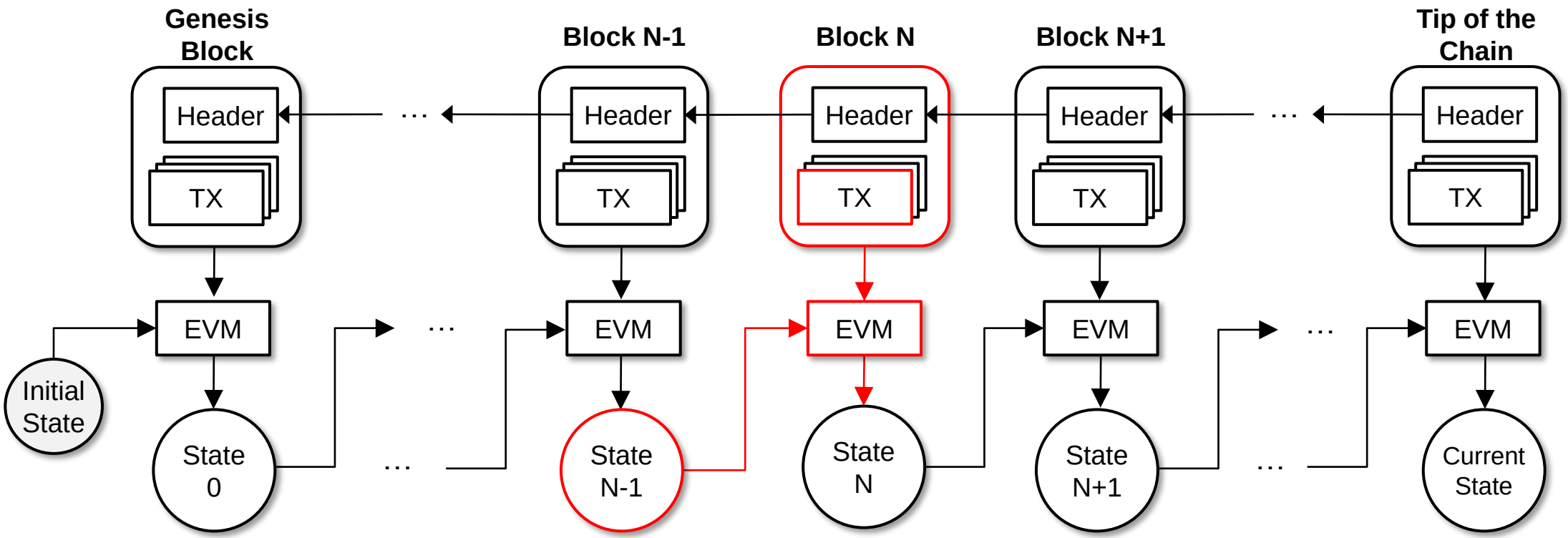
AN OFF-THE-CHAIN EXECUTION ENVIRONMENT FOR SCALABLE TESTING AND PROFILING OF SMART CONTRACTS

Yeonsoo Kim[1], Seongho Jeong[1], Kamil Jezek[2], Bernd Burgstaller[1], and Bernhard Scholz[2]

[1]Yonsei University, Korea
[2]The University of Sydney, Australia

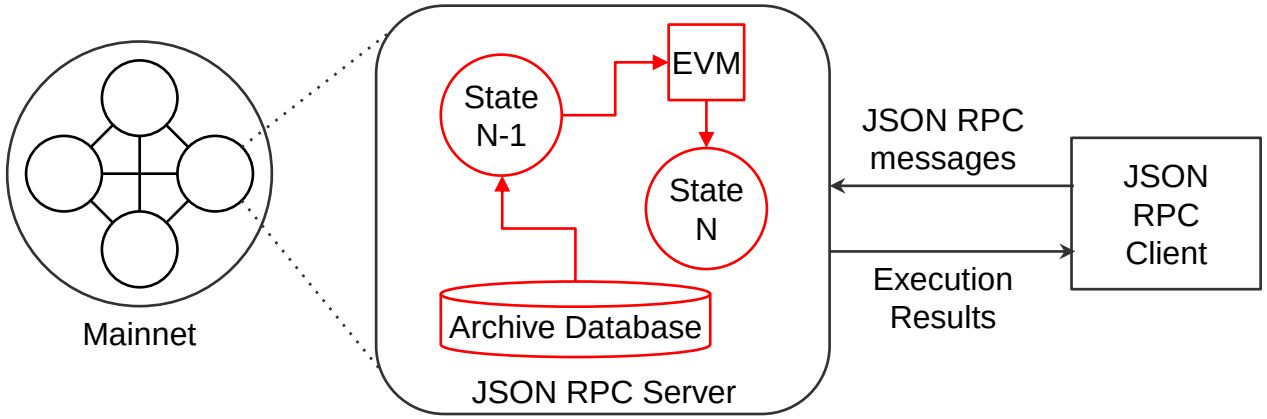
Problem: Transaction Replay

- Transaction (TX) replay
  - Important for auditing, testing, profiling, and debugging of smart contracts
  - Challenge: retrieval of the historical state that the transaction was originally executed on



Transaction Replay on Geth Archive Node

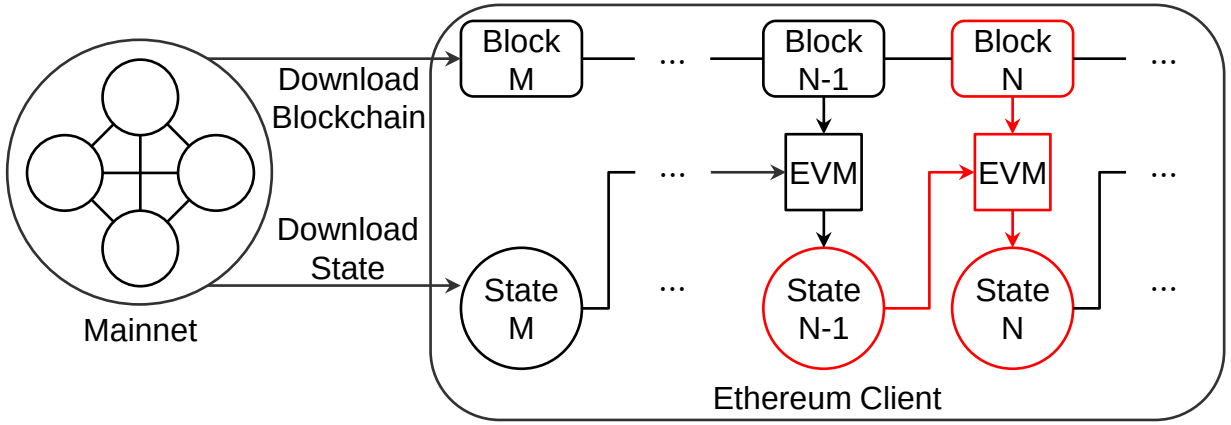
- Geth archive node
  - Entire history of world states stored in archive database
  - Delegation of transaction replay via JSON RPC



- Limitations of Geth archive node
  - Substantial time and disk space required to maintain archive database
    - 6 TB at block 11.5M
  - Very low transaction replay throughput
    - 19.4 tx/s at block 9M

Transaction Replay on Geth Full Node

- Geth full node in fast-sync mode
  - Starting from the intermediate state at block M instead of the genesis block
  - Less time and disk space required compared to a Geth archive node



- Limitations of Geth full node in fast-sync mode
  - Time required to process intermediate blocks from block M to block N-1
  - Space required to store a complete world state with all accounts from previous blocks
  - Only available for recent blocks

Efficient Transaction Replay is an Unsolved Problem

> ``*EVM is a single-threaded machine that cannot run transactions in parallel. ... This maybe a big problem for people who have a higher requirement on the timely reaction and verification of their transactions.*''

[50] Weiqin Zou, et al. ``Smart Contract Development: Challenges and Opportunities." *IEEE Transactions on Software Engineering, 2019.*

Developers suffering from ``*Reliability of and the lack of good development tools. Like testing frameworks, and the difficulty of debugging.*''

Developers need ``*Easy way of forking mainnets for testing purposes, ...*''

[5] Amiangshu Bosu, et al. ``Understanding the motivations, challenges and needs of Blockchain software developers: a survey." *Empirical Software Engineering, 24(4):2636–2673, 2019.*

Contributions

A testing environment for smart contracts on the Ethereum blockchain

- Purpose: fast re-execution of historical transactions

1) **Transaction record/replay mechanism** to enable off-the-chain execution of transactions
   - in isolation and at scale

2) **Re-organization of the Ethereum world state** into transaction-relevant **substates**
   - Space-efficient representation of the information required to faithfully replay a transaction

3) **Evaluation of the proposed approach** through three use cases:
   - a regression tester for hard forks
   - a dead-code analysis
   - a program fuzzer for smart contracts

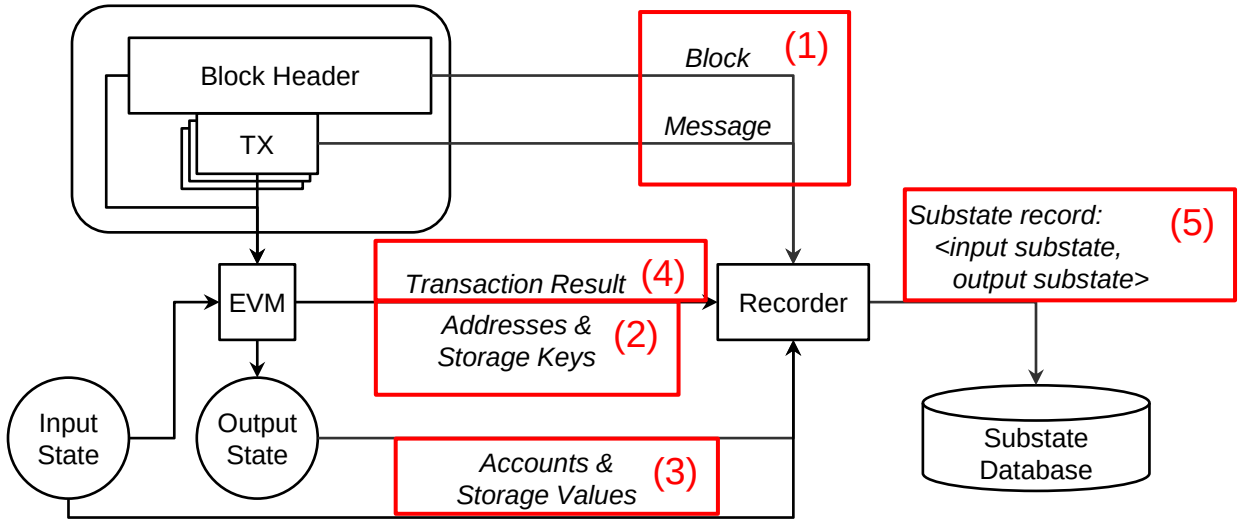
Solution (Overview)

- Transactions **recorded** by importing all blocks from the genesis block up to the tip of the chain.
  - Includes full cryptographic verification of blocks as mandated by the Ethereum protocol (**on-chain**)

- Historical state captured from before and after each transaction on the chain.
  - Restricted to the **subset of the world state** required to faithfully replay the transaction
  - No dependency on earlier transactions on the chain
  - Substates stored in the **substate database**

- Transaction replay from substate database conducted **off-the-chain**
  - Mocking the Ethereum protocol to execute transactions without the overhead of the distributed system
  - Absence of dependencies allows instant replay of historical transactions, in isolation and at scale

Off-The-Chain Testing – Recorder

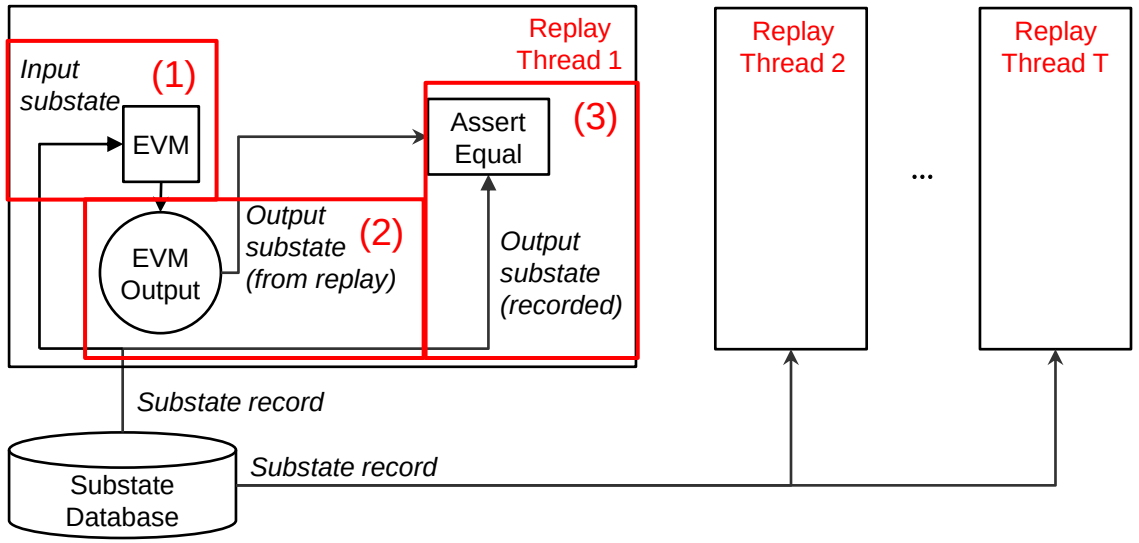


Input tuples:
<<address1,key2>, 25>
<<address2,key2>, 80>

Output tuples:
<<address1,key2>, 15>
<<address2,key2>, 90>

- Recorder captures substates during on-chain transaction execution
  - (1) Environment parameters (block, message)
  - (2) Accessed indices (addresses, storage keys)
  - (3) Accessed key-value pairs (input/output tuples) from account storage
  - (4) Transaction result (status code, logs, gas usage)
  - (5) Collection of input/output substates
- Example: token transfer of 10 units at *key2* of account 1 (*<address1,key2>*) to *key2* of account 2 (*<address2,key2>*)

Off-The-Chain Testing – Replayer



Record input:
<<address1,key2>, 25>
<<address2,key2>, 80>

Record output:
<<address1,key2>, 15>
<<address2,key2>, 90>

unequal

Replay input:
<<address1,key2>, 25>
<<address2,key2>, 80>

Replay output:
<<address1,key2>, 0>
<<address2,key2>, 90>

``Assert Equal'' will fail

- Replayer loads substates and replays transactions off-the-chain
  - (1) Copies input substate to in-memory EVM context
  - (2) Executes the transaction in isolation and captures its output substate as the recorder did
  - (3) Checks correctness of transaction replay
    - Raises an exception if accessed indices or the replay output differ from the recorded output
- The elimination of dependencies on earlier transactions enables transaction replay at scale

Evaluation – System Configuration

Server	Geth full node	Substate replayer	Geth archive node
CPU	Intel Xeon E5-2699 v4, 2.2 GHz to 3.6 GHz, 22 cores × 2 sockets		
RAM	512 GB DDR4 RAM @ 2,400 MHz		
SSDs (PCIe)	Intel Optane 900P 480 GB Intel Optane DC P3700 800 GB		Samsung PM1725b 6.4 TB
OS	CentOS Linux release 7.9.2009 (core), kernel version 4.11.3-1.el7.elrepo.x86_64		
Filesystem	ZFS pool (1.2 TB total size)		XFS

- Evaluated for the initial 9M blocks of the Ethereum blockchain
  - Containing 590M transactions
- Replayer finished without raising an exception, signifying a 100% replay accuracy

Evaluation – Replay Performance (Space)

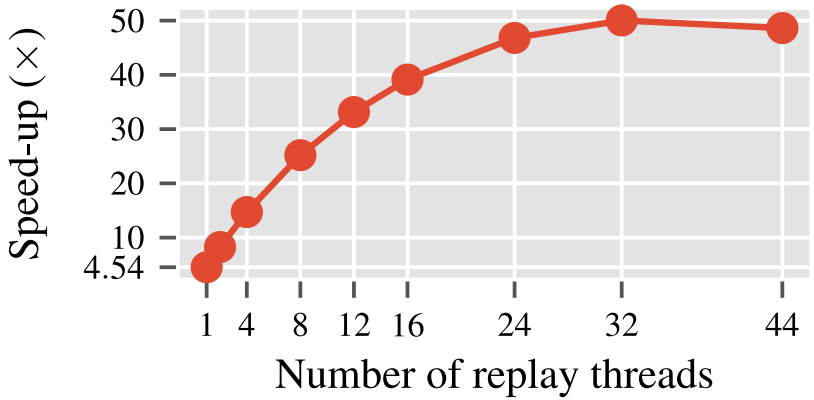
- Substate database saves 59.24% of disk space compared to a Geth full node

Blocks (M)	Geth full node (GB)	Substate replayer (GB)	Space savings (%)
0–1	0.96	0.68	29.17
1–2	3.00	1.83	39.00
2–3	29.30	16.91	42.29
3–4	37.76	6.58	82.57
4–5	124.57	39.55	68.25
5–6	272.06	51.58	81.04
6–7	407.48	50.30	87.66
7–8	551.04	55.96	89.84
8–9	700.11	62.00	91.14
0–9	700.11	285.39	59.24

Evaluation – Replay Performance (Time)

- 4.54 times faster than a Geth full node with a single thread
- When scaled to 44 cores, replay of 590M transactions finishes within six hours

Blocks (M)	Geth full node		Substate replayer		Speed-up (×)
	Time (s)	tx/s	Time (s)	tx/s	
0–1	1184	1414.07	526	3183.01	2.25
1–2	2879	2217.13	1517	4207.73	1.90
2–3	28906	252.73	24125	302.82	1.20
3–4	10775	1946.33	5222	4016.03	2.06
4–5	94868	1196.67	28873	3931.90	3.29
5–6	165673	748.71	35390	3504.97	4.68
6–7	173503	552.82	33672	2848.55	5.15
7–8	224426	485.51	38060	2862.87	5.90
8–9	248519	447.70	41999	2649.17	5.92
0–9	950733	620.62	209384	2817.98	4.54



Use Cases

- Demonstration of effectiveness of our record/replay infrastructure for dynamic analyses

1) **Metric use case**

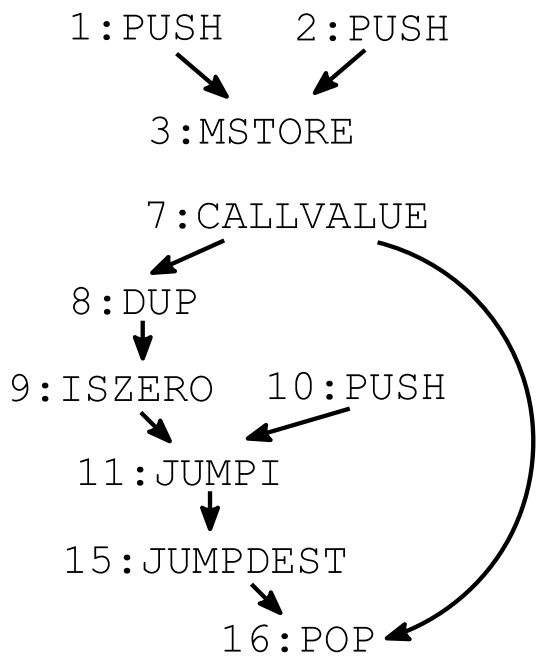
2) **Fuzzer use case**

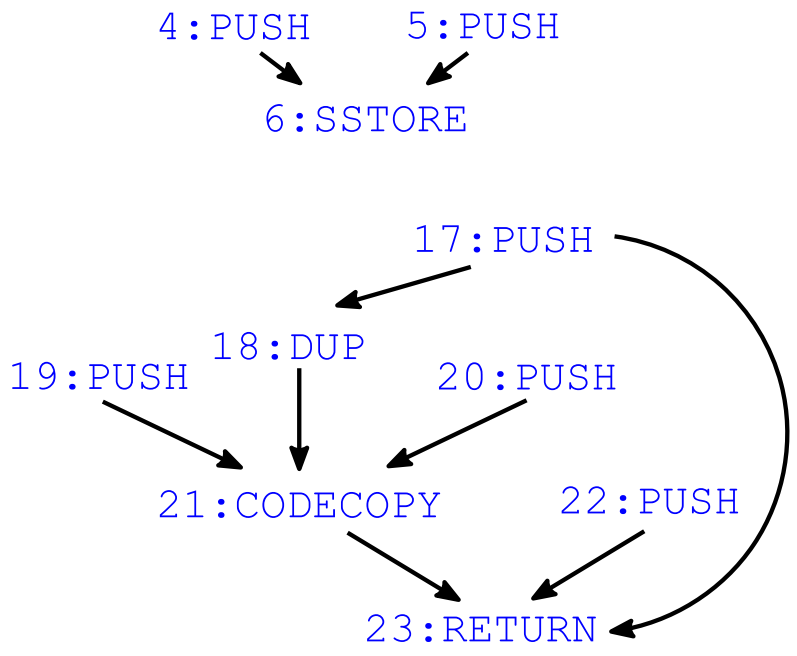
3) **Hard fork assessment**

Use Case 1: Metric

- Wasteful instructions have no lasting side effect on the blockchain state
- Propagation of necessity of instructions in the value graph in a backward fashion
  - `6:SSTORE` and `23:RETURN`
  - All instructions that they depend upon (by transitivity)

Nr.	Opcode	Nr.	Opcode
1	PUSH 0x80	13	DUP
2	PUSH 0x40	14	REVERT
3	MSTORE	15	JUMPDEST
4	PUSH 0x00	16	POP
5	PUSH 0x01	17	PUSH 0x3797
6	SSTORE	18	DUP
7	CALLVALUE	19	PUSH 0x25
8	DUP	20	PUSH 0x00
9	ISZERO	21	CODECOPY
10	PUSH 0x15	22	PUSH 0x00
11	JUMPI	23	RETURN
12	PUSH 0x00		





Use Case 1: Metric (cont.)

- The analysis of 9M blocks took 75 hours
- Revealed increasing wasteful instruction ratios and wasted gas per transaction with later blocks
- 50% instructions and 25k gas are wasteful per transaction in block range 8-9M

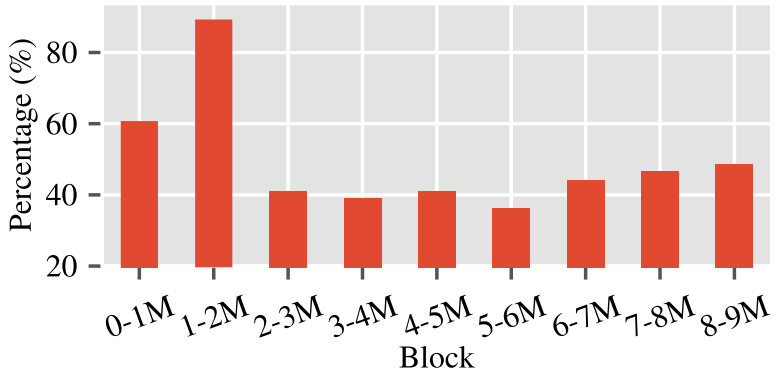


Figure 9: Average ratios of wasteful instructions for ranges of 1 M blocks.

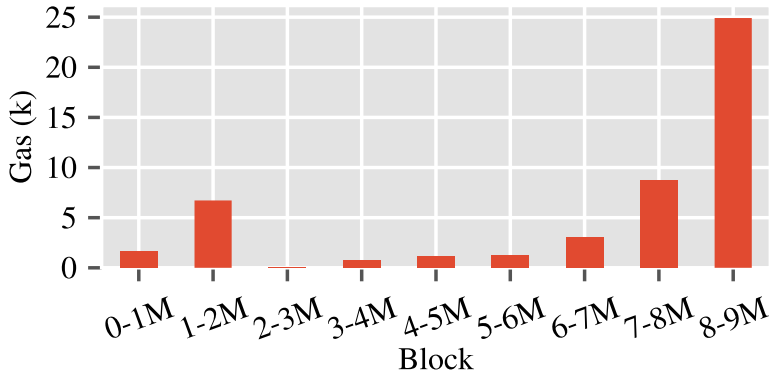
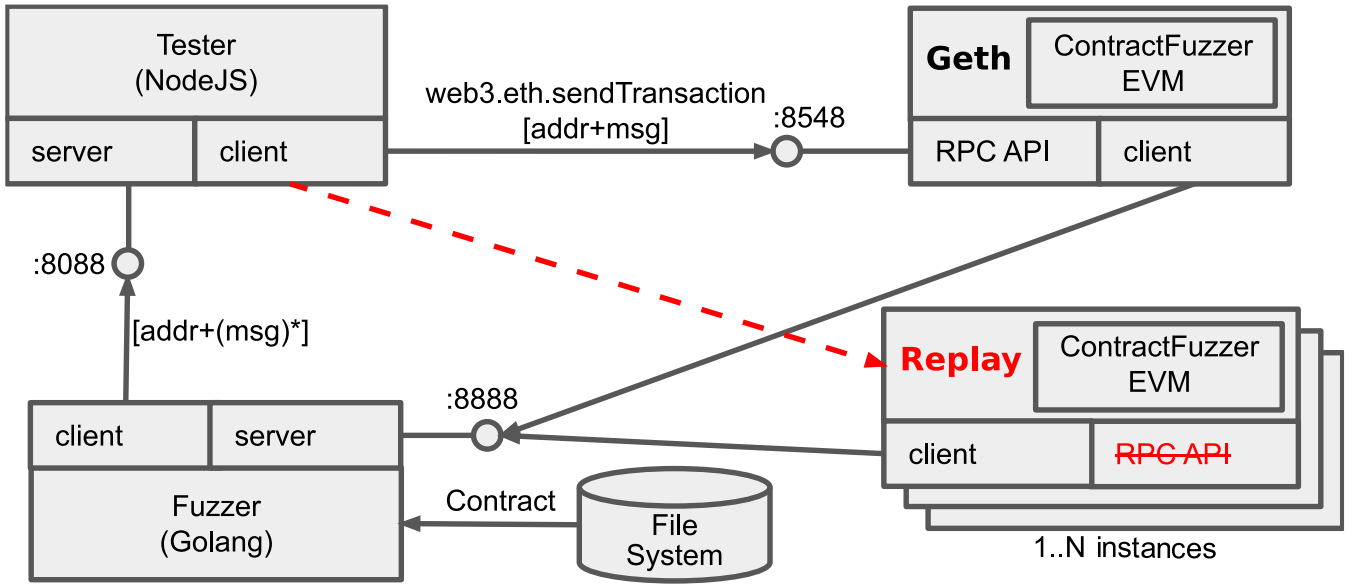


Figure 10: The average of wasted gas per transaction for ranges of 1 M blocks.

Use Case 2: Fuzzer

- Fuzzers test programs with randomized inputs
  - Low-performing fuzzers limit the effectiveness and may result in false negatives
- Demonstration of the effectiveness of parallel replay for fuzzing
  - Integration of our off-the-chain replayer with ContractFuzzer



Use Case 2: Fuzzer (cont.)

- Evaluated with substates recorded from the Ethereum mainnet
  - Increased throughput of contracts with a single thread
  - Established scalability on multicore architectures

Number of threads		**Throughput** (contracts/min)	**Speedup** (×) **over original**	**Speedup** (×) **replay**
Original	1	0.46	n/a	n/a
Replayer	1	3.28	7.07	n/a
	4	4.65	10.01	1.42
	8	6.25	13.46	1.90
	16	10.05	21.65	3.06
	32	15.99	34.44	4.87

Use Case 3: Hard Fork Assessment

- Ethereum specification has been updated over the years via hard forks

- Hard forks that update EVM specification may cause problems with already-deployed contracts
  - 680 contracts of the Aragon framework reportedly failed due to EIP-1884
  - Important to assess effects of hard forks on existing contracts, as reported by the developer community:

> *``Security and backward compatibility are held with utmost importance here ..."*
>
> *``... we have to consider backward compatibility all the time."*

[5] Amiangshu Bosu, et al. “Understanding the motivations, challenges and needs of Blockchain software developers: a survey.” *Empirical Software Engineering, 24(4):2636–2673, 2019.*

Use Case 3: Hard Fork Assessment (cont.)

Hard fork	Assessed transactions (M)	EVM runtime exception (%)			Output changed (%)	Gas usage changed (%)			Unaffected (%)
		Invalid JUMP	Invalid opcode	Reverted		Out-of-gas	Increased	Decreased	
Homestead	0.416	0.000	0.000	0.000	0.000	0.002	0.000	0.000	99.998
Tangerine Whistle	2.508	0.585	0.000	0.000	3.667	2.761	75.125	0.003	17.859
Spurious Dragon	3.055	0.530	0.000	0.000	9.407	2.549	71.182	0.001	16.331
Byzantium	26.014	0.062	0.000	0.000	2.560	0.299	8.359	0.001	88.718
Constantinople / Petersburg	193.668	0.008	0.000	0.000	0.344	0.040	1.123	0.000	98.485
Istanbul	303.300	0.064	0.198	1.009	3.804	7.884	68.494	18.547	0.000

- Analysis took 15.15 hours for a total of 529M transactions
- Effects on execution path: ``EVM runtime exception'' + ``Output changed''
- Effects on gas consumption: ``Gas usage changed''

- Hard forks with highest ratios of effects on both execution path and gas consumption
  - *Spurious Dragon, Tangerine Whistle, Istanbul*
  - Mainly increased gas costs of EVM instructions

Conclusion

- Our work proposes a new infrastructure for the lightweight execution of smart contracts in isolation and at scale.
  - Single threaded execution of all available smart contracts, 4.54 times faster than a Geth full node
  - Scaled effectively on a multicore architecture, 50.03 times faster on 44 cores

- Our infrastructure is highly suitable for scenarios that require the fast and repeated execution of smart contracts.
  - As demonstrated through three use cases:
    1) our metric use case for dead code analysis
    2) a program fuzzer for smart contracts
    3) the assessment of hard forks.

- Our infrastructure scales for the whole blockchain, which is not attainable with prior approaches.

Q&A

Thank you!

Contact information for follow-up questions:

Mr. Yeonsoo Kim
Department of Computer Science
Yonsei University
Email: yeonsoo.kim@yonsei.ac.kr