Toward Scalable Docker-Based Emulations of Blockchain Networks

Diego Pennino and Maurizio Pizzonia
26 May 2023, Bologna

5th Distributed Ledger Technology Workshop
DLT 2023
Performing realistic experiments for blockchain networks is notoriously hard.
Performing realistic experiments for blockchain networks is notoriously hard.

The complexity:
Performing realistic experiments for blockchain networks is notoriously hard.

The complexity:

- Large number of nodes,
Blockchain Network Testing

Performing realistic experiments for blockchain networks is notoriously hard.

The complexity:

- Large number of nodes,
- Software complexity,
Performing realistic experiments for blockchain networks is notoriously hard.

The complexity:

- Large number of nodes,
- Software complexity,
- Communications among nodes (properties of transport protocols),
Performing realistic experiments for blockchain networks is notoriously hard.

The complexity:

- Large number of nodes,
- Software complexity,
- Communications among nodes (properties of transport protocols),
- Nodes are spread over the internet (delay and packet loss)
Performing realistic experiments for blockchain networks is notoriously hard.

The complexity:

- Large number of nodes,
- Software complexity,
- Communications among nodes (properties of transport protocols),
- Nodes are spread over the internet (delay and packet loss)

Emulation or Simulation???
Performing realistic experiments for blockchain networks is notoriously hard.

The complexity:

- Large number of nodes,
- Software complexity,
- Communications among nodes (properties of transport protocols),
- Nodes are spread over the internet (delay and packet loss)

**Emulation** or **Simulation**???
Performing realistic experiments for blockchain networks is notoriously hard.

The complexity:

- Large number of nodes, Simulation
- Software complexity, Emulation
- Communications among nodes (properties of transport protocols), Emulation
- Nodes are spread over the internet (delay and packet loss) Emulation

Emulation or Simulation???
Performing realistic experiments for blockchain networks is notoriously hard.

The complexity:

- Large number of nodes, Simulation
- Software complexity, Emulation
- Communications among nodes (properties of transport protocols), Emulation
- Nodes are spread over the internet (delay and packet loss) Emulation

Emulation or Simulation???
Emulation: two choices
Emulation: two choices

**PRO:**
- Simple to handle
- Simple to modify

**CONS:**
- Small amount of nodes (few hundreds)
- Fake Networks

Local
Emulation: two choices

**PRO:**
- Huge amount of nodes
- “Real” Network environment

**CONS:**
- Hard to handle
- Slow to modify
- Clusters

**Local**

**Distributed**

**PRO:**
- Simple to handle
- Simple to modify

**CONS:**
- Small amount of nodes (few hundreds)
- Fake Networks

**Local**

**Distributed**
Emulation: Our Solution

**PRO:**
- Huge amount of nodes
- "Real" Network environment

**CONS:**
- Hard to handle
- Slow to modify
- Clusters

**PRO:**
- Simple to handle
- Simple to modify

**CONS:**
- Small amount of nodes (few hundreds)
- Fake Networks

**PRO:**
- Large amount of nodes
- "Real" Network environment

**CONS:**
- Hard to handle
- Slow to modify
- Clusters

**Local**

**Distributed**
Emulation: Our Solution

Boosted Hardware on premises

**PRO:**
- Huge amount of nodes
- “Real” Network environment

**CONS:**
- Hard to handle
- Slow to modify
- Clusters

**PRO:**
- Simple to handle
- Simple to modify

**CONS:**
- Small amount of nodes (few hundreds)
- Fake Networks
The typical Docker scenario

Host

Docker managed
linux bridge

4
1st bottleneck: Resources usage

Security limits
/etc/security/limits.conf

Kernel parameters
/etc/sysctl.conf
1st bottleneck: Resources usage

Security limits
/etc/security/limits.conf

- **nofile**: number of open files
- **nproc**: maximum number of processes

Kernel parameters
/etc/sysctl.conf
1st bottleneck: Resources usage

Security limits
/etc/security/limits.conf

- **nofile**  number of open files
- **nproc**  maximum number of processes

Kernel parameters
/etc/sysctl.conf

- **pty**  maximum number of pseudo-terminal  def:4096
- **gc_thresh1**  garbage collector ARP entries  def:128
- **gc_thresh2**  garbage collector ARP entries  def:512
- **gc_thresh3**  garbage collector ARP entries  def:1024
1\textsuperscript{st} bottleneck: Resources usage

Security limits
/etc/security/limits.conf

\begin{align*}
\text{nofile} &\quad \text{number of open files} \\
\text{nproc} &\quad \text{maximum number of processes}
\end{align*}

Kernel parameters
/etc/sysctl.conf

\begin{align*}
\text{pty} &\quad \text{maximum number of pseudo-terminal} & \text{def}:4096 \\
\text{gc\_thresh1} &\quad \text{garbage collector ARP entries} & \text{def}:128 \\
\text{gc\_thresh2} &\quad \text{garbage collector ARP entries} & \text{def}:512 \\
\text{gc\_thresh3} &\quad \text{garbage collector ARP entries} & \text{def}:1024
\end{align*}
2nd bottleneck: The Bridge

Default Linux Bridge

\[ 2^{10} = 1024 \text{ ports} \]
2nd bottleneck: The Bridge

Default Linux Bridge

\[2^{10} = 1024\text{ ports}\]

Our Linux Bridge

\[2^{17} = 131,072\text{ ports}\]
2nd bottleneck: The Bridge

Default Linux Bridge

\[ 2^{10} = 1024 \text{ ports} \]

Our Linux Bridge

\[ 2^{17} = 131,072 \text{ ports} \]

Kernel

```c
#define BR_HASH_SIZE (1 << BR_HASH_BITS)
#define BR_HOLD_TIME (1*HZ)
#define BR_PORT_BITS 10
#define BR_MAX_PORTS (1<<BR_PORT_BITS)
```
2nd bottleneck: The Bridge

Default Linux Bridge

\[2^{10} = 1024 \text{ ports}\]

Our Linux Bridge

\[2^{17} = 131,072 \text{ ports}\]

Kernel

```c
#define BR_HASH_SIZE (1 << BR_HASH_BITS)
#define BR_HOLD_TIME (1*HZ)
#define BR_PORT_BITS 10
#define BR_MAX_PORTS (1<<BR_PORT_BITS)
```
3rd bottleneck: ARP Broadcast
3rd bottleneck: ARP Broadcast
3rd bottleneck: ARP Broadcast

- **Resources**
- **Bridge**
- **ARP broadcast**

**eth0** -> Resources -> Bridge -> ARP broadcast -> User

**IP to contact**

**MAC address?**

**ARP**
3rd bottleneck: ARP Broadcast
3rd bottleneck: ARP Broadcast

Resources

Bridge

ARP broadcast

eth0

ARP

IP to contact

MAC address?

ARP cache

Kernel

User
3rd bottleneck: ARP Broadcast

- Resources
- Bridge
- ARP broadcast

- IP to contact
- ARP cache
- Broadcast to find MAC
- MAC address?
- ARP
- eth0
- Kernel
- User
3rd bottleneck: ARP Broadcast
3rd bottleneck: ARP Broadcast

Neighbour Unreachability Detection (NUD): Reachable = Valid entry recently used
3rd bottleneck: ARP Broadcast

3rd bottleneck: ARP Broadcast

3\textsuperscript{rd} bottleneck: ARP Broadcast

Usage: `autoarpd <RULE> [ interfaces ]`  
E.g.: `autoarpd 02:42:ip1:ip2:ip3:ip4 eth0`

**AutoArpd**: https://gitlab.com/uniroma3/compunet/networks/AutoARPD
**3rd bottleneck: ARP Broadcast**

**3rd bottleneck: ARP Broadcast**

---

3rd bottleneck: ARP Broadcast

**AutoArpd.** https://gitlab.com/uniroma3/compunet/networks/AutoARPD
4th bottleneck: Emulating Realistic Internet Delays

- Resources
- Bridge
- ARP broadcast
- Delays

Delay:
- 250 ms
- 70 ms
- 10 ms

World Map with delays marked on connections.
4th bottleneck: Emulating Realistic Internet Delays

Rice University, Internet delay space synthesizer [https://www.cs.rice.edu/~eugeneng/research/ds2/](https://www.cs.rice.edu/~eugeneng/research/ds2/)
4th bottleneck: Emulating Realistic Internet Delays

Resources → Bridge → ARP broadcast → Delays

User Application

Linux Kernel

IP Stack

Level 2

Ingress qdisc

Egress qdisc (queuing discipline)

interface eth0

FIFO
4th bottleneck: Emulating Realistic Internet Delays
4th bottleneck: Emulating Realistic Internet Delays

Resources
Bridge
ARP broadcast
Delays

TrafficControl (TC)
- qdiscs structure
4th bottleneck: Emulating Realistic Internet Delays

Resources → Bridge → ARP broadcast → Delays

- TrafficControl (TC)
  - qdiscs structure
  - qdiscs types

netem:
- delay=10 ms
- 20 ms
- 30 ms
- 50 ms
- 150 ms
- 270 ms

- delay=50 ms
- delay=20 ms
- delay=30 ms
- delay=150 ms
- delay=270 ms
4th bottleneck: Emulating Realistic Internet Delays

Resources → Bridge → ARP broadcast → Delays

*netem*: delay=10 ms
20 ms  30 ms  50 ms  150 ms  270 ms

**TrafficControl (TC)**
- qdiscs structure
- qdiscs types

**NFTables (NFT)**
(a new version of ip tables)
- packets coloring
The 4th bottleneck: Emulating Realistic Internet Delays

TrafficControl (TC)
- qdiscs structure
- qdiscs types
- packets filters

NFTables (NFT)
(a new version of ip tables)
- packets coloring

Resources
Bridge
ARP broadcast
Delays

Netem:
delay=10 ms
20 ms 30 ms 50 ms 150 ms 270 ms

Qdiscs types
- packets coloring
- filters by ip destination

TrafficControl (TC)
4th bottleneck: Emulating Realistic Internet Delays

- Netem:
  - delay = 10 ms
- TrafficControl (TC)
  - qdiscs structure
  - qdiscs types
  - packets filters
- NFTables (NFT)
  - packets coloring (a new version of ip tables)

Resources → Bridge → ARP broadcast → Delays
5th bottleneck: CPUs workload

CPUs workload
5th bottleneck: CPUs workload

Time inflation

Resources → Bridge → ARP broadcast → Delays → CPUs
5th bottleneck: CPUs workload

- Network delay

Time inflation
$5^{th}$ bottleneck: CPUs workload

- Network delay
- Blockchain software

Time inflation
5th bottleneck: CPUs workload

- Network delay
- Blockchain software
- Transaction load

Time inflation
5th bottleneck: CPUs workload

- Network delay
- Blockchain software
- Transaction load
- Kernel TCP protocol
5th bottleneck: CPUs workload

- Network delay
- Blockchain software
- Transaction load
- Kernel TCP protocol

Time inflation

Resources → Bridge → ARP broadcast → Delays → CPUs
5th bottleneck: CPUs workload

Resources
Bridge
ARP broadcast
Delays
CPUs

packet_1
ACK_1
packet_2

 packet_1

 packet_2

ACK 1
A
B

CPUs

Resources
Bridge
ARP broadcast
Delays


...
5th bottleneck: CPUs workload
5th bottleneck: CPUs workload

Berkeley Packet Filter (BPF) code

```
#include <net/tcp.h>

2173 static inline u32 tcp_timeout_init(struct sock *sk)
2174 {
2175     int timeout;
2176     timeout = tcp_call_bpf(sk, BPF_SOCK_OPS_TIMEOUT_INIT, 0, NULL);
2177     if (timeout <= 0)
2178         timeout = TCP_TIMEOUT_INIT;
2179     return timeout;
2180 }
```
5th bottleneck: CPUs workload

Berkeley Packet Filter (BPF) code

```c
#include <net/tcp.h>

static inline u32 tcp_timeout_init(struct sock *sk)
{
    int timeout;

    timeout = tcp_call_bpf(sk, BPF_sock_ops_timeout_init, 0, NULL);
    if (timeout <= 0)
        timeout = TCP_TIMEOUT_INIT;
    return timeout;
}
```
Bottleneck: CPUs workload

Berkeley Packet Filter (BPF) code

```
#include <net/tcp.h>

static inline u32 tcp_timeout_init(struct sock *sk)
{
    int timeout;
    timeout = tcp_call_bpf(sk, BPF_SOCK_OPS_TIMEOUT_INIT, 0, NULL);
    if (timeout <= 0)
        timeout = TCP_TIMEOUT_INIT;
    return timeout;
}
```

```c
int set_initial_rto(struct bpf_sock_ops *skops)
{
    const int timeout = 3; // initial RTO timeout in seconds
    const int hz = 250; // this value has to match the Hz value of the system
    int op = (int) skops->op;
    if (op != BPF_SOCK_OPS_TIMEOUT_INIT)
        skops->reply = timeout * hz;
    return 1;
}
```
**5th bottleneck: CPUs workload**

![Diagram showing the flow of data through CPUs, Resources, Bridge, ARP broadcast, Delays, and CPUs.]

### Berkeley Packet Filter (BPF) code

```c
#include <net/sock.h>

static inline u32 tcp_timeout_init(struct sock *sk) {
    int timeout;
    timeout = tcp_call_bpf(sk, BPF_SOCKOPS_TIMEOUT_INIT, 0, NULL);
    if (timeout <= 0)
        timeout = TCP_TIMEOUT_INIT;
    return timeout;
}

int set_initial_rto(struct bpf_sock_ops *skops) {
    const int timeout = 3; // initial RTD timeout in seconds
    const int hz = 250;  // this value has to match the hz value of the system
    int op = (int) skops->op;
    if (op == BPF_SOCKOPS_TIMEOUT_INIT) {
        skops->reply = timeout * hz;
        return 1;
    }
    return 1;
}
```

Our Solution

CPUs
Resources
Bridge
ARP broadcast
Delays
CPUs
OUR Emulation:

- Huge amount of nodes → 3500 containers in 400GB RAM
- “Real” Network env. → end-to-end realistic internet delays, 8000 TCP-based and 64000 UDP-based connections
- Simple to handle → Makefile and Python scripts
- Simple to modify → Python scripts
Future works:

- Simplify the setup
- Multiple host (kubernetes)
- Real software of a blockchain node
- Create a library to create transaction load
- Create a library to support data gathering
The End

Thank you!

diego.pennino@unitus.it, diego.pennino@uniroma3.it, pizzonia@ing.uniroma3.it