Benchmarking and profiling DNS systems with modern Linux tools

Robert Edmonds (edmonds@fsi.io)
Farsight Security, Inc.

Intro

- Demonstrate Linux tools useful for DNS benchmarking
  - trafgen, ifpps (from the netsniff-ng toolkit)
  - tc (“traffic control”)
  - perf_events
  - mpstat
- Use output to generate graphs
- Profile hotspots revealed by benchmarking

Hardware

- Entry level hardware, two years old
- Receiver:
  - Intel Xeon E3-1245v3 (3.4 GHz, quad core)
- Sender:
  - Intel Core i3-4130 (3.4 GHz, dual core)
- No HyperThreading, SpeedStep, Turbo Boost
- Intel Ethernet Server Adapter I350-T2 (PCIe cards)
  - Dedicated 1 Gbit/s link, directly connected

Software

- Linux kernel 4.2
  - Defaults, no attempts to tune networking stack
- DNS servers
  - BIND 9.10.3 (ISC)
  - Knot DNS 2.0.1 (CZ.NIC)
  - Unbound 1.5.4 (NLnet Labs)
  - dns-echo 0.1 (Ray Bellis / ISC)

Software

- Benchmarking tools
  - netsniff-ng toolkit
    - trafgen
    - ifpps
  - tc
  - mpstat
  - perf_events

trafgen (netsniff-ng toolkit)

- High speed userspace network traffic generator
- Uses packet(7) TX_RING interface of the Linux kernel
- Low-level packet description language
- Sample DNS trafgen config on next two slides
- Easily generates ***>1Mq/s*** with two CPU cores

```
# Ethernet header
0xa0, 0x36, 0x9f, 0x28, 0x96, 0x11, # Destination MAC
0xa0, 0x36, 0x9f, 0x28, 0x95, 0xb1, # Source MAC
const16(0x0800),    # Ethernet protocol: IPv4

# IP header
0b01000101,         # IP version (4b), header length (4b)
0,                  # TOS
const16(71),        # Total length
drnd(2),            # Fragment ID (randomized)
0b00000000, 0,      # Flags (3b), fragment offset (13b)
64,                 # IP TTL
17,                 # IP protocol: UDP
csumip(14, 33),     # IP checksum
192, 168, 0, 1,     # Source IP address
192, 168, 0, 53,    # Destination IP address

# UDP header
drnd(2),            # UDP source port (randomized)
const16(53053),     # UDP destination port
const16(51),        # UDP length
const16(0),         # UDP checksum (disabled)
```

```
# DNS header
drnd(2),              # ID (randomized)
const16(0x0100),      # Flags: RD=1
const16(1),           # Question count
const16(0),           # Answer count
const16(0),           # Authority count
const16(1),           # Additional count

# DNS question section
0x03, "www",          #
0x06, "google",       #
0x03, "com",          #
0x00,                 # Question name: "www.google.com."
const16(28),          # Question type: AAAA
const16(1),           # Question class: IN

# DNS additional section
0x00,                 # "."
const16(41),          # OPT
const16(4096),        # UDP payload size
0x00,                 # Extended RCODE
0x00,                 # EDNS version
0x00, 0x00            # Z
const16(0)            # Data length
```

trafgen (netsniff-ng toolkit)

- Send packets on ***iface*** until stopped:

```
# trafgen --qdisc-path -i trafgen.cfg -o ${iface}
```

- This generates >1Mq/s
- Need a way to control the query rate

tc (“traffic control”)

- Linux kernel facility that can shape bits going out an interface
- E.g., allow ***x*** Mbit/sec out ***iface***:

```
# tc qdisc replace dev ${iface} root handle 1: tbf \
    burst 20480 limit 20480 rate ${x}Mbit
```

- For each benchmark run, ramp ***x*** from 25 to 750 Mbps in 25 Mbps increments

tc (“traffic control”)

- tc limits are specified by bandwidth (bits/sec), not packets/sec
- Query messages are identical, packets are all the same size (85 bytes)
- Therefore:
  - 250 Mbit/s → 368 Kq/s
  - 500 Mbit/s → 736 Kq/s
  - 750 Mbit/s → 1.10 Mq/s

ifpps (netsniff-ng toolkit)

- Network interface packet statistics
- Queries /proc, low overhead
- “Top”-like interface or CSV output

mpstat

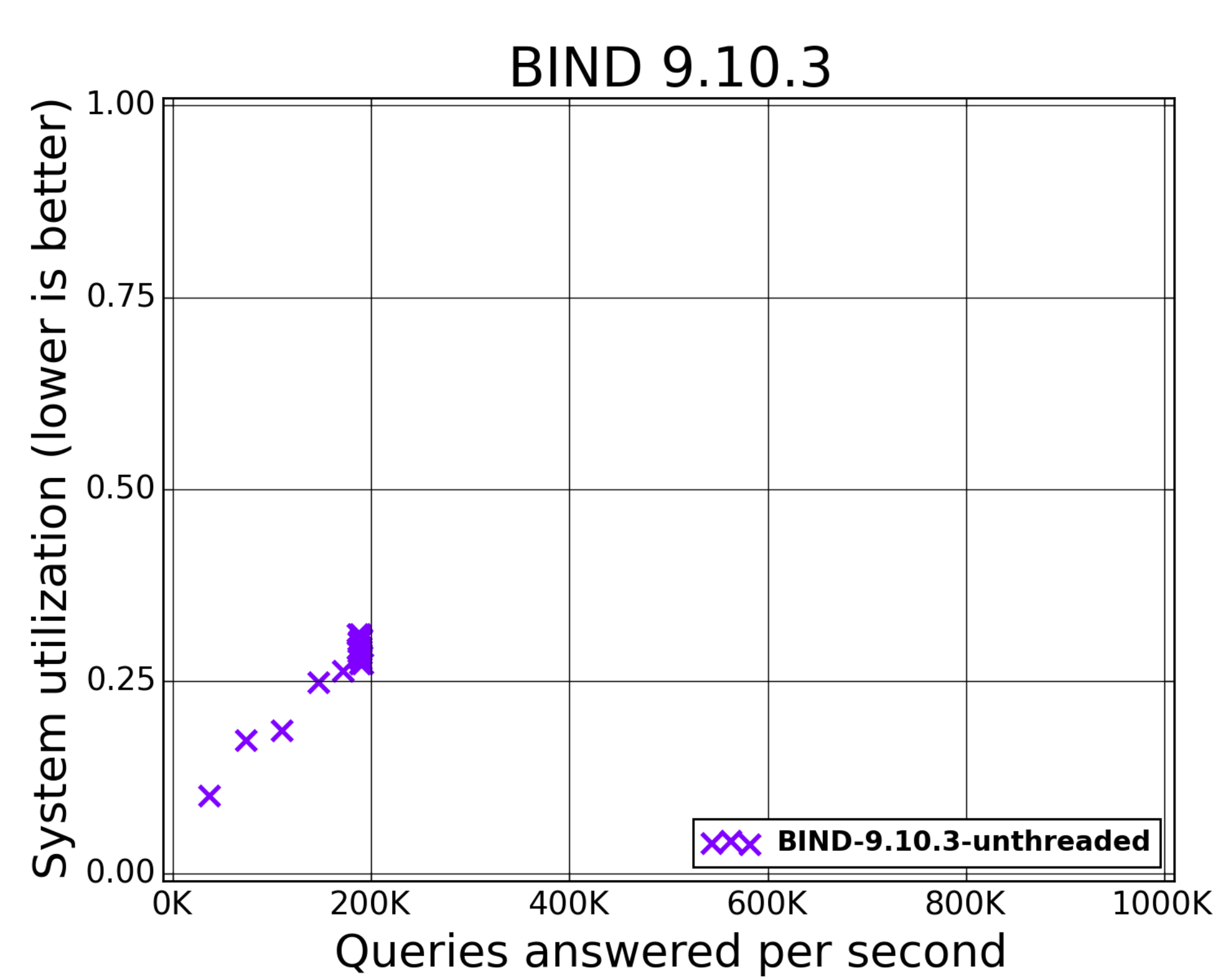
- Record system idle % during the benchmark run
- 100% minus idle % is the overall system load
  - 0% idle → system is 100% loaded

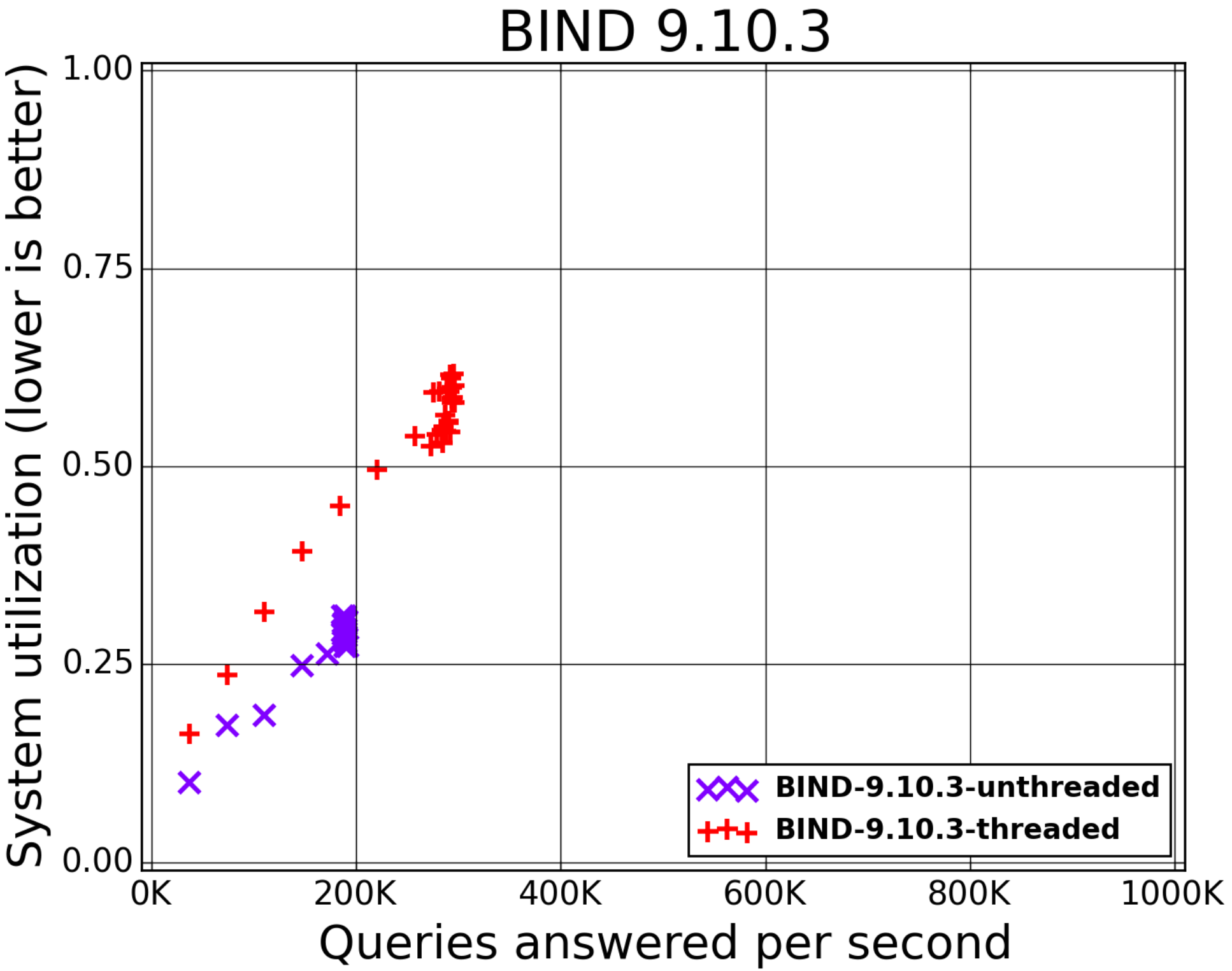
Orchestration

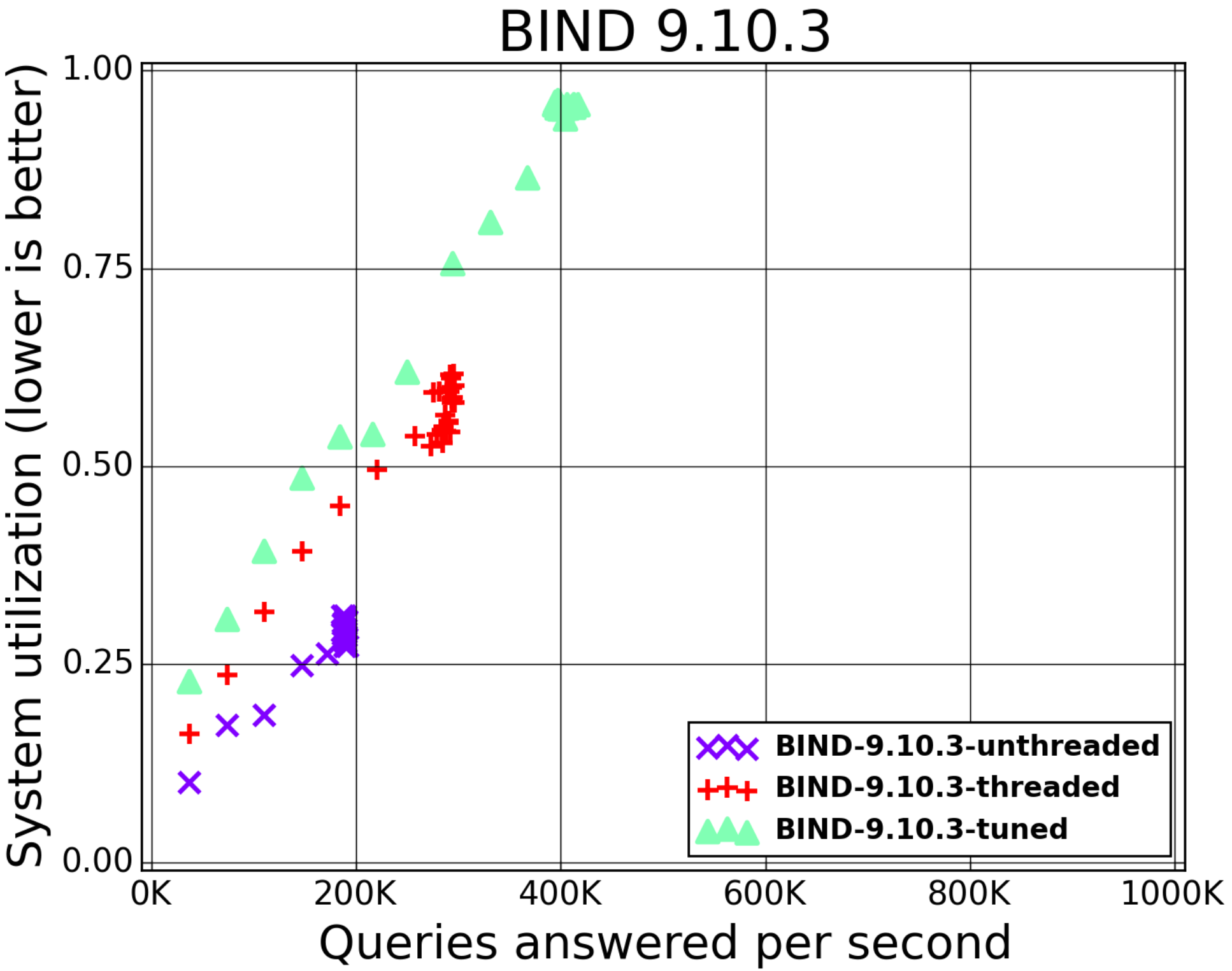
- Home grown Python script:
  - Start server on receiver
  - Clamp sender to 1 Mbit/s (tc)
  - Start trafgen on sender
  - Ramp sender from 25 Mbit/s .. 750 Mbit/s
    - Record network Rx/Tx load (ifpps)
    - Record system load on receiver (mpstat)
  - Stop server, trafgen
- Repeat for each DNS server

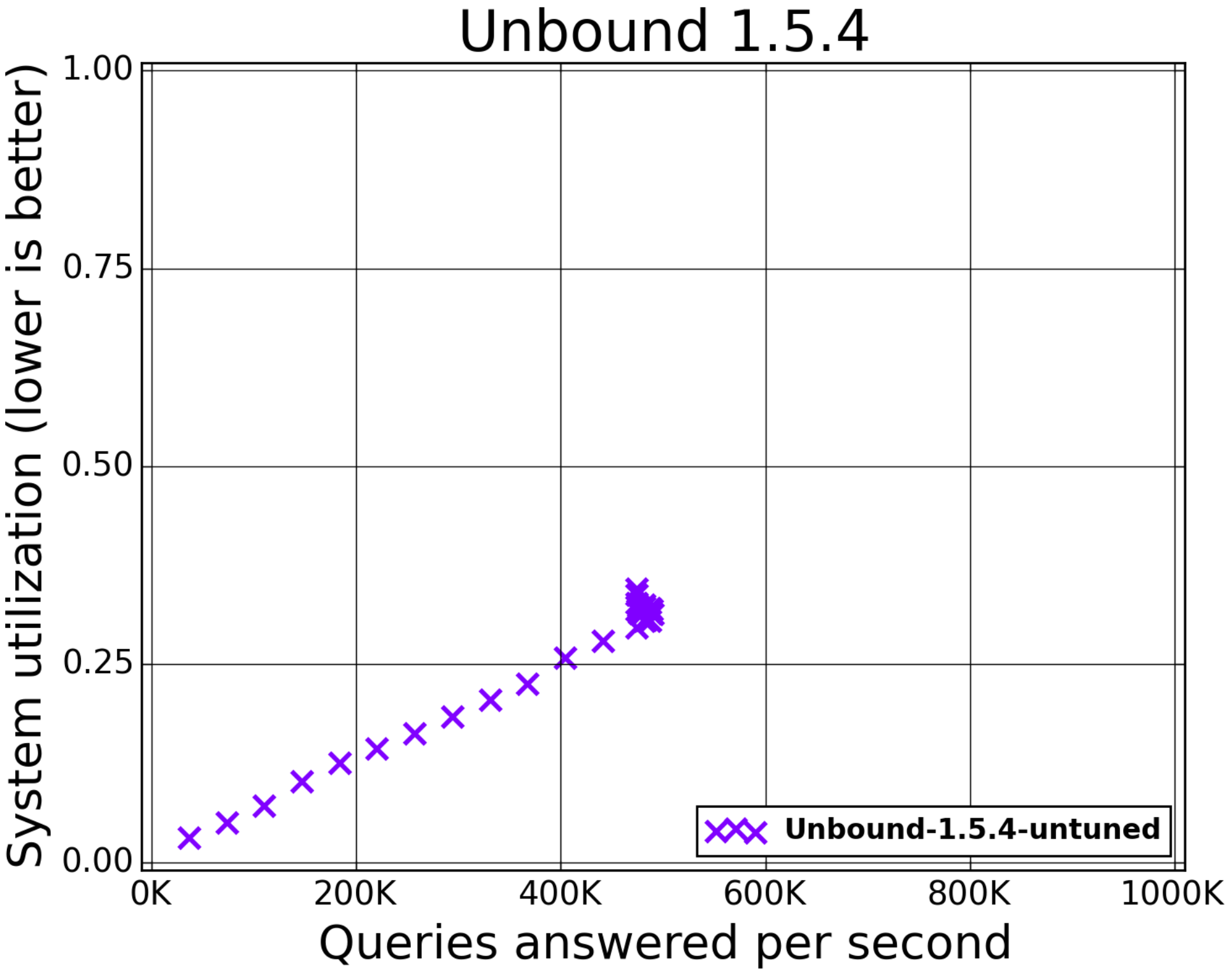
Performance graphs

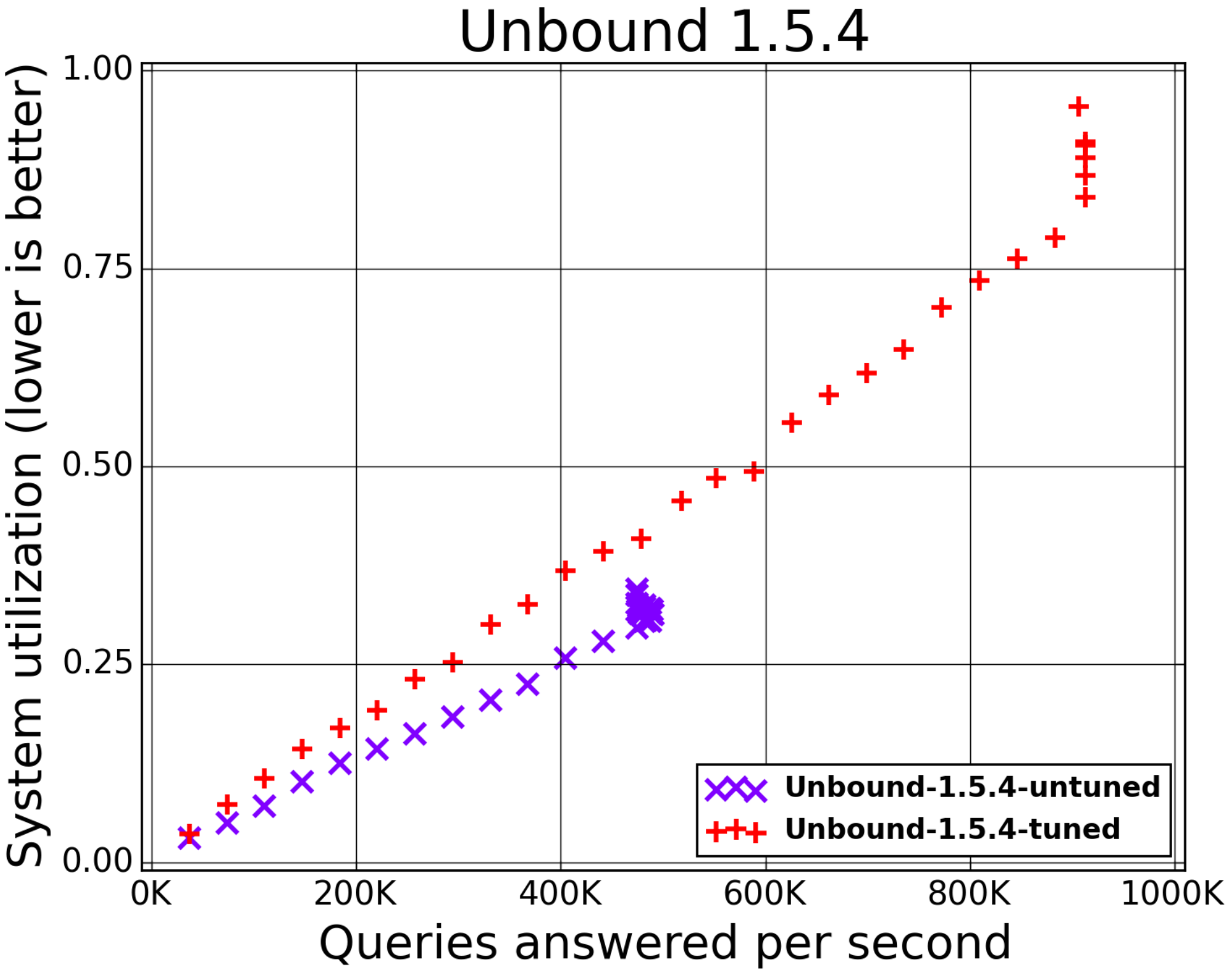
- ***Responses/second*** graphed against ***system load***
  - R/s taken from ifpps
  - System load taken from mpstat
- Want a nice line with low slope and evenly spaced points topping out at 100% system load
- <100% response rate will cause a cluster of points

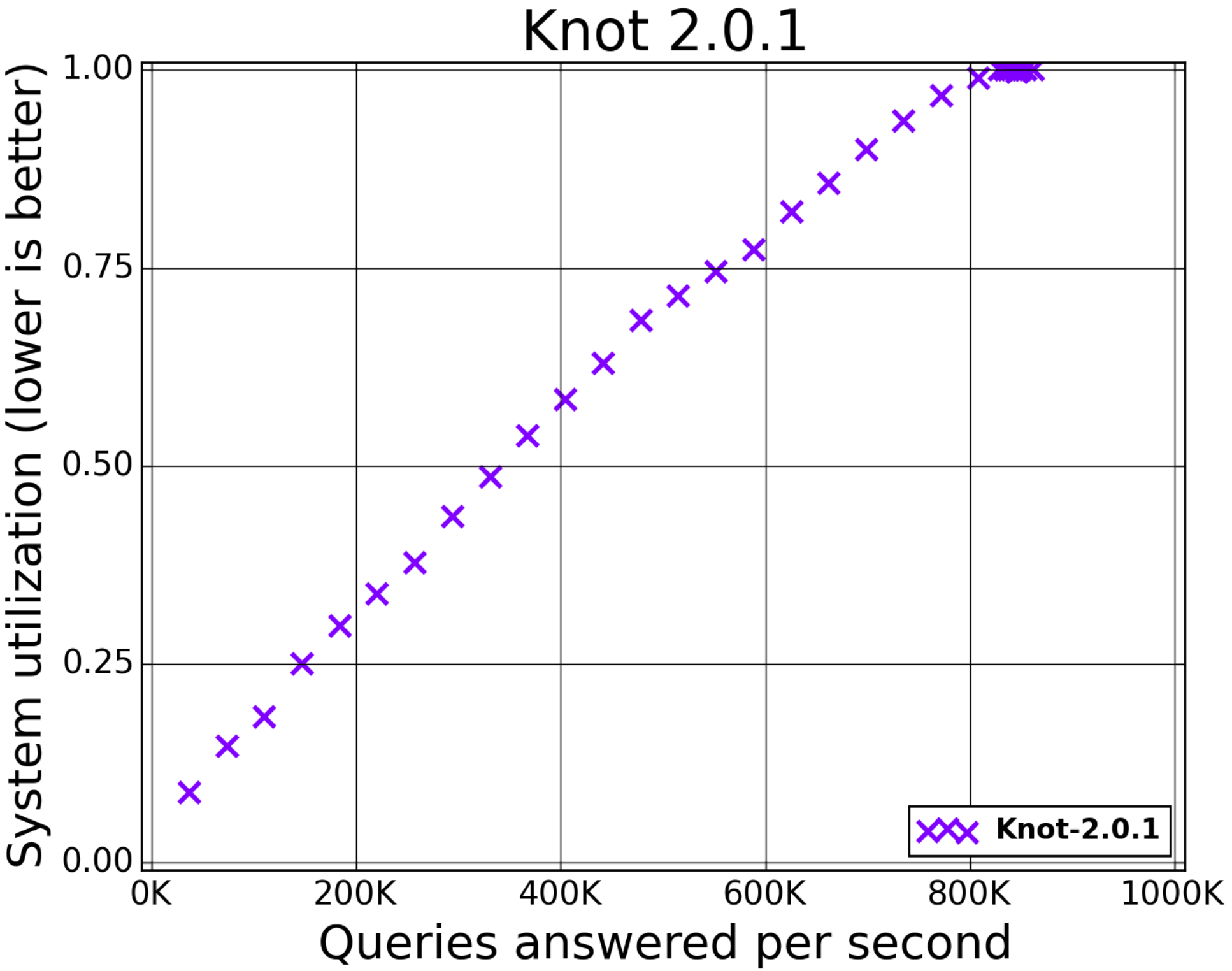

BIND 9.10.3
System utilization (lower is better)
1.00
0.75
0.50
0.25
0.00
0K
200K
400K
600K
800K
1000K
Queries answered per second
BIND-9.10.3-unthreaded

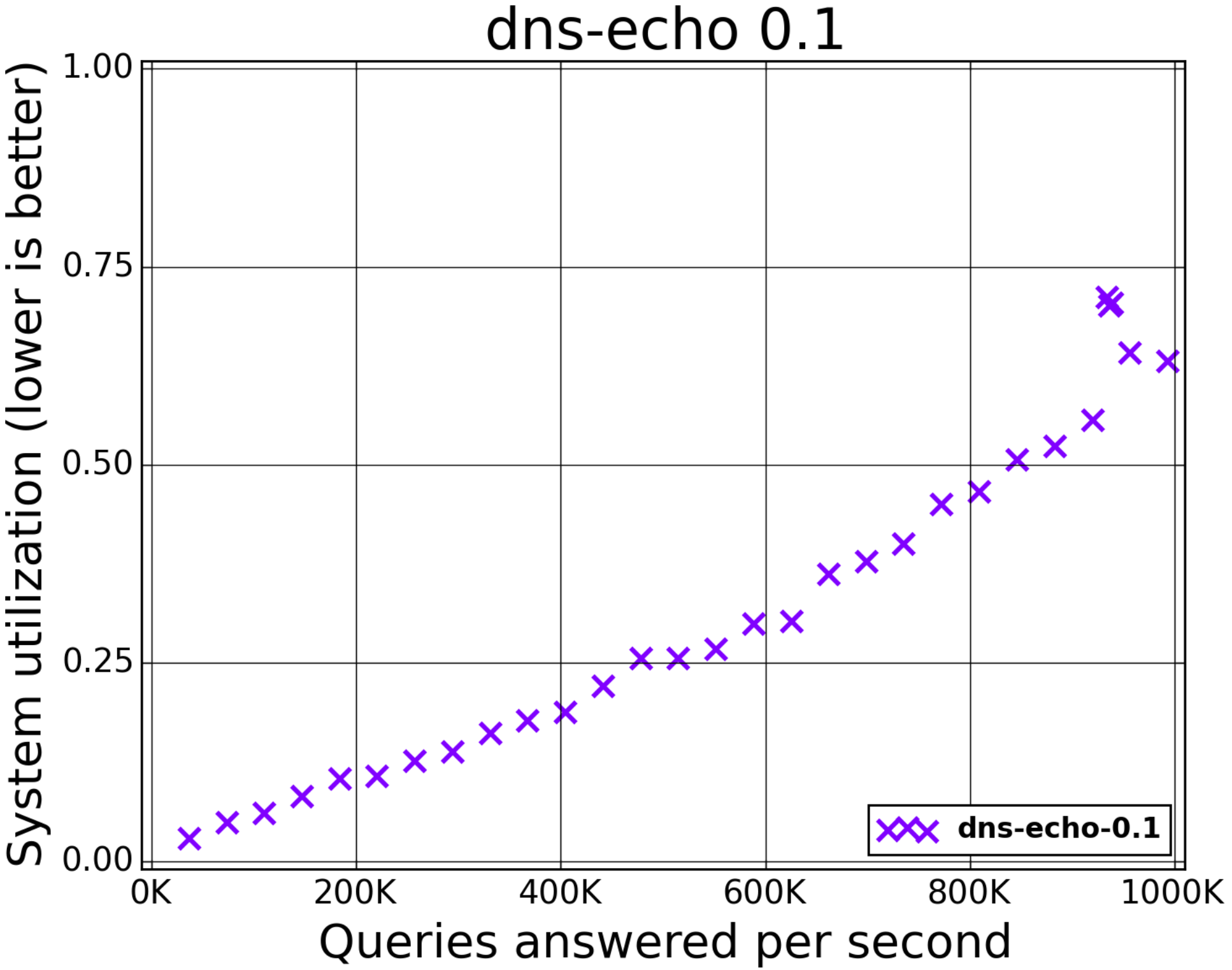

BIND 9.10.3
System utilization (lower is better)
1.00
0.75
0.50
0.25
0.00
0K
200K
400K
600K
800K
1000K
Queries answered per second
BIND-9.10.3-unthreaded
BIND-9.10.3-threaded

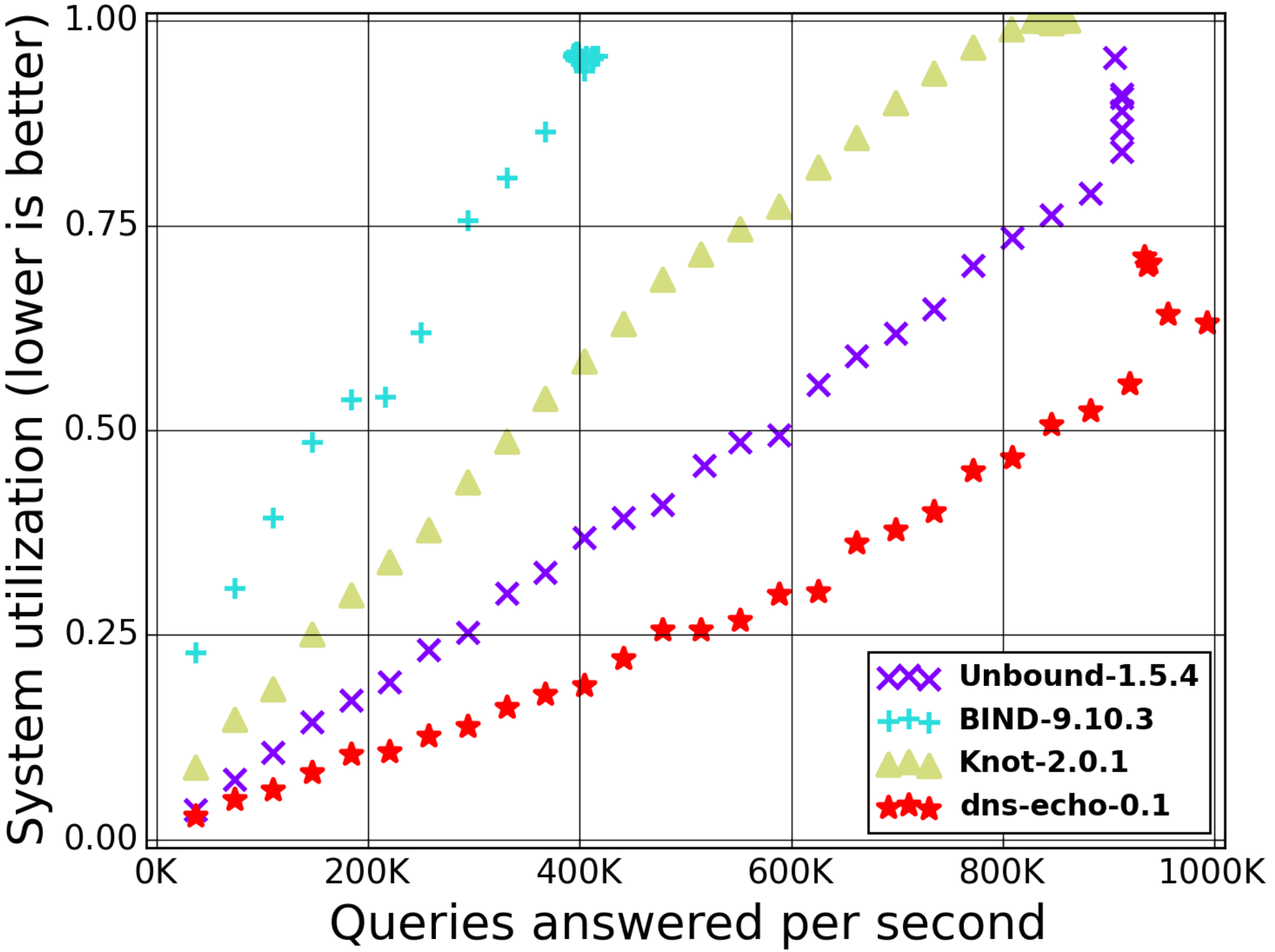

BIND 9.10.3
System utilization (lower is better)
1.00
0.75
0.50
0.25
0.00
0K
200K
400K
600K
800K
1000K
Queries answered per second
BIND-9.10.3-unthreaded
BIND-9.10.3-threaded
BIND-9.10.3-tuned


Unbound 1.5.4
System utilization (lower is better)
1.00
0.75
0.50
0.25
0.00
0K
200K
400K
600K
800K
1000K
Queries answered per second
Unbound-1.5.4-untuned


Unbound 1.5.4
System utilization (lower is better)
1.00
0.75
0.50
0.25
0.00
0K
200K
400K
600K
800K
1000K
Queries answered per second
Unbound-1.5.4-untuned
Unbound-1.5.4-tuned


Knot 2.0.1
System utilization (lower is better)
1.00
0.75
0.50
0.25
0.00
0K
200K
400K
600K
800K
1000K
Queries answered per second
Knot-2.0.1


dns-echo 0.1
System utilization (lower is better)
1.00
0.75
0.50
0.25
0.00
0K
200K
400K
600K
800K
1000K
Queries answered per second
dns-echo-0.1


System utilization (lower is better)
1.00
0.75
0.50
0.25
0.00
0K
200K
400K
600K
800K
1000K
Queries answered per second
Unbound-1.5.4
BIND-9.10.3
Knot-2.0.1
dns-echo-0.1

perf_events

- Kernel-based lightweight profiler
- Profiles the whole system: userspace and kernel
- “perf top”: real-time “top”-like output, but at the level of individual functions
- “perf record”, “perf report”, “perf diff”: take multiple snapshots throughout the benchmark run, diff results
- Call-graph visualization
  - Compile with `-fno-omit-frame-pointer`

perf_events

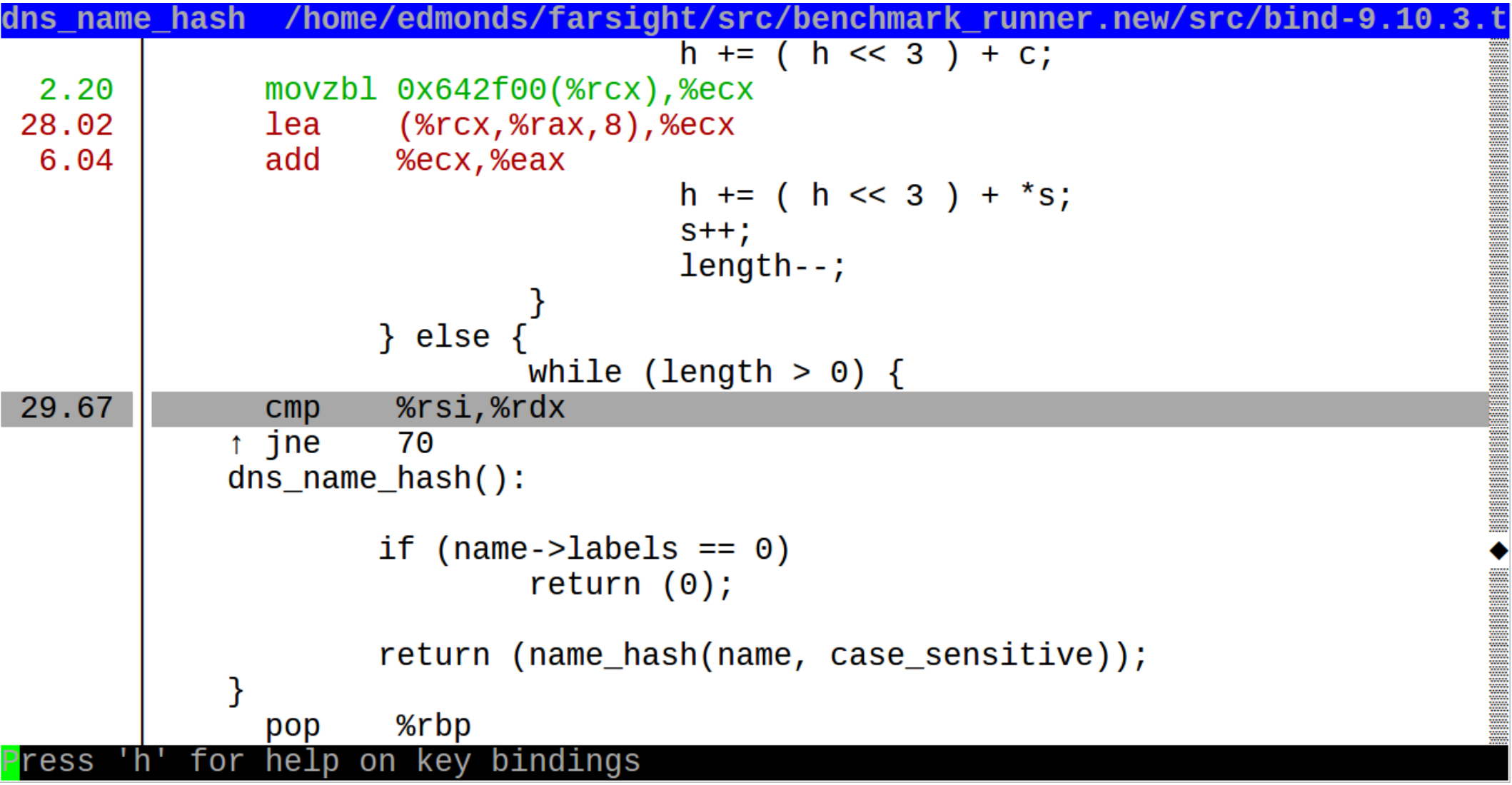
Record a 3 second snapshot, with call-graph data:

```
# perf record --call-graph fp -a -o perf.data sleep 3
```

Start interactive tool to browse the profile:

```
# perf report
```

```
Samples: 35K of event 'cycles', Event count (approx.): 25399929993
  Children      Self  Command          Shared Object                Symbol
-   67.82%     0.00%  named            libpthread-2.19.so           [.] start_t
   - start_thread
      - run
         - 98.57% client_request
            - 76.94% ns_query_start
               - 92.93% query_find
                  - 44.63% ns_client_send
                     - 86.32% client_send
                        - 31.01% dns_message_rendersection
                           - 99.27% dns_rdataset_towiresorted
                              - 95.33% towiresorted.isra.3
                                 - 67.42% dns_name_towire
                                    - 44.60% dns_compress_findglobal
                                       57.17% dns_name_hash
                                       26.86% dns_name_getlabelsequence
                                       10.65% __memcmp_sse4_1
                                       3.19% dns_name_caseequal
                                       1.07% memcmp@plt
                                       1.06% __netif_receive_skb_core
For a higher level overview, try: perf report --sort comm,dso
```


dns_name_hash /home/edmonds/farsight/src/benchmark_runner.new/src/bind-9.10.3.t
h += (h << 3) + c;
2.20 movzbl 0x642f00(%rcx),%ecx
28.02 lea (%rcx,%rax,8),%ecx
6.04 add %ecx,%eax
h += (h << 3) + *s;
s++;
length--;
}
} else {
while (length > 0) {
29.67 cmp %rsi,%rdx
↑ jne 70
dns_name_hash():
if (name->labels == 0)
return (0);
return (name_hash(name, case_sensitive));
}
pop %rbp
Press 'h' for help on key bindings

Thanks!