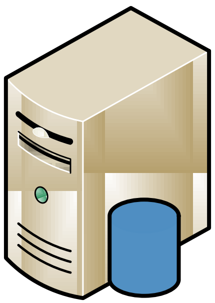
Value of Distributed Analysis

- A single user:
  - Profiles an **instance**
- A non-trivial population of users:
  - Represents code paths not tested nor executed with any frequency by any user

Distributed DCF Profiling

- User base profiles application via sampling

Users



Developer Analysis

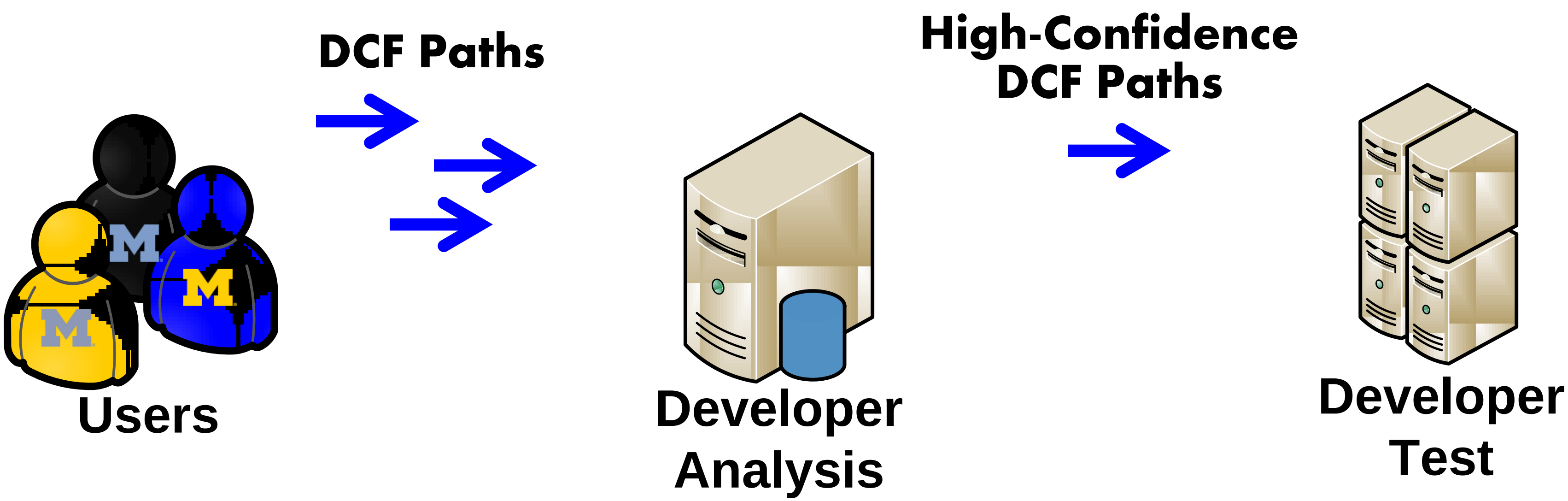
Developer Test

Distributed DCF Profiling

- User base profiles application via sampling

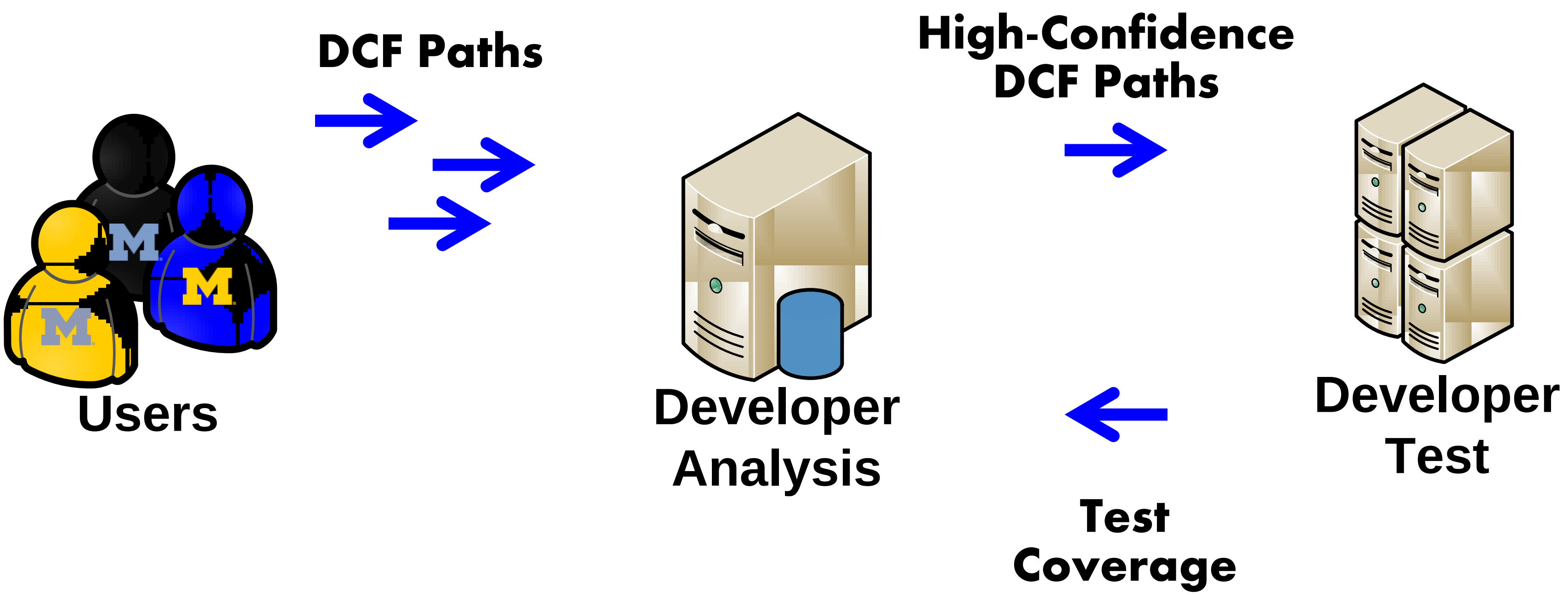
Distributed DCF Profiling

- User base profiles application via sampling



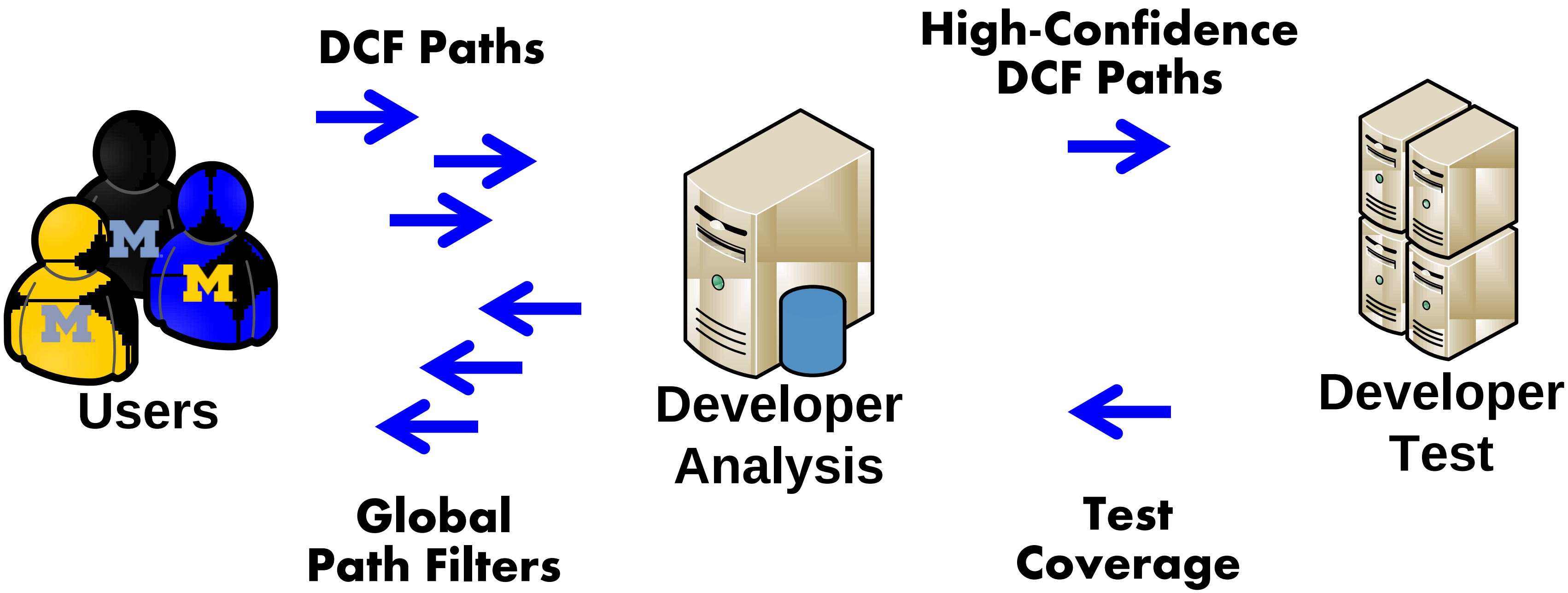
Distributed DCF Profiling

- User base profiles application via sampling



Distributed DCF Profiling

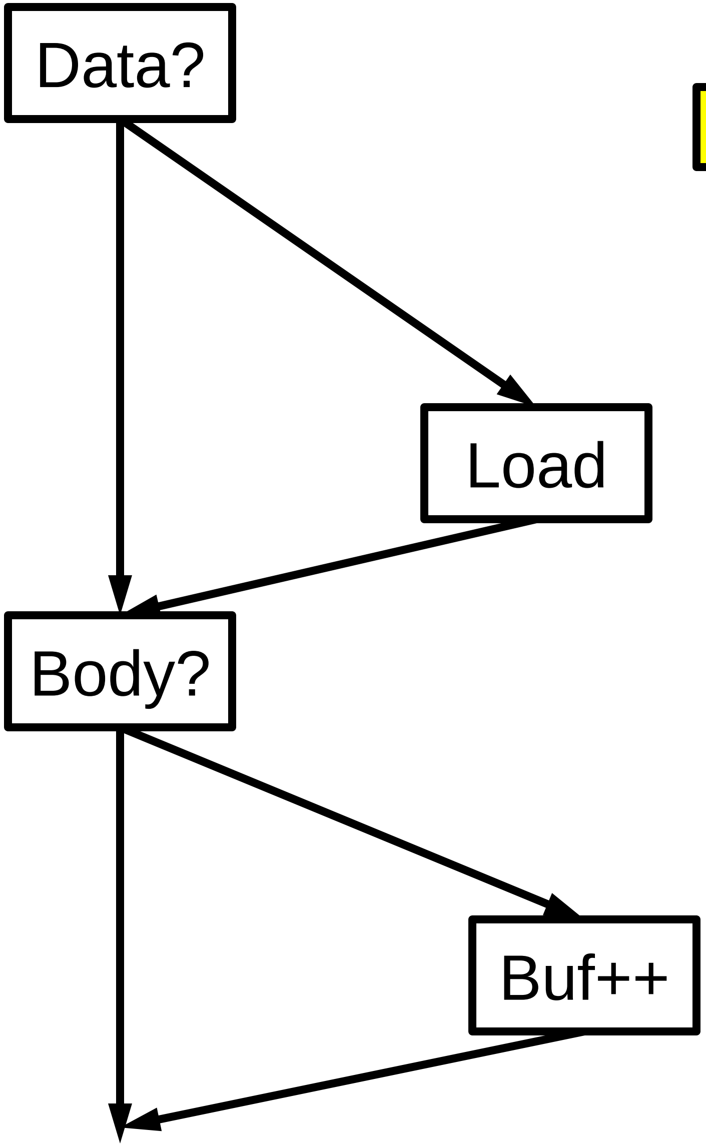
- User base profiles application via sampling



Experimental Evaluation

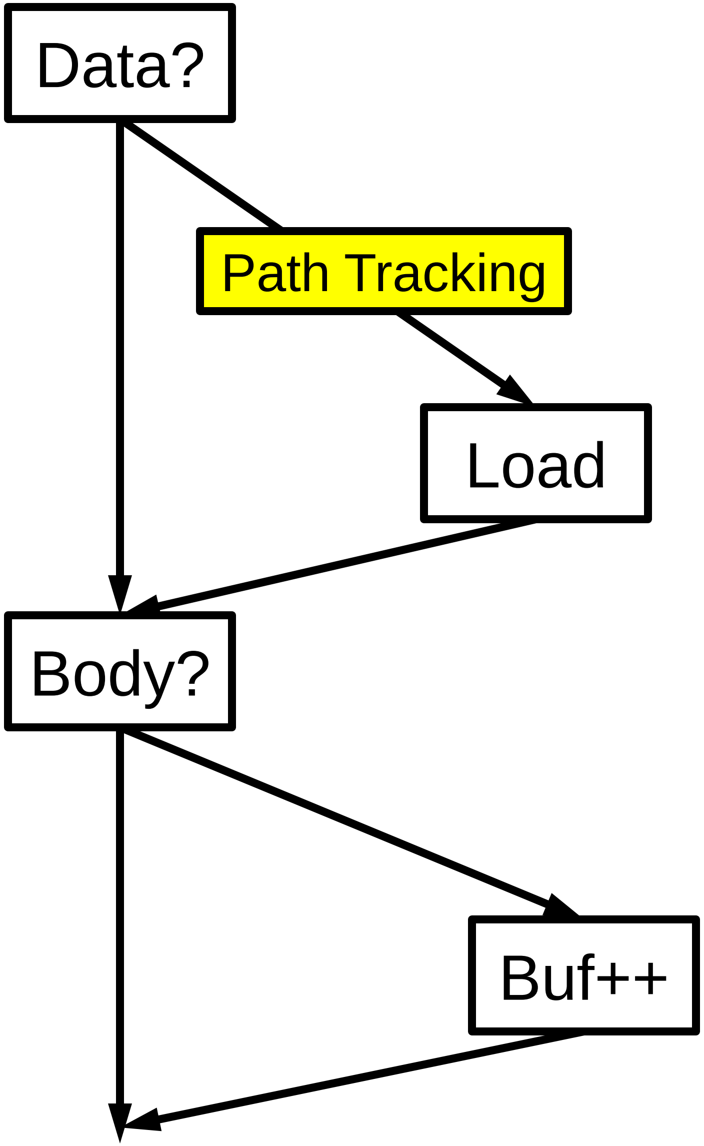
- DynamoRIO-based dynamic path profiling
  - Only instrument paths which are actively sampled

Experimental Evaluation

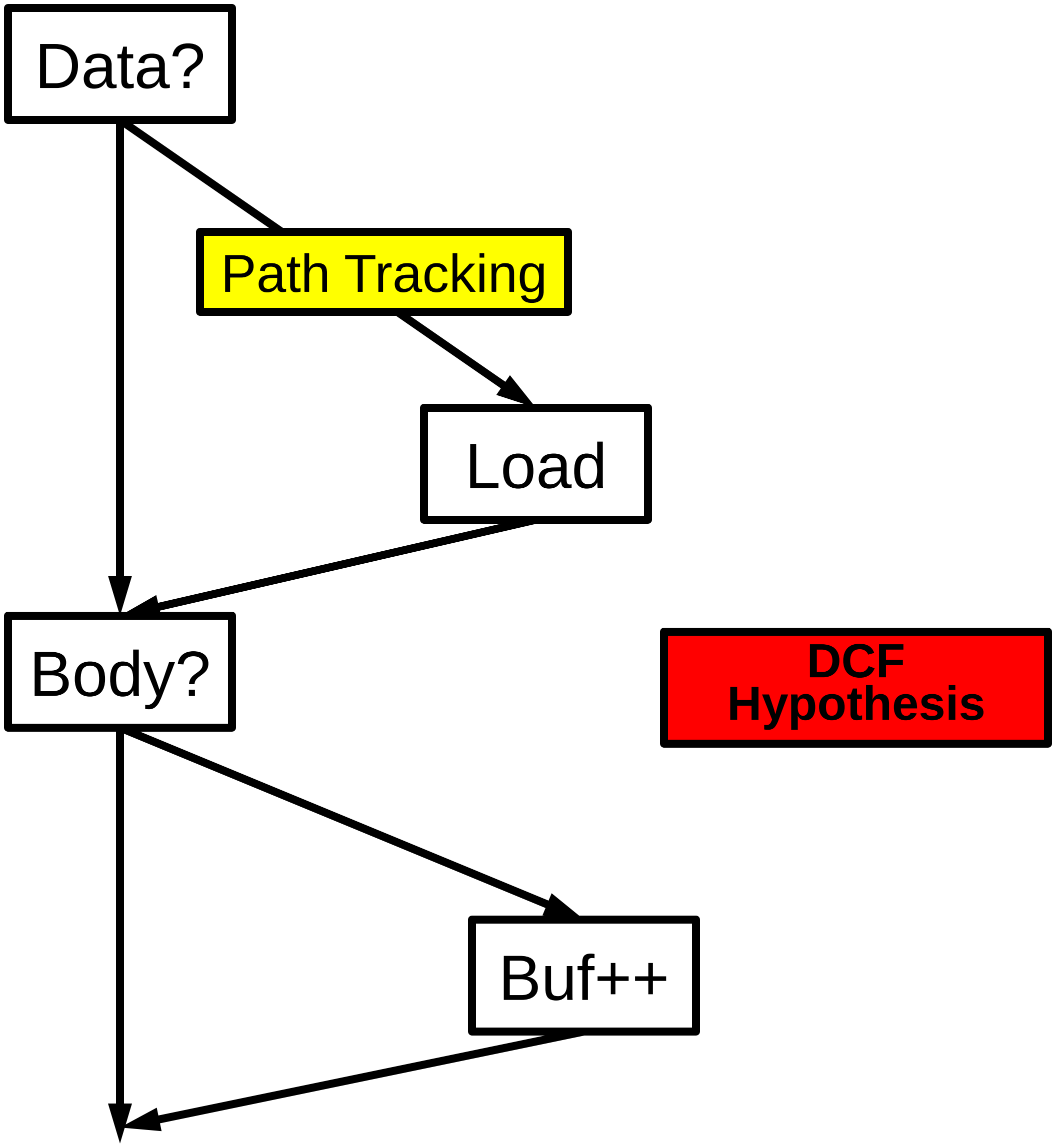


Path Tracking

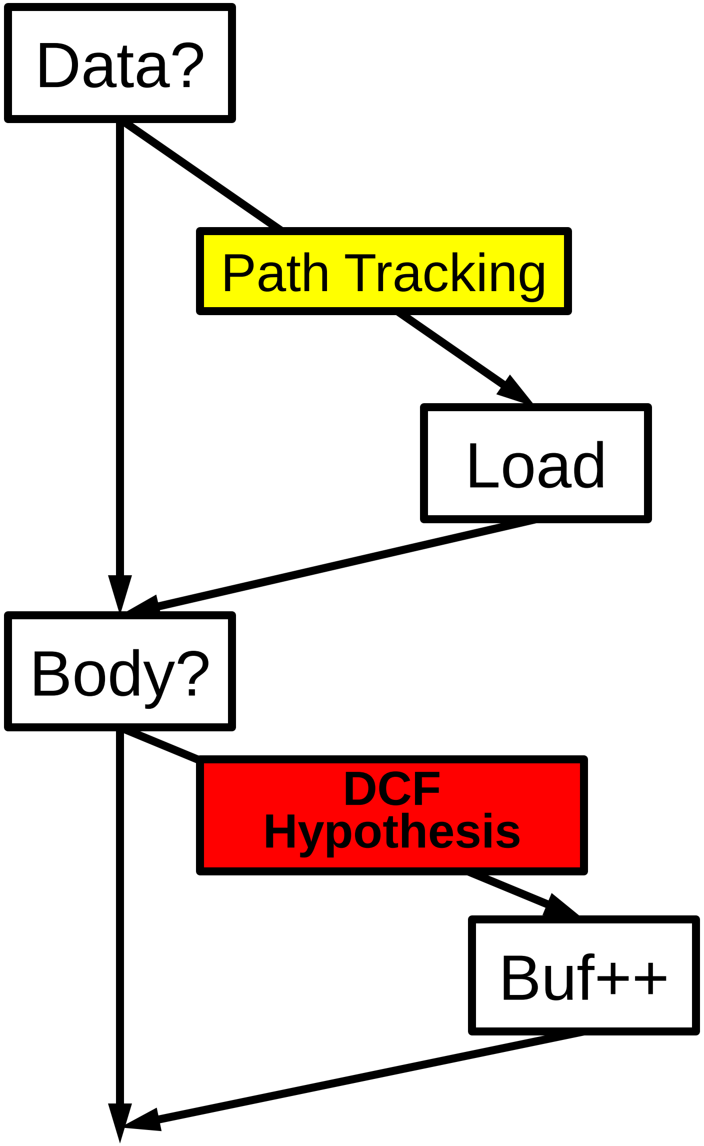
Experimental Evaluation



Experimental Evaluation



Experimental Evaluation

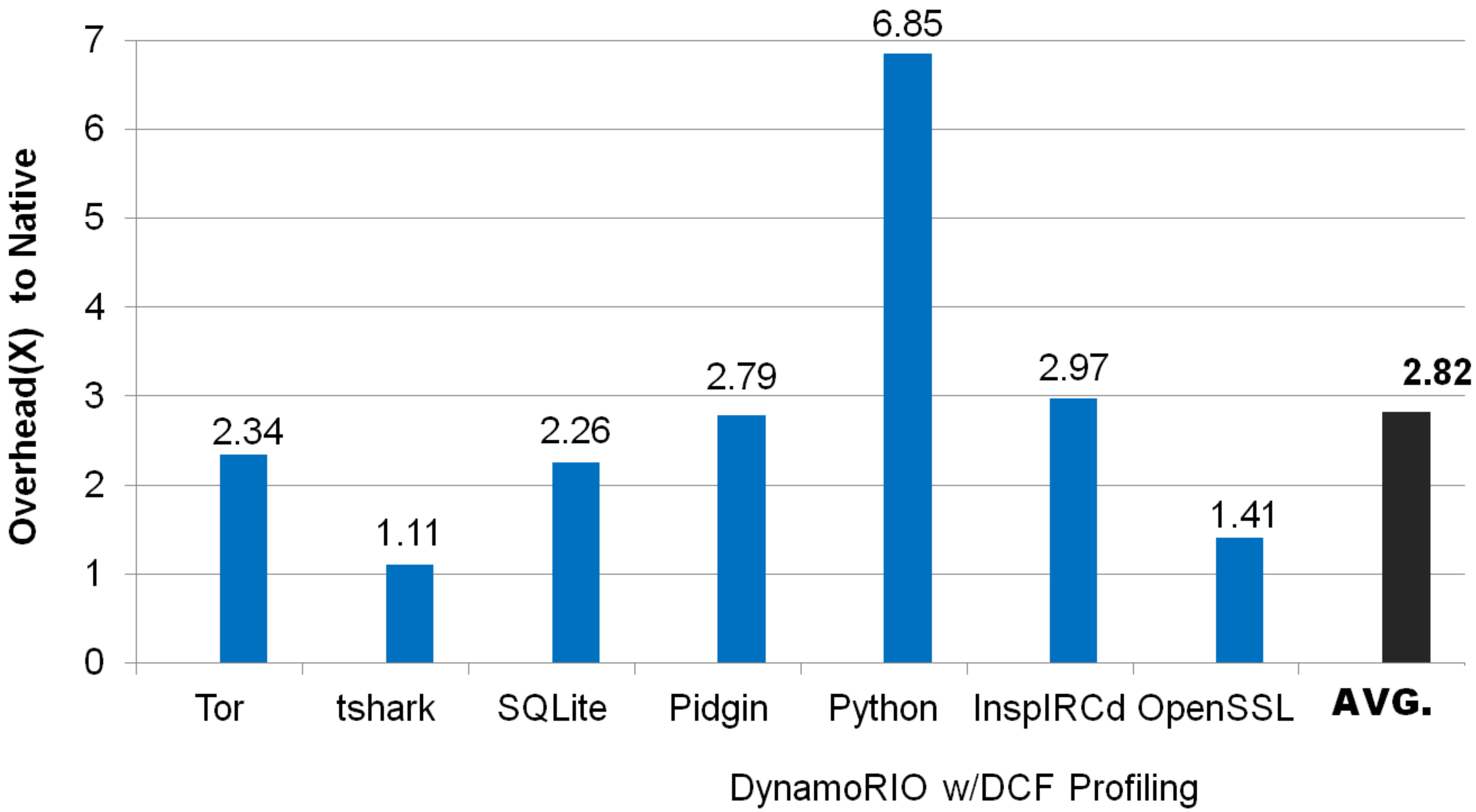


Benchmark Applications

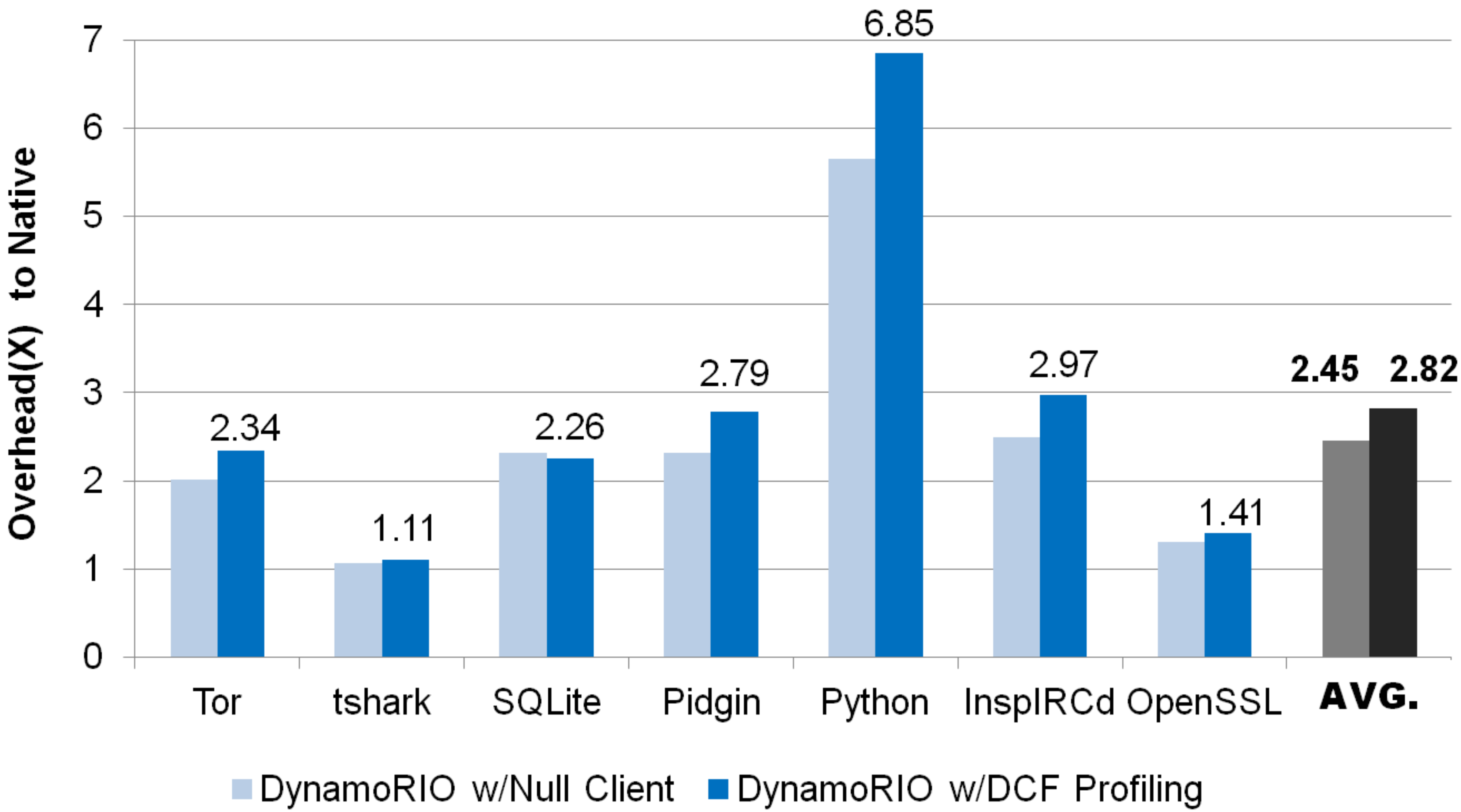
- Popular, network-facing applications

Application	# Instructions Profiled	# Potential Paths	# DCF Paths
SQLite	16,948,864,926	13,642,304	17,351
OpenSSL	5,014,034,838	23,221,696	10,086
tshark	684,000,546	38,467,136	178
Python	656,068,272	12,175,712	35,026
Tor	118,310,256	1,191,280	10,639
InspIRCd	46,246,206	11,165,696	3,950
Pidgin	4,762,914	6,833,360	3,641

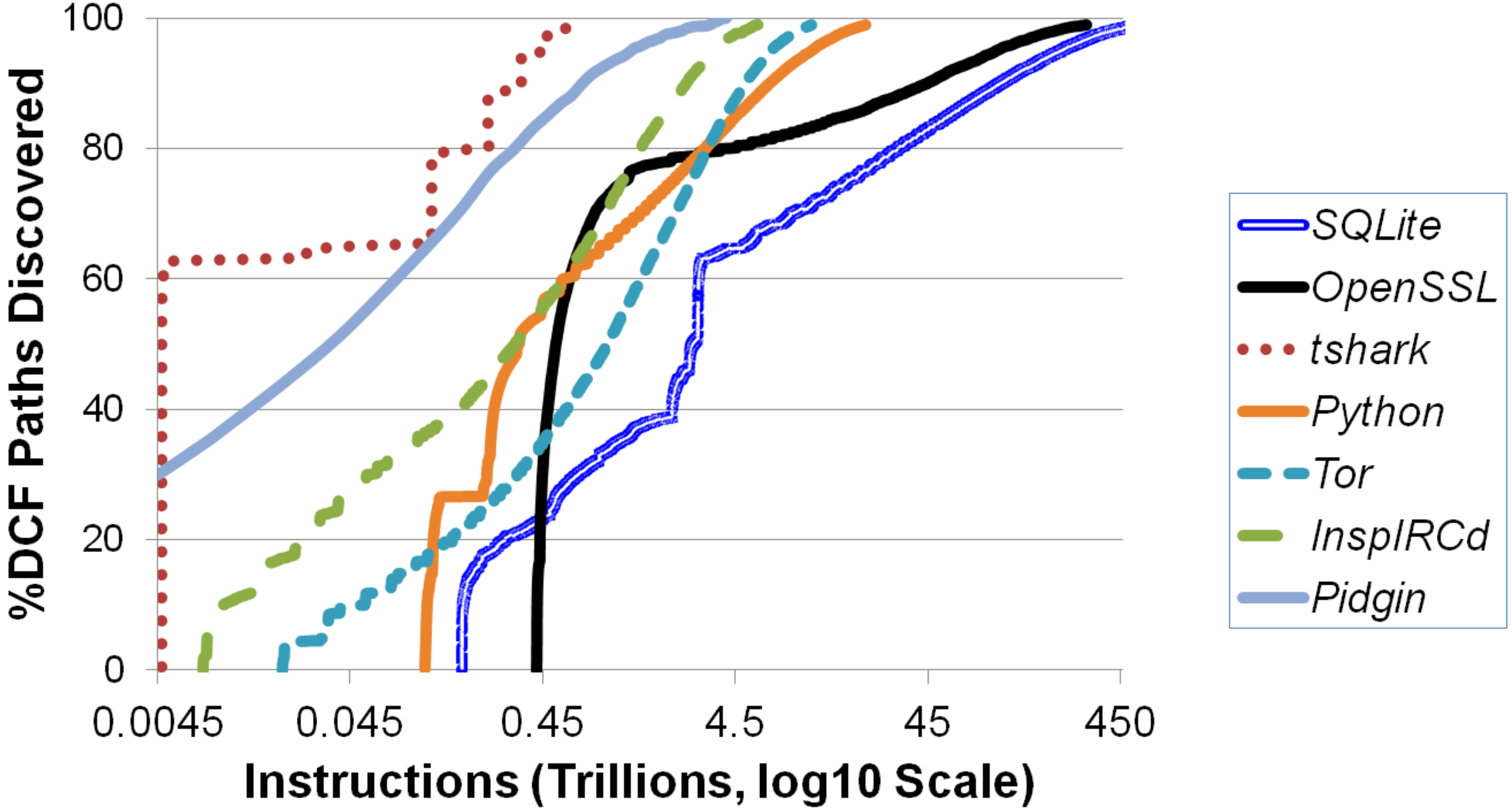
Profiling Overheads



Profiling Overheads



DCF Coverage



DCF Effectiveness

- Challenged Schnauzer to find known security bugs
  - Known bugs have precise code location

106 Million+ Potential Length-n Paths

80,000 DCF Paths

14 Security Bugs

{ *Buffer Overflow, Integer Underflow, DoS, Format String, Heap Overflow* }

DCF analysis would have given opportunity to determine paths for these bugs **before they were exploited**

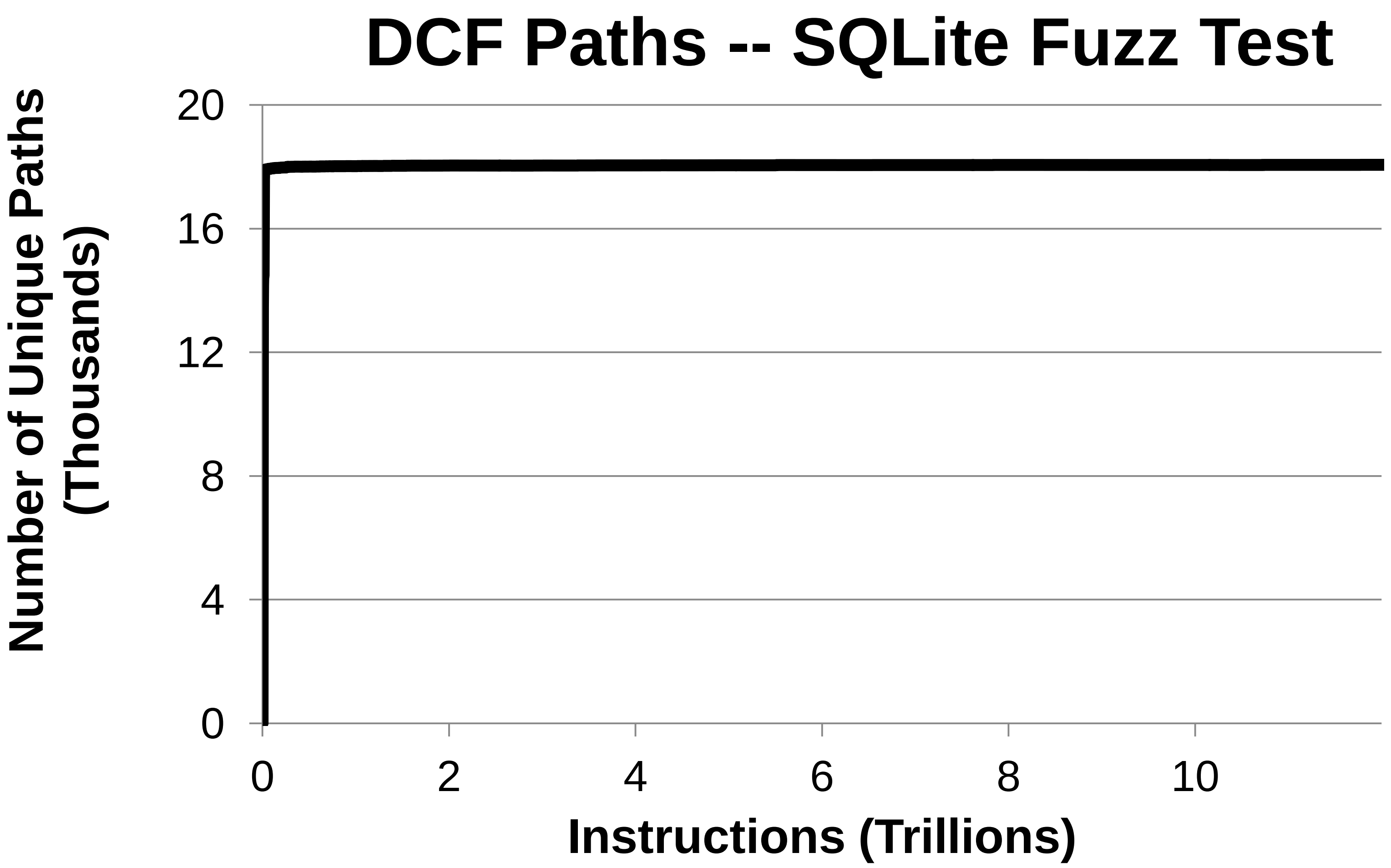
Conclusions & Future Directions

- Efficient, user-enabled DCF profiling can expand test for software security
- Identify code paths harboring bugs more likely to be exploited
  - Before they are exploited
  - Making software more secure
- Going Forward:
  - More efficient user profiling
  - Deployment of DCF for substantial application
  - Integration with state-of-art automated test

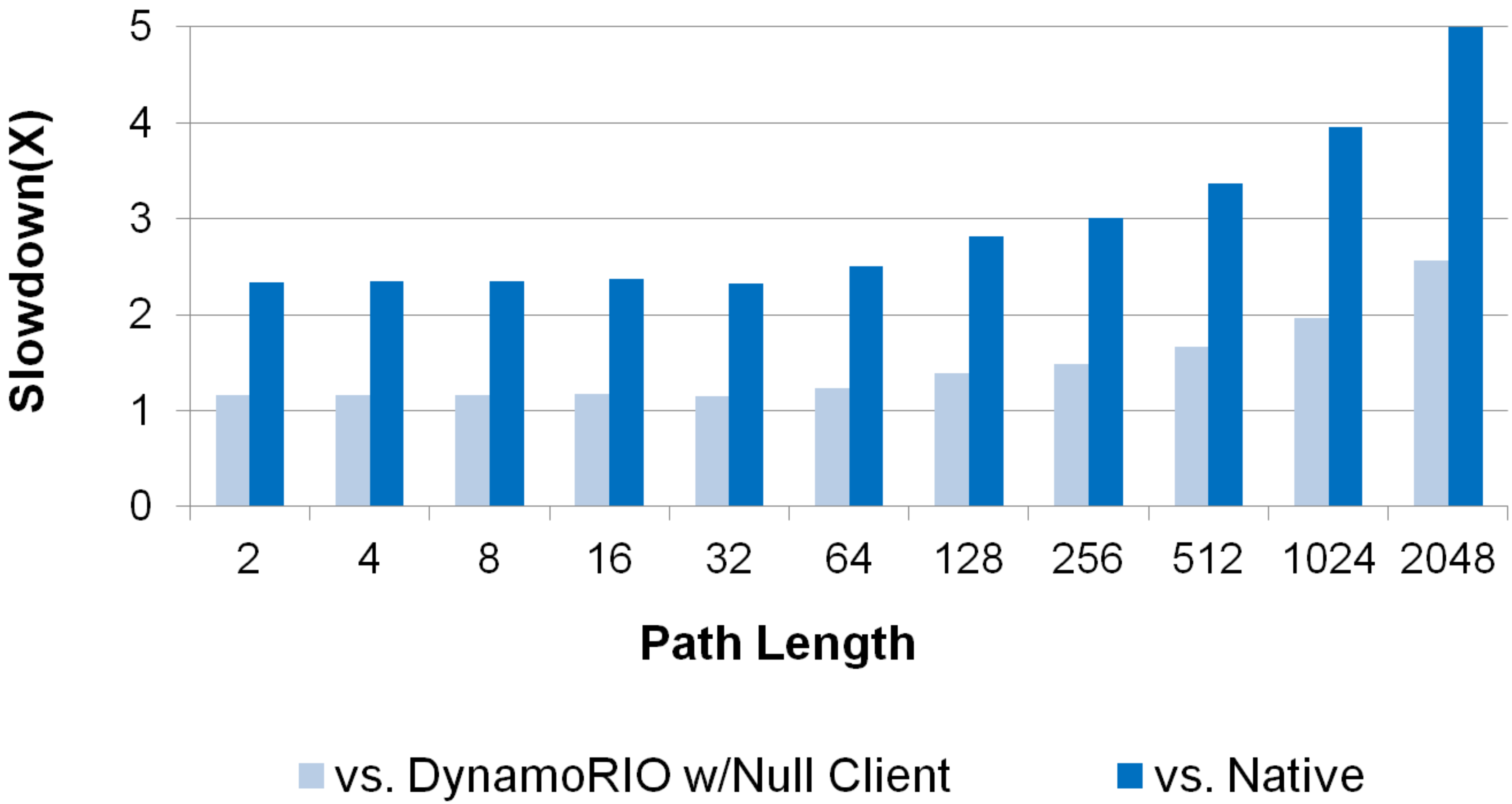
Thank You

Supplemental Material

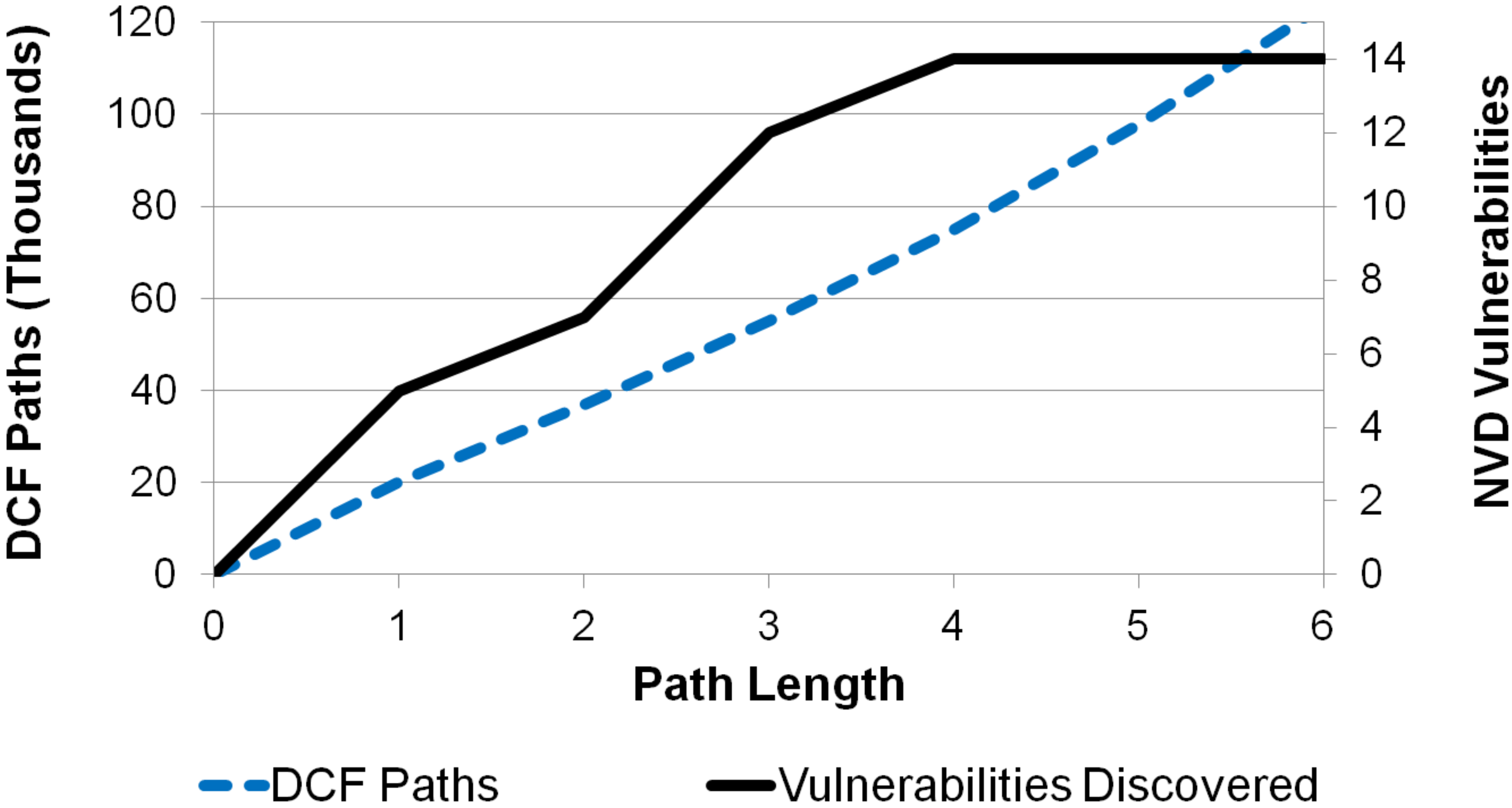
Profiling Scalability



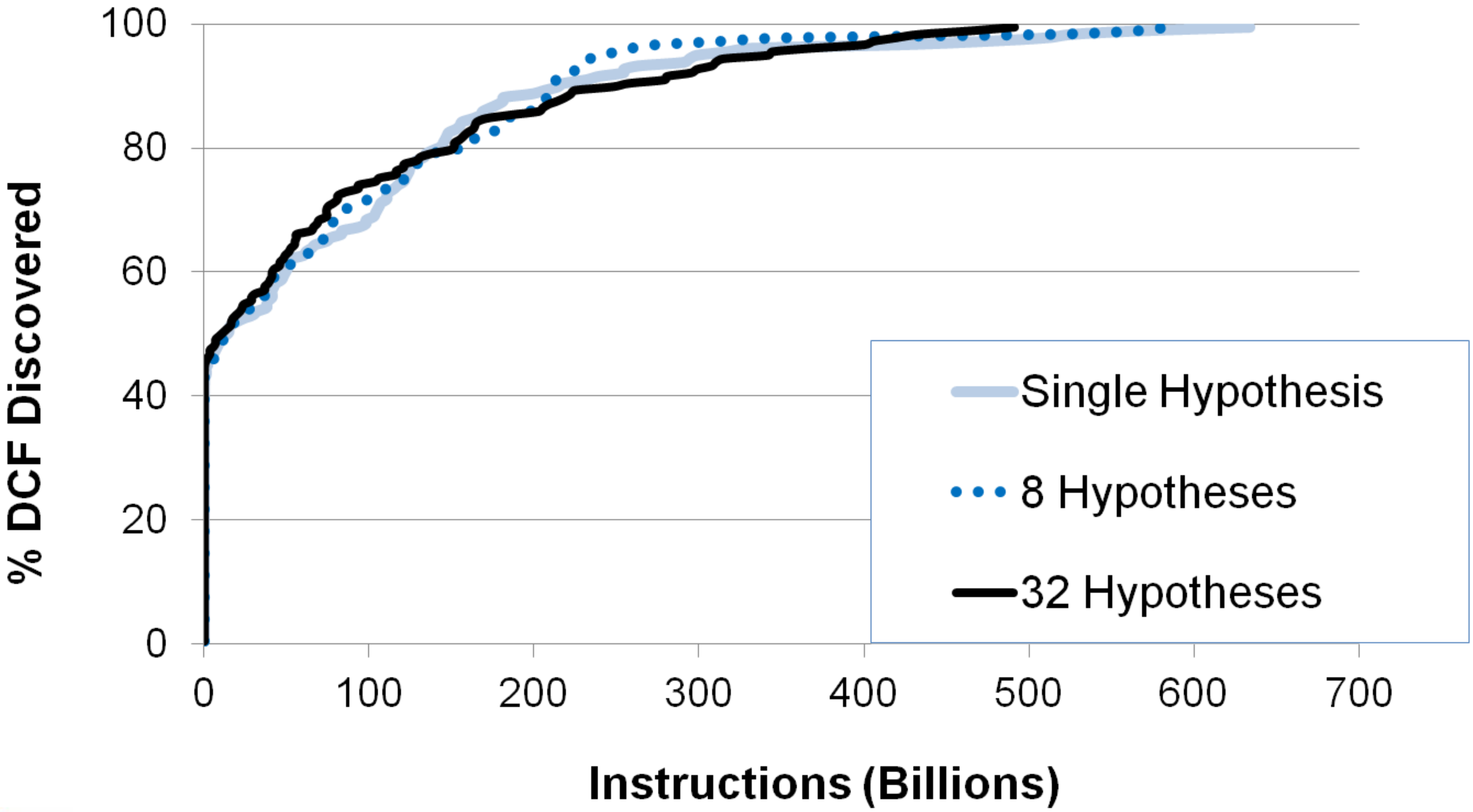
Path Length Scalability



Path Length : Vulnerability



Concurrent Hypotheses



DCF

$$DCF(P) = \{p_i, p_j, \ldots, p_m\}$$

$$p_i = \langle bb_1, bb_2, \ldots, bb_{n-1}, bb_n \rangle$$

$$\mid \langle bb_1, bb_2, \ldots, bb_{n-1} \rangle \in EX(P)$$

$$\wedge \langle bb_1, \ldots, bb_{n-1}, bb_n \rangle \notin EX(P)$$

$$EX(P) = \{\ldots \text{ all paths executed } \ldots\}$$