WrapSix

How To Build It, How To Run It

Michal Zima

20 October 2012

What is this talk about

- What is WrapSix
- Performance issues of specialised network apps
- How to optimize for high performance

What is this talk about

- Brief introduction to NAT64 and DNS64
- DNS64 implementations and configuration
- Running WrapSix

What is WrapSix

- Open-source NAT64
- Userspace implementation
- Alternatives: Ecdysis, Tayga

What I wanted it to be

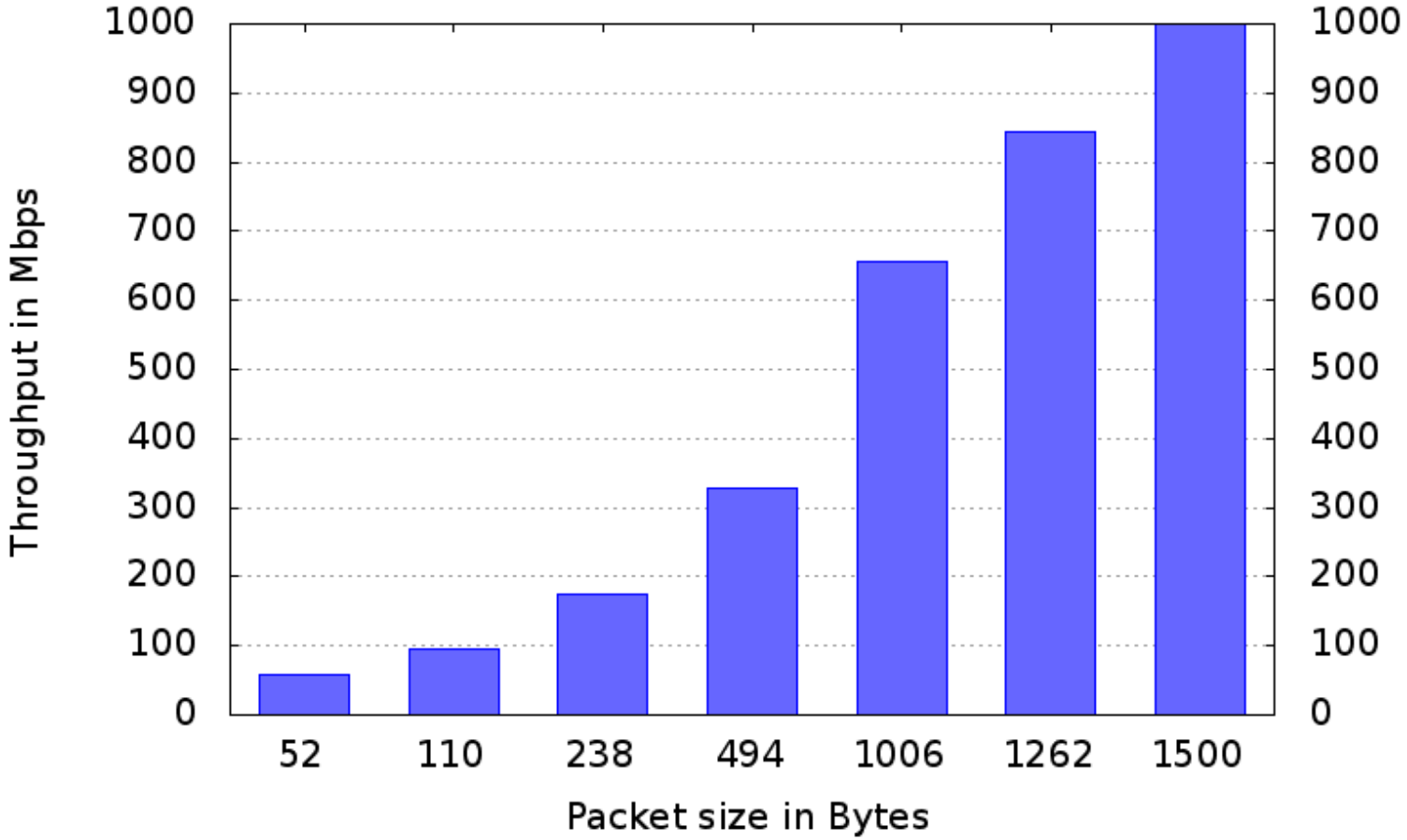
- Implementation for **production**
  - handling high-speed traffic
  - causing minimal latency to it

Bitrate vs. Packets count

- Is 1Gbps througput enough?
- Two ways of understanding "high throughput":
  - 1 Gbps = ∼83k packets, each 1500 B
  - 1 Gbps = ∼2.4M packets, each 52 B
- *"TAYGA is fast – can saturate gigabit Ethernet on modest PC hardware"*

— http://www.litech.org/tayga/

Throughput of Tayga



Maximise pps

- 83k vs. 2.4M: difference is in ∼2.3M calling of processing routine every second!
- But not only in the application – in kernel too!

Use more hardware

- I.e. parallelization
- Sharing state information
- NAT64 requires preserving order of packets of the same flow
- Easier for distinguishable packet flows, e.g. *IPv4/IPv6*, TCP/UDP/ICMP etc.

Optimise the processing

- Handling the same amount of packets with less power
- No race condition bugs
- Guarantees higher performance (parallelization doesn't!)
- When limits hit, we can parallelize it too

1. Code cleaning

- Get rid of useless code – every instruction counts
- Precompute data if possible
- Optimise jumps
  - unpredicted jumps break CPU pipeline
  - if-else: `if` should be always the more probable variant
  - cycles!

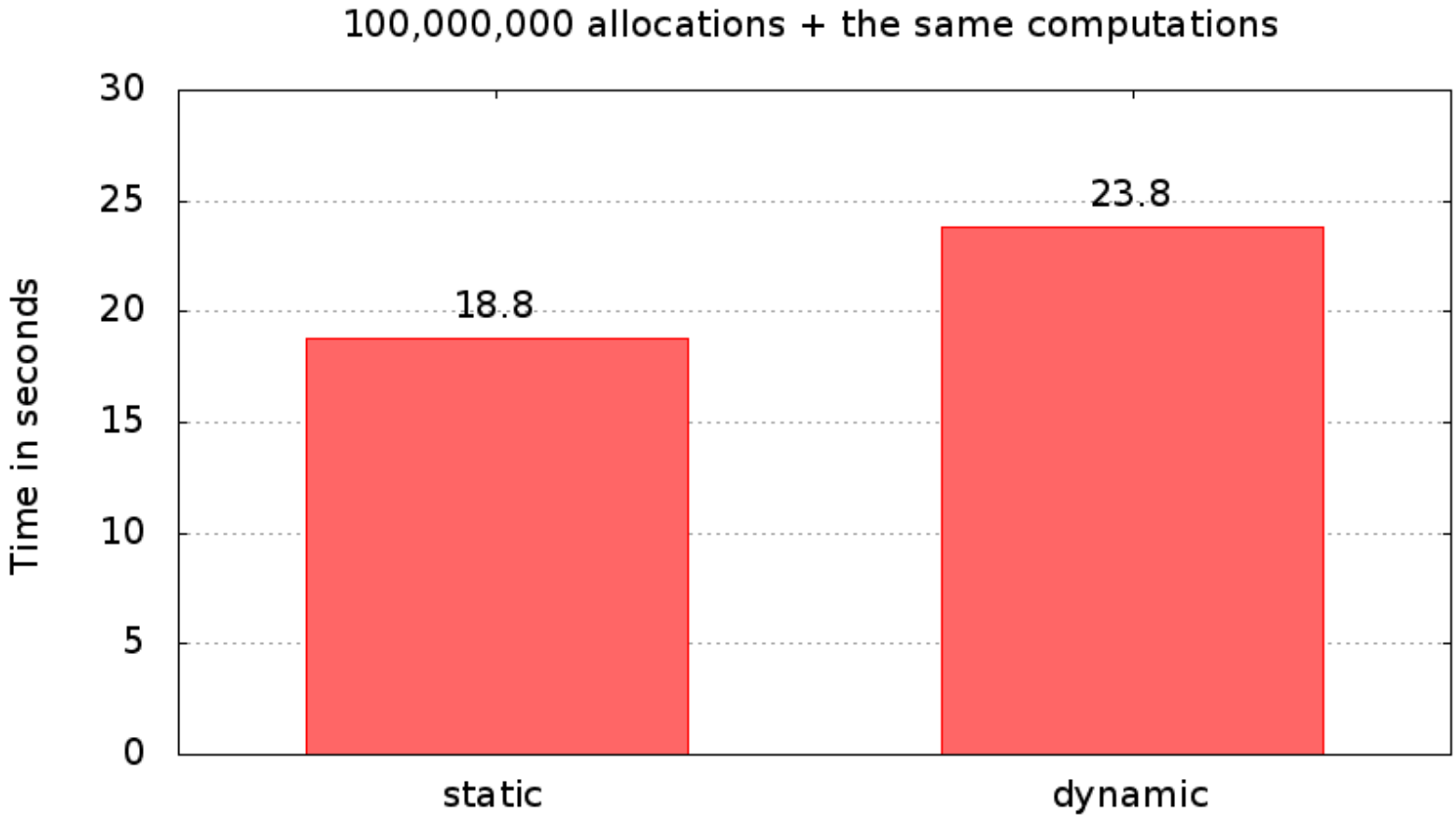
2. Memory work optimization

- Main memory (RAM) is **not** fast!
  - Access to RAM: hundreds of CPU cycles
  - Access to L3/L2/L1 cache: tens/∼10/∼4 CPU cycles
  - Access to registers: no delay
- Do as much work as you can while the data you need is in the cache

3. Memory allocation

- Dynamic allocation is slow
- `malloc` is not the only available allocator

Dynamic vs. static allocation



Impact on throughput

- We want to get closer to 2.4M pps
- From benchmark: difference is $\sim$5 s
- 1 dynamic allocation per packet $\Rightarrow$ + $\sim$0.12 s for 2.4M packets
- But there are usually several allocations per packet!

4. Efficient data structures

- Bad structure can cause significant slow down
- NAT64 stores mainly state information
  $\rightarrow$ connection lookup
  $\rightarrow$ fragments lookup
  $\rightarrow$ expiration of connections

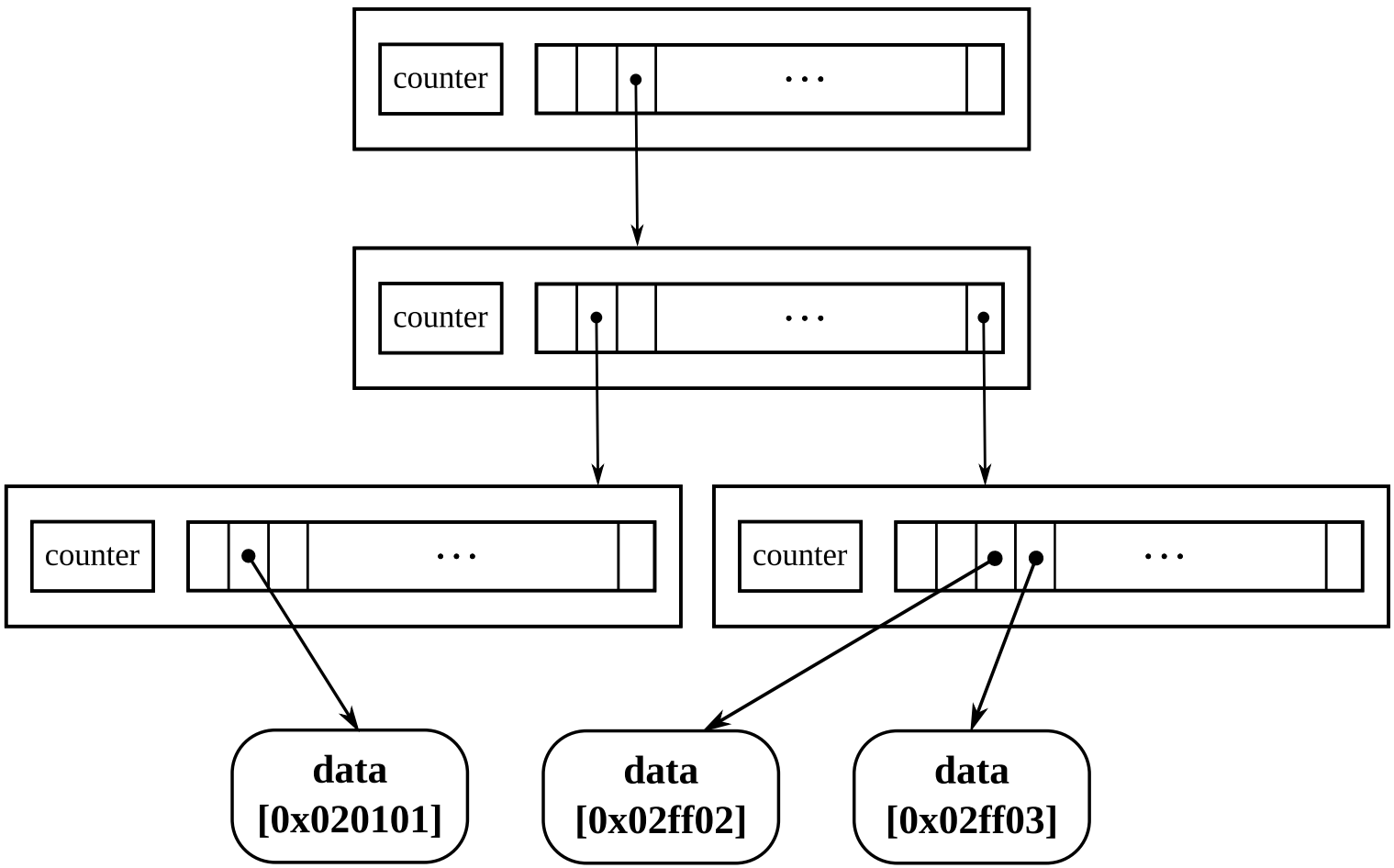
Hash table

- Potentially *very fast* – needs good hash function
- NAT64 may store from few entries to tens/hundreds of thousands entries
- Dynamic resizing
- Secondary storage for colliding entries

Radix tree

- Tree of fixed height – depending on key length
- Chain of arrays with key chunks as indexes
- Universal use in NAT64
- Allows "partial lookups"
- Fast with thousands entries just like with few

Radix tree


counter
counter
counter
counter
data
[0x020101]
data
[0x02ff02]
data
[0x02ff03]

Linked list

- Ideal for queues
- In WrapSix used to look for expired connections
- Faster than traversing any tree or hash table

OS tuning

- Set CPU affinity of the application
- If threaded then try putting each on different core
- Different applications need different type of distribution
- Give them the highest priority

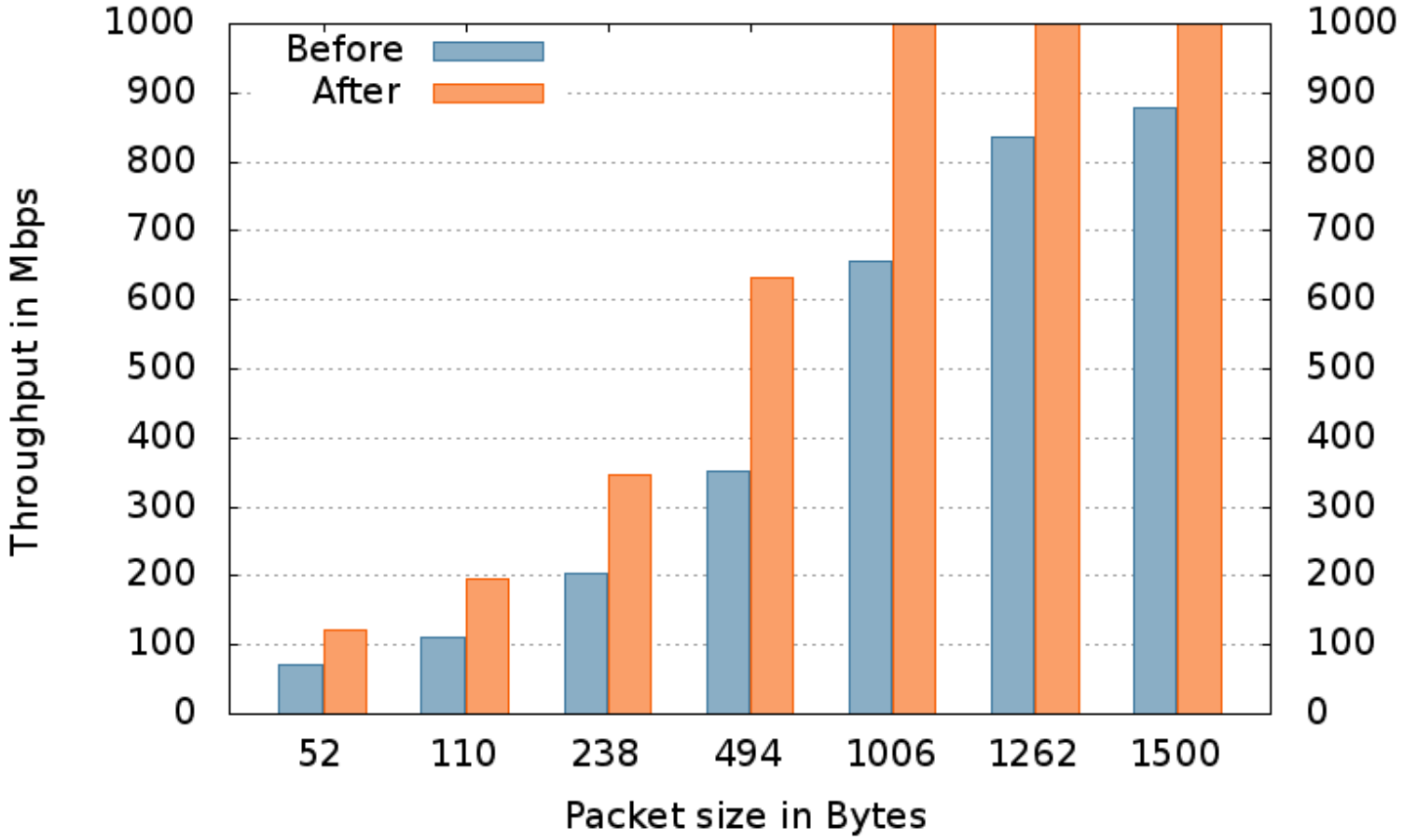
HW tuning: less is more

- More cores than threads $\Rightarrow$ turn off Hyper Threading
- Reduce number of CPUs

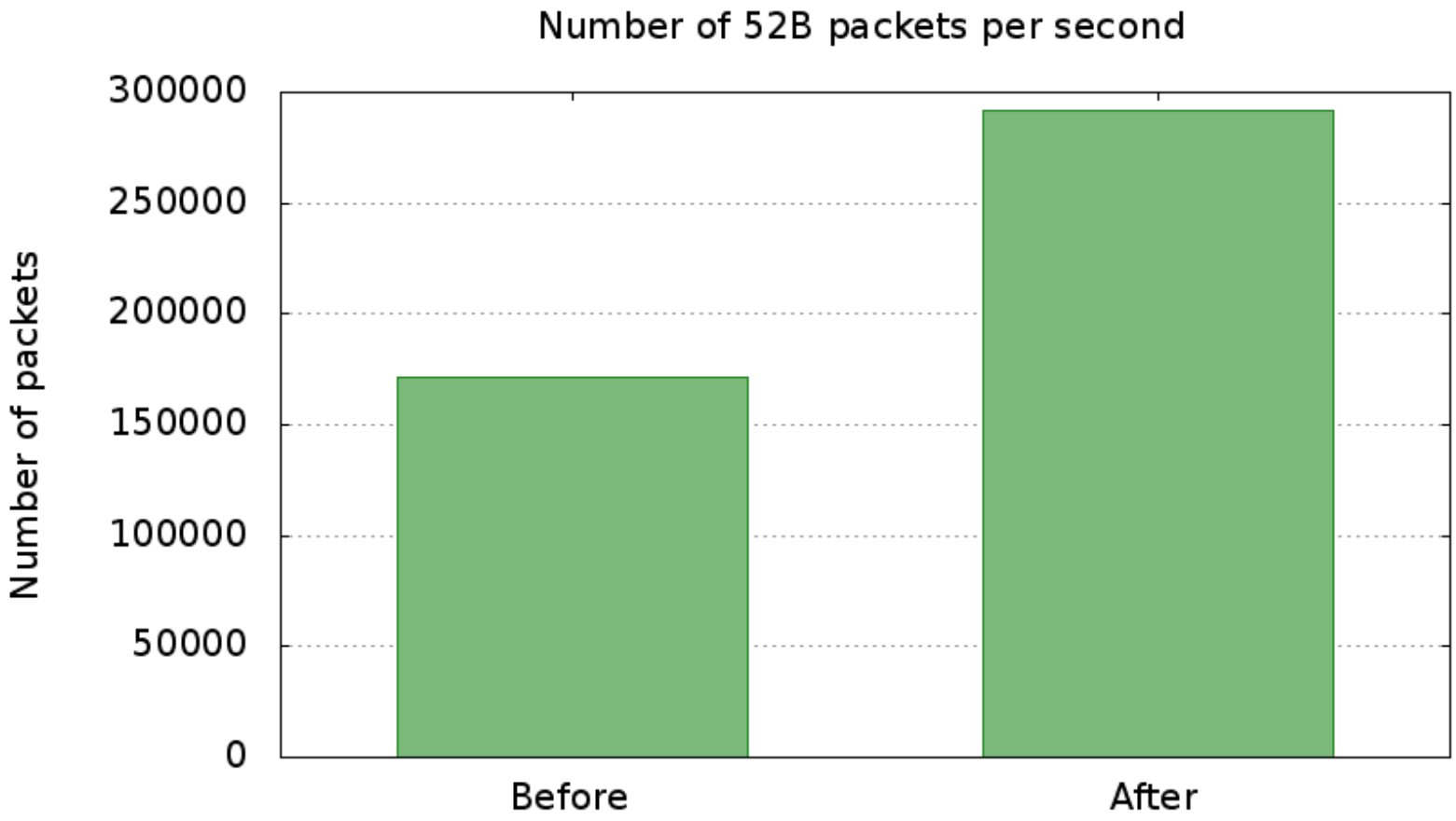
HW tuning – example

- Application runs on 1 CPU
- OS processes packets from network card on the second one and forwards them to the other CPU

Optimizations in WrapSix

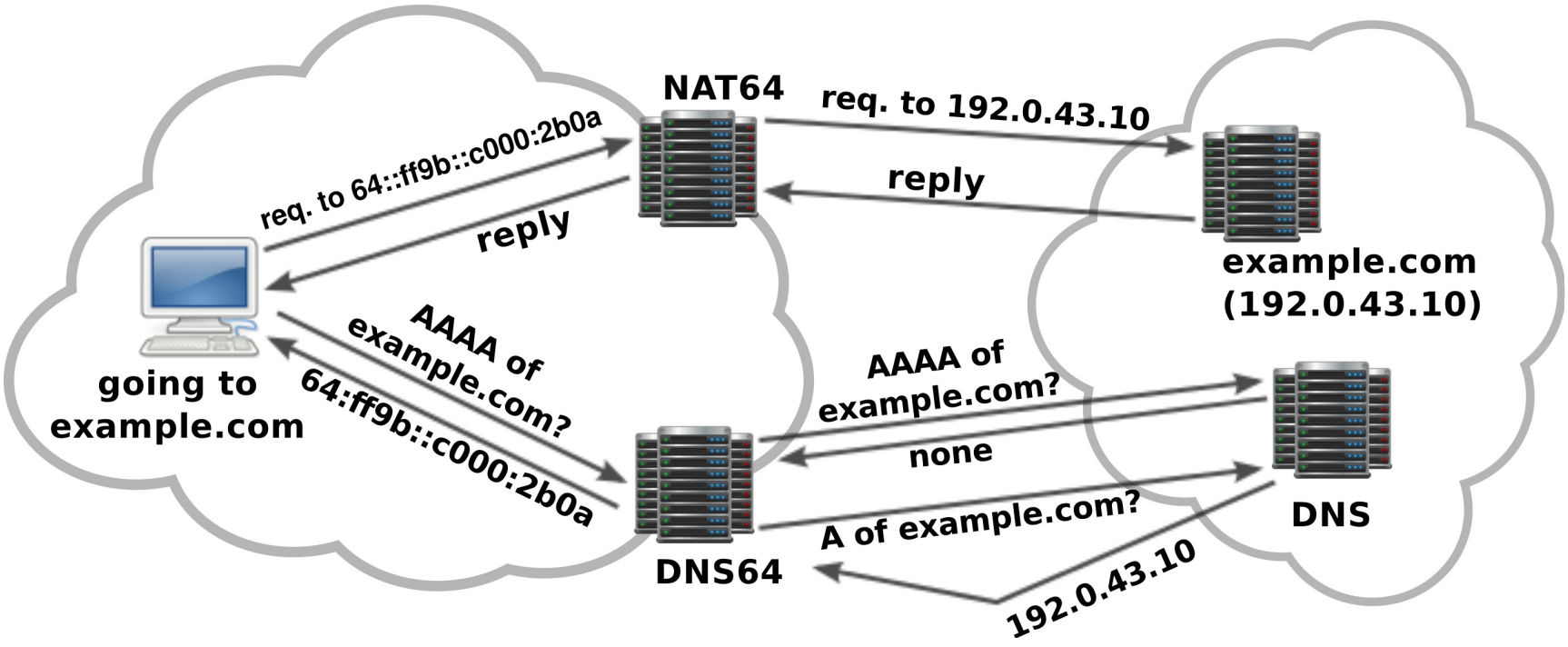


Optimizations in WrapSix



Running WrapSix

What is NAT64



1. DNS64

- BIND 9.8.0+ supports it natively
- For Unbound 1.4.7 exists a patch
  - but 1.4.7 is quite old

BIND configuration

```
options {
        listen-on { none; };
        listen-on-v6 { any; };
        recursion yes;
        dns64 64:ff9b::/96 {};
};
```

Testing DNS64

```
$ dig @fdcb::d64 linuxdays.cz AAAA +short
64:ff9b::d9aa:6312
```

2. NAT64

- Download WrapSix from *www.wrapsix.org*
- Modify configuration
- `./configure && make && make install`
- Start WrapSix with `wrapsix-wrapper`
- Be sure to disable IPv6 forwarding

Configuring WrapSix

- Edit `src/wrapper.c` and `src/wrapper.h`
- Configure:
  - Interface
  - IPv4 address – unassigned one!
  - NAT64 prefix
  - IPv6 MTU

Testing NAT64

```
$ ping6 -c 2 linuxdays.cz
PING linuxdays.cz(64:ff9b::d9aa:6312) 56 data bytes
64 bytes from 64:ff9b::d9aa:6312: icmp_seq=1 ttl=53 time=11.8 ms
64 bytes from 64:ff9b::d9aa:6312: icmp_seq=2 ttl=53 time=11.5 ms

--- linuxdays.cz ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 11.545/11.699/11.854/0.188 ms
```

Questions?