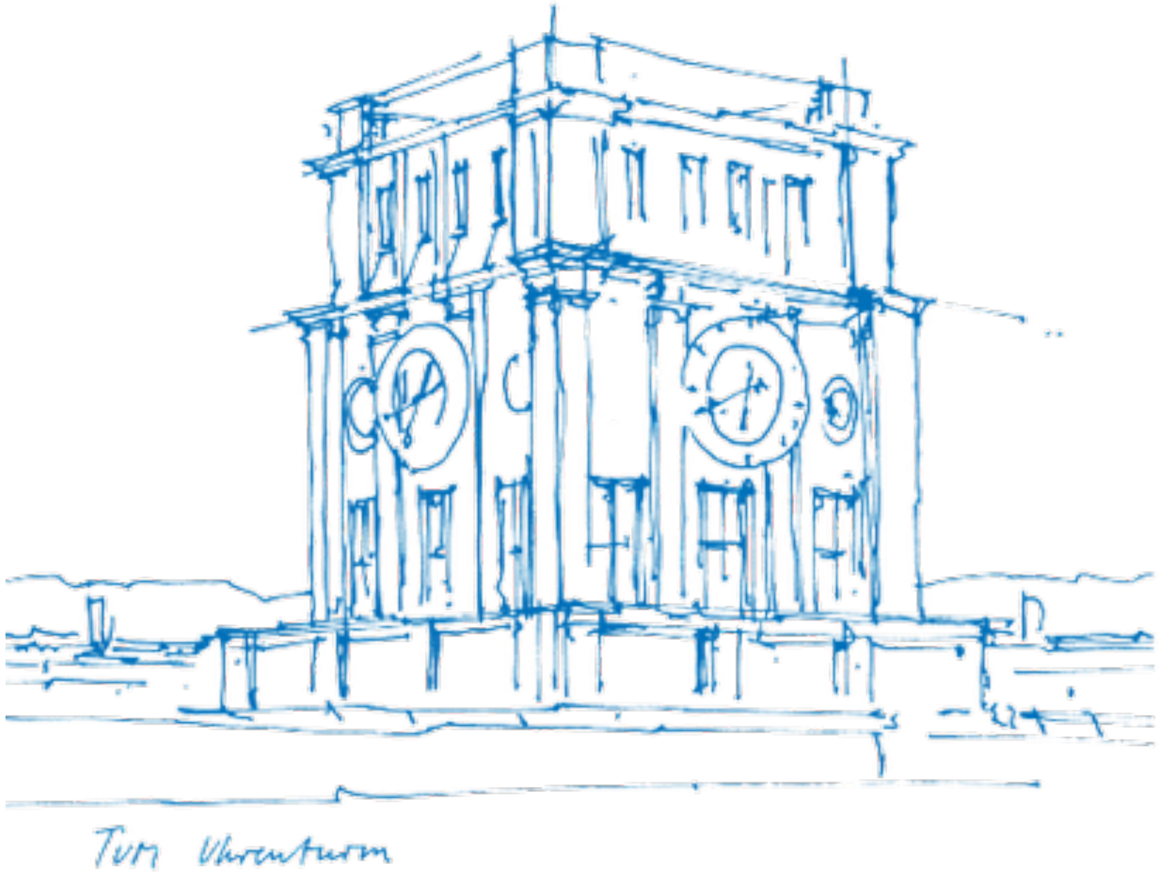
MoonGen: A Fast and Flexible Packet Generator

Paul Emmerich
emmericp@net.in.tum.de
Technical University of Munich
Chair of Network Architectures and Services

IETF-100, 16.11.2017



Research at net.in.tum

- AS56357: Chair of Network Architectures and Services
  - Prof. Dr.-Ing. Georg Carle
  - 5 Post-docs
  - 15 PhD students/research associates
- Broad range of network research topics
  - Traffic measurement and analysis
  - Software-defined networking
  - Security
  - Privacy
  - Peer-to-peer networks
  - IoT
  - Performance analysis and modeling

Performance analysis and modeling

- Packet processing becomes more complex
  - Software-defined networking, network function virtualization, …
- More and more can be done in software nowadays
  - Frameworks like DPDK
  - Complex virtualized network functions, e.g., in 5G
  - Performance impacts unclear

- Research questions
  - What are important performance metrics?
  - How to measure them in a realistic scenario?
  - How to make measurements reproducible?
  - How can performance be predicted with models?

Our testbed

- 15 servers, 36 x 10 Gbit/s ports, 8 x 40 Gbit/s ports
  - NICs from Intel, Mellanox, and Netronome
  - SDN switches/routers
- Fully automated test workflow from a management server
  - Allocate servers exclusively
  - Define and run experiment test scripts
  - Get results in a Jupyter notebook
- Servers boot pre-built live images via PXE
  - Ensures reproducibility
  - Collection of different kernel versions/distributions

About me

- PhD student at Technical University of Munich
- Started in 2014
- PhD thesis about testing network devices
- Built the MoonGen packet generator for this
  - Used quite often in academia nowadays :)

Packet generators

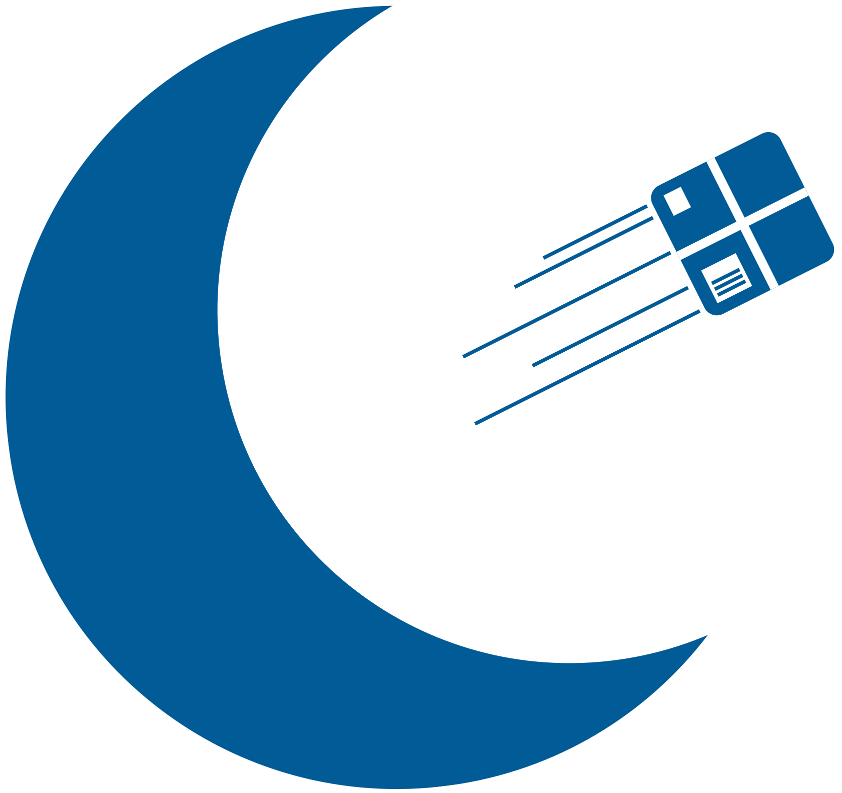
Source: www.spirent.com

Commodity hardware

Source: www.intel.com

MoonGen - A fast software packet generator

Combines the advantages of software (cheap, flexible) and hardware (precise, accurate) packet generators.



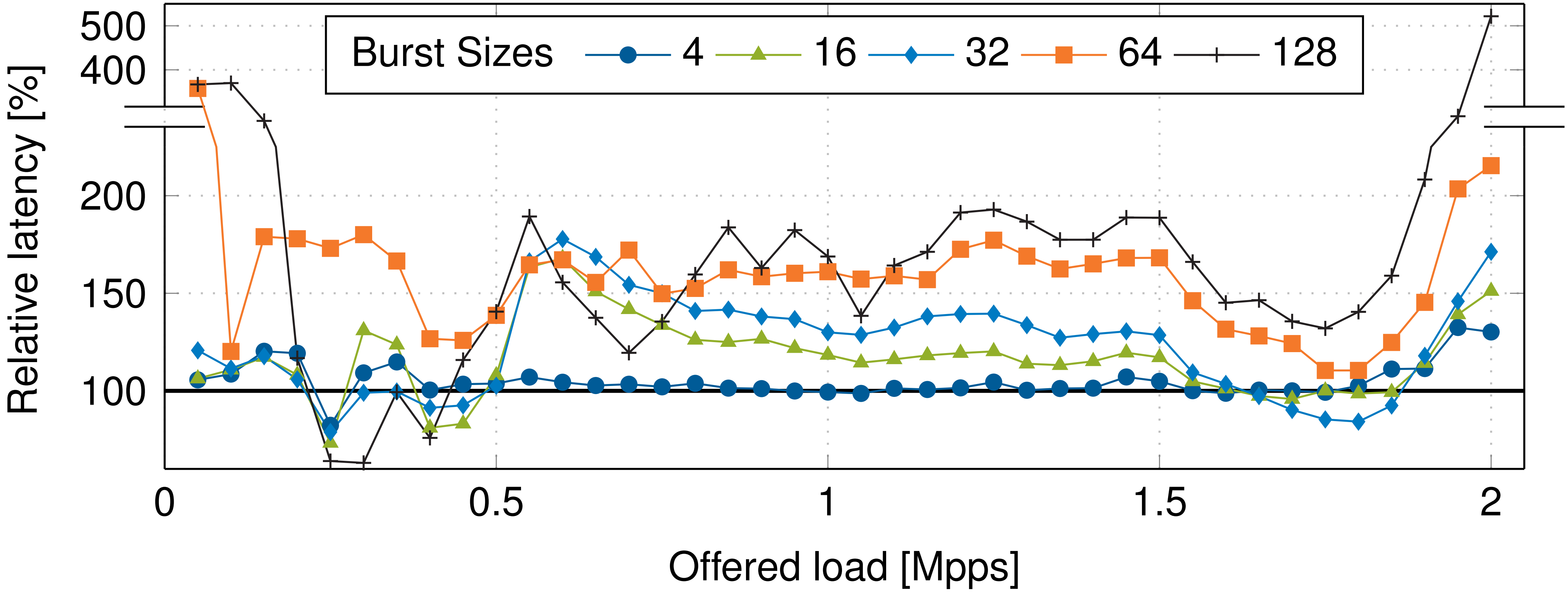
- *Fast*: DPDK for packet I/O, explicit multi-core support
- *Flexible*: Craft all packets in user-controlled Lua scripts
- *Timestamping*: Hardware features found on NICs
- *Rate control*: Hardware features and novel software approach
- *Free and open source*: Code available on GitHub

https://github.com/emmericp/MoonGen

Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle.
MoonGen: A Scriptable High-Speed Packet Generator. *Internet Measurement Conference (IMC) 2015*, October 2015.

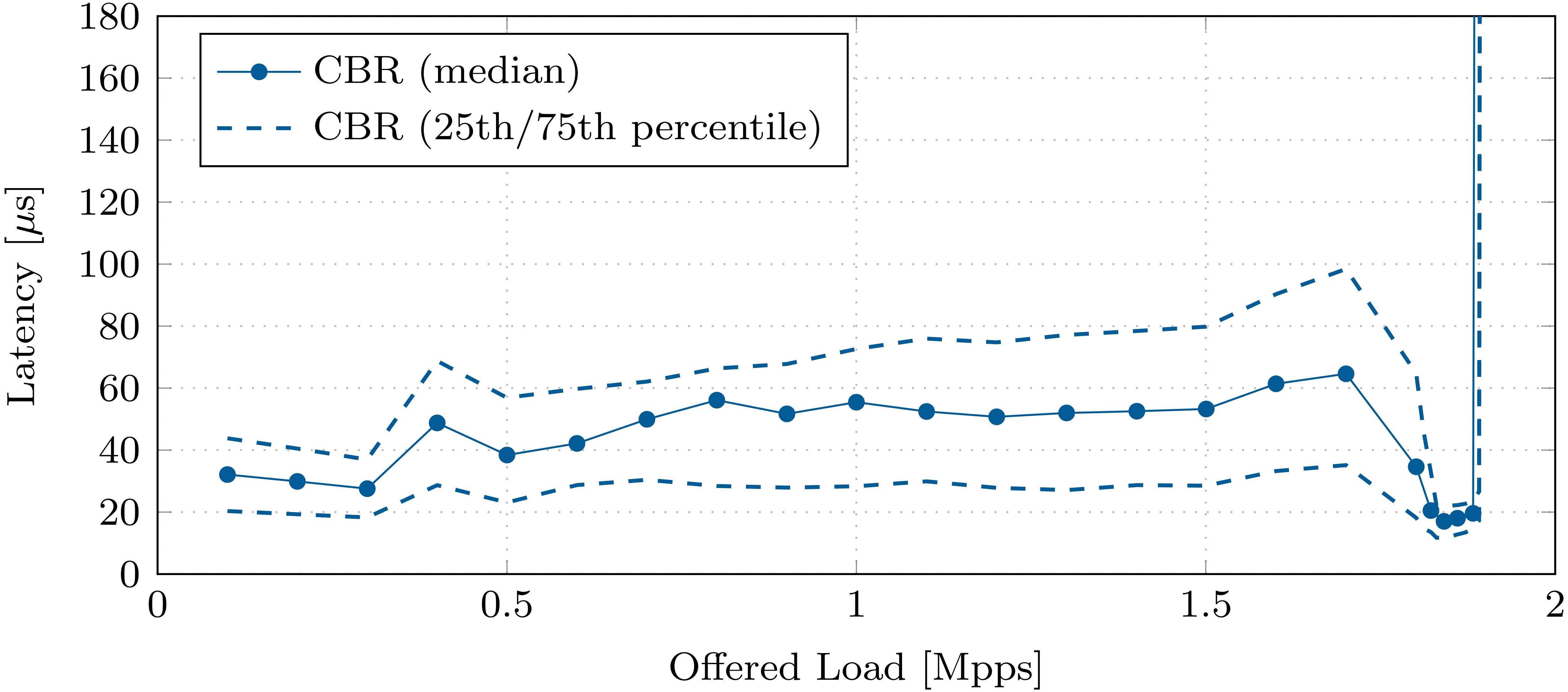
Traffic patterns matter: CBR is hard!

- Forwarding latency of Open vSwitch (kernel), increasing load
- Baseline latency: CBR traffic, varying burst sizes



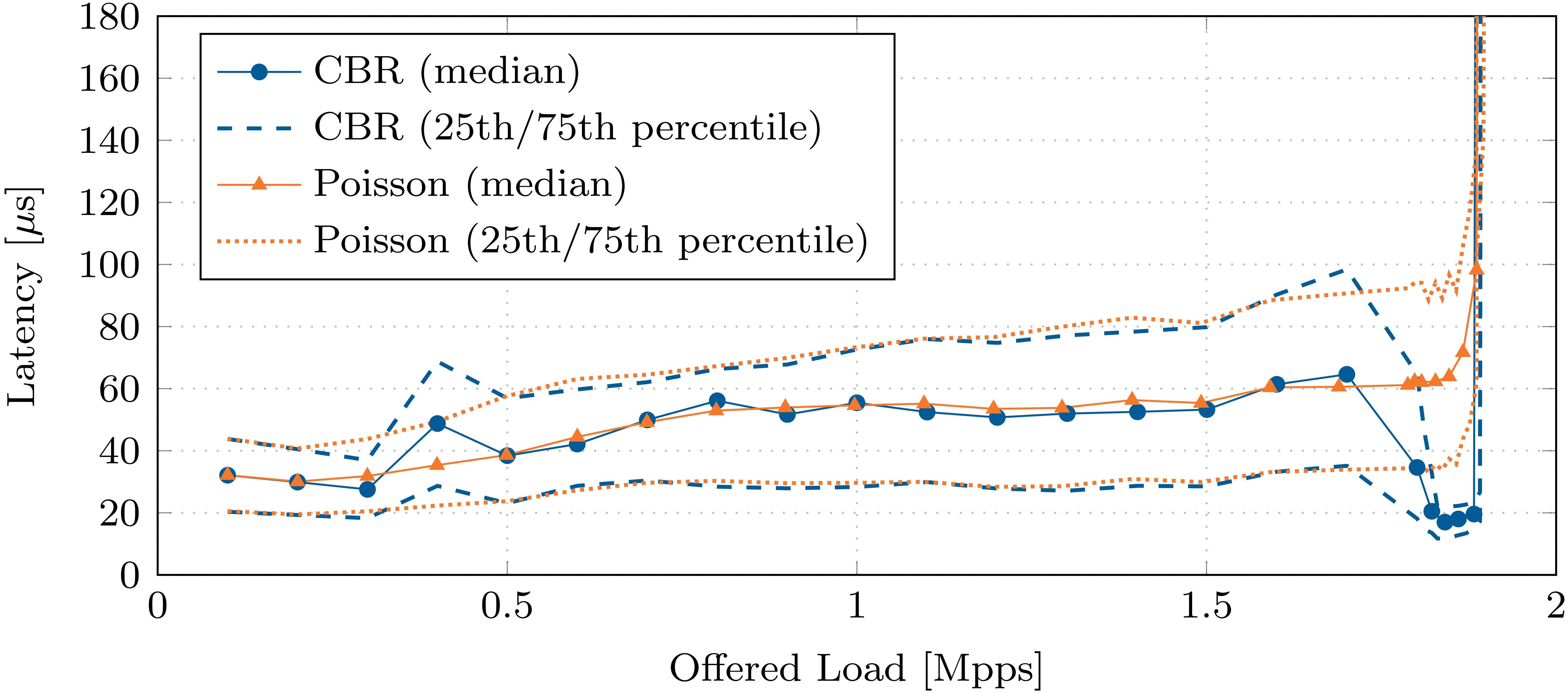
- Bursts are important for performance
- Typical default burst sizes: 16 to 256
- Packet generators often fail to generate CBR reliably

CBR can lead to weird effects



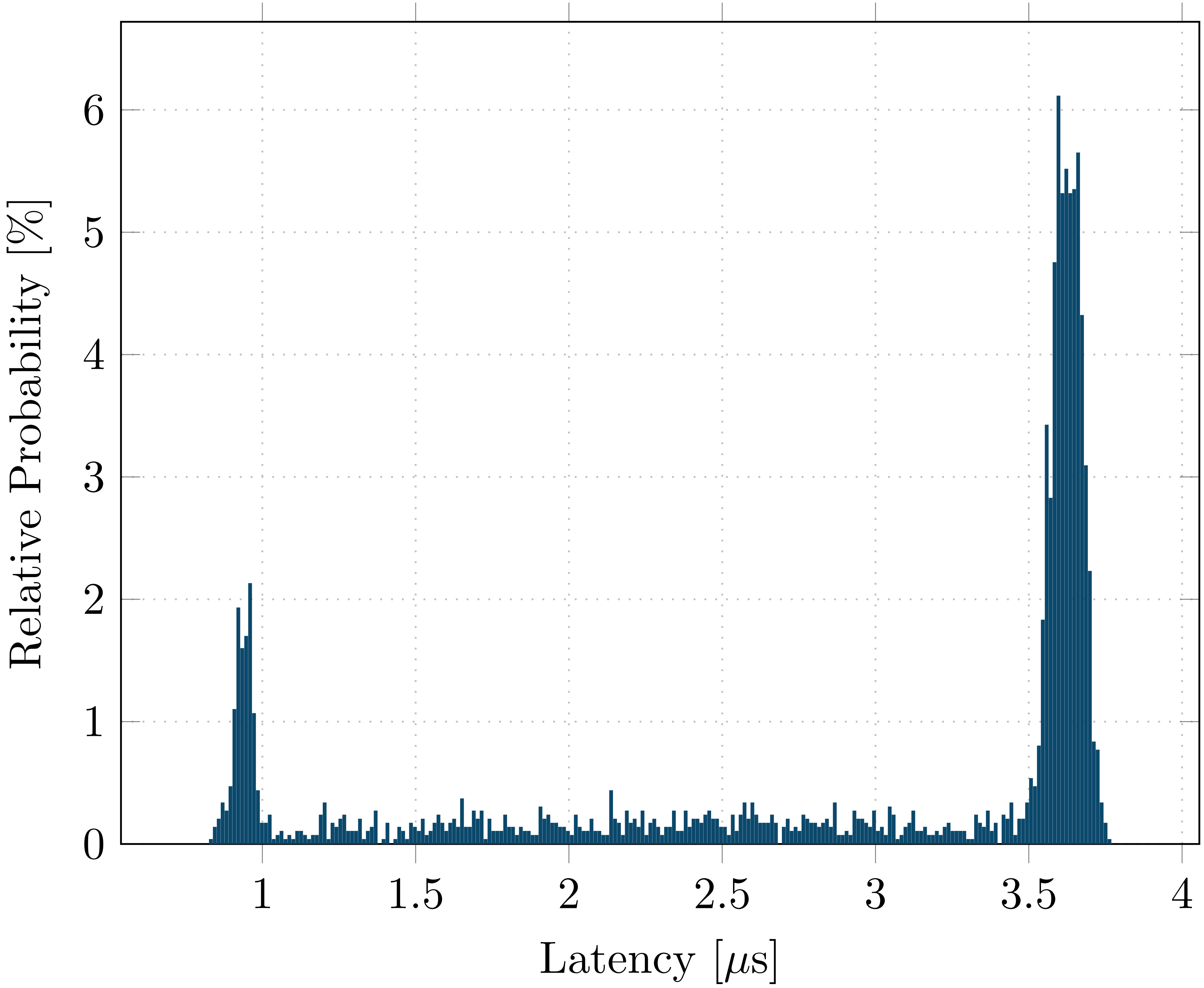
- Forwarding latency of Open vSwitch (kernel), increasing load
- Dynamic interrupt throttling (ixgbe driver) and poll-mode (NAPI) don’t play well with CBR traffic

Real-world traffic isn’t CBR

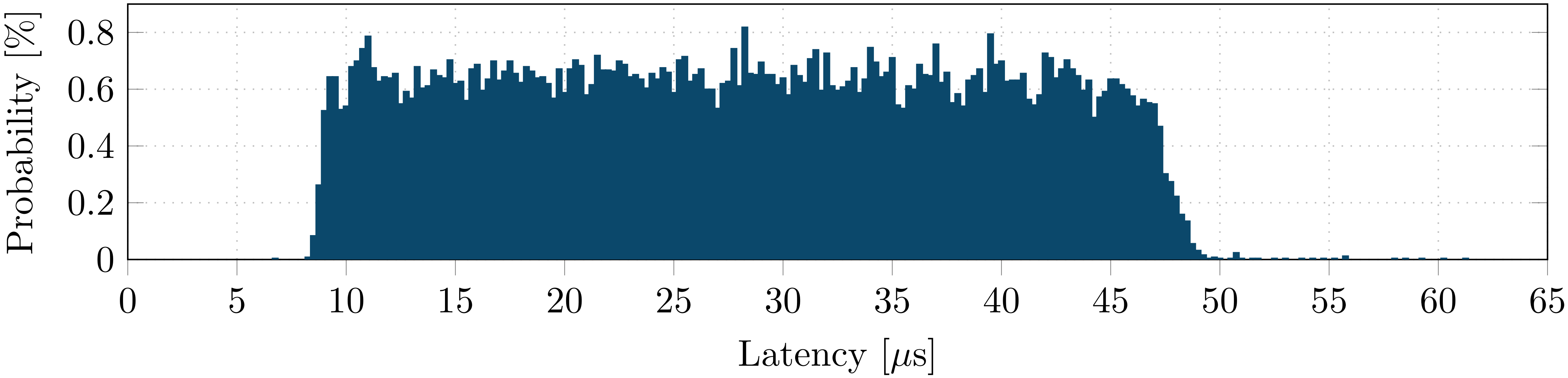


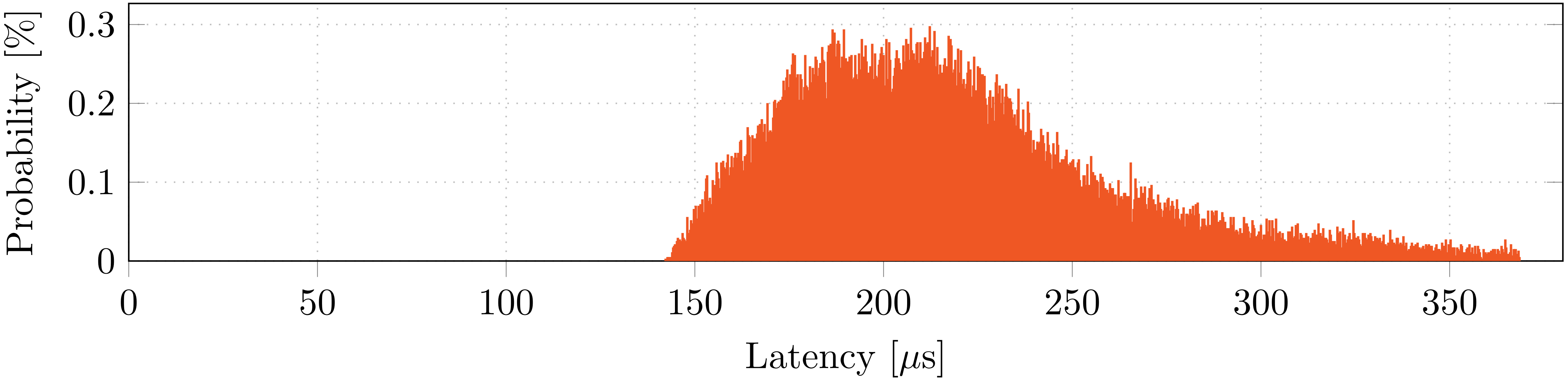
- Only change: time between packets
- Real-world traffic is a self-similar pattern
- Can be approximated with a Poisson process on short time scales

Latency measurements



Latency measurements





Generating complex packets

- Arbitrarily complex header stacks
- Generates and JIT compiles C structs
- Defaults for all header fields
  - E.g., calculates lengths, ports based on upper protocol
- Getters and setters, automatic endianness handling
- Following example code based on
  https://github.com/emmericp/moongen-scripts/blob/master/vxlan.lua

```
local vxlanStack = packetCreate(
  "eth", "ip4", "udp", "vxlan",
  {"eth_8021q", "innerEth"},
  {"ip4", "innerIp4"},
  {"udp", "innerUdp"}
)
```

Generating complex packets

- Create a mempool with a packet archetype

```
 local mempool = memory.createMemPool(function(buf)
    local pkt = vxlanStack(buf)
    pkt:fill{
        -- fields not explicitly set here are initialized to defaults
        ethSrc = queue, -- MAC of the tx device
        ethDst = arpTask.lookup("10.0.0.3"),
        ip4Src = "10.0.0.2",
        ip4Dst = "10.0.0.3",
        vxlanVNI = 10100,
        -- outer UDP ports are set automatically by the VXLAN handler
        innerEthSrc  = "12:34:56:78:90:ab",
        innerEthDst  = eth.BROADCAST,
        innerEthVlan = 100,
        innerIp4Src  = "192.168.0.1",
        innerIp4Dst  = "255.255.255.255",
        innerUdpSrc  = 1024,
        innerUdpDst  = 1024,
        pktLength    = 128
    }
    pkt.innerIp4:calculateChecksum()
end)
```

Generating complex packets

- Write a transmit loop

```
local bufs = mempool:bufArray()
while mg.running() do
    bufs:alloc()
    for i, buf in ipairs(bufs) do
        local pkt = vxlanStack(buf)
        pkt.innerUdp:setDstPort(
            1000 + math.random(0, 1000)
        )
        -- randomize other fields here
    end
    bufs:offloadUdpChecksums()
    queue:send(bufs)
 end
```

Don’t want to write a script? Use our CLI!

- Define one or multiple flows in a config file, e.g.

```
Flow{"syn-flood6", Packet.Tcp6{
             ethSrc = txQueue(),
             ethDst = mac"12:34:56:78:90:00",
             ip6Dst = ip"2a00:4700::2:225:90ff:fe74:7716",
             ip6Src = range(ip"fe80::1", ip“fe80::ffff:ffff“),
             tcpSrc = randomRange(0, 2^16 – 1),
             tcpDst = 80,
             tcpSyn = 1,
             tcpSeqNumber = randomRange(0, 2^32 – 1),
             tcpWindow = 10
     }
}
```

Don’t want to write a script? Use our CLI!

- Send out previously defined flows

```
./moongen−simple start syn−flood6:<dev>,<dev>:rate=40Gbit/s
```

- Combine arbitrary flows
- Different traffic patterns: CBR, Poisson, …
- Time limits for automated tests
- Per-flow packet counters
- Quick debugging by printing instead of sending
- See `./moongen−simple help` for more

- Caution: the CLI is still new and you might encounter bugs

How are others using MoonGen?

- OPNFV project: Test/benchmark framework VSPERF, MoonGen is one of multiple supported packet generators
- PISCES, SIGCOMM’16: Software P4 switch, performance evaluation
- NFVnice, SIGCOMM’17: NFV service chain scheduling, performance evaluation
- Flurries, CoNEXT’16: NFV framework, performance evaluation
- DNS DDoS Resilience Tests, RIPE 74: DNS traffic generation

How are others using MoonGen?

Project and authors	**Publication venue**	**Doing what**
PISCES Shahbaz et al.	SIGCOMM’16	Software P4 switch, performance evaluated with MoonGen. Contributed timestamping code for Intel 40 Gbit/s NIC.
Neutral Net Neutrality Yiakoumis et al.	SIGCOMM’16	Privacy-preserving quality of service, MoonGen used for the evaluation. Custom protocol/payload for test traffic.
NFVnice Kulkarni et al.	SIGCOMM’17	NFV chaining and scheduling, performance evaluated with MoonGen.
DNS DDoS Resilience Rincón et al.	RIPE-74	Replicating large DDoS attacks against DNS servers. Contributed DNS protocol code for MoonGen.
OPNFV VSPERF Linux Foundation	-	MoonGen is one of multiple supported packet generators to test and benchmark the OPNFV project. Complex MoonGen script as test harness.

Check out MoonGen on GitHub

MoonGen comes with a lot of examples

See if one fits your use case

`https://github.com/emmericp/MoonGen`

Questions?