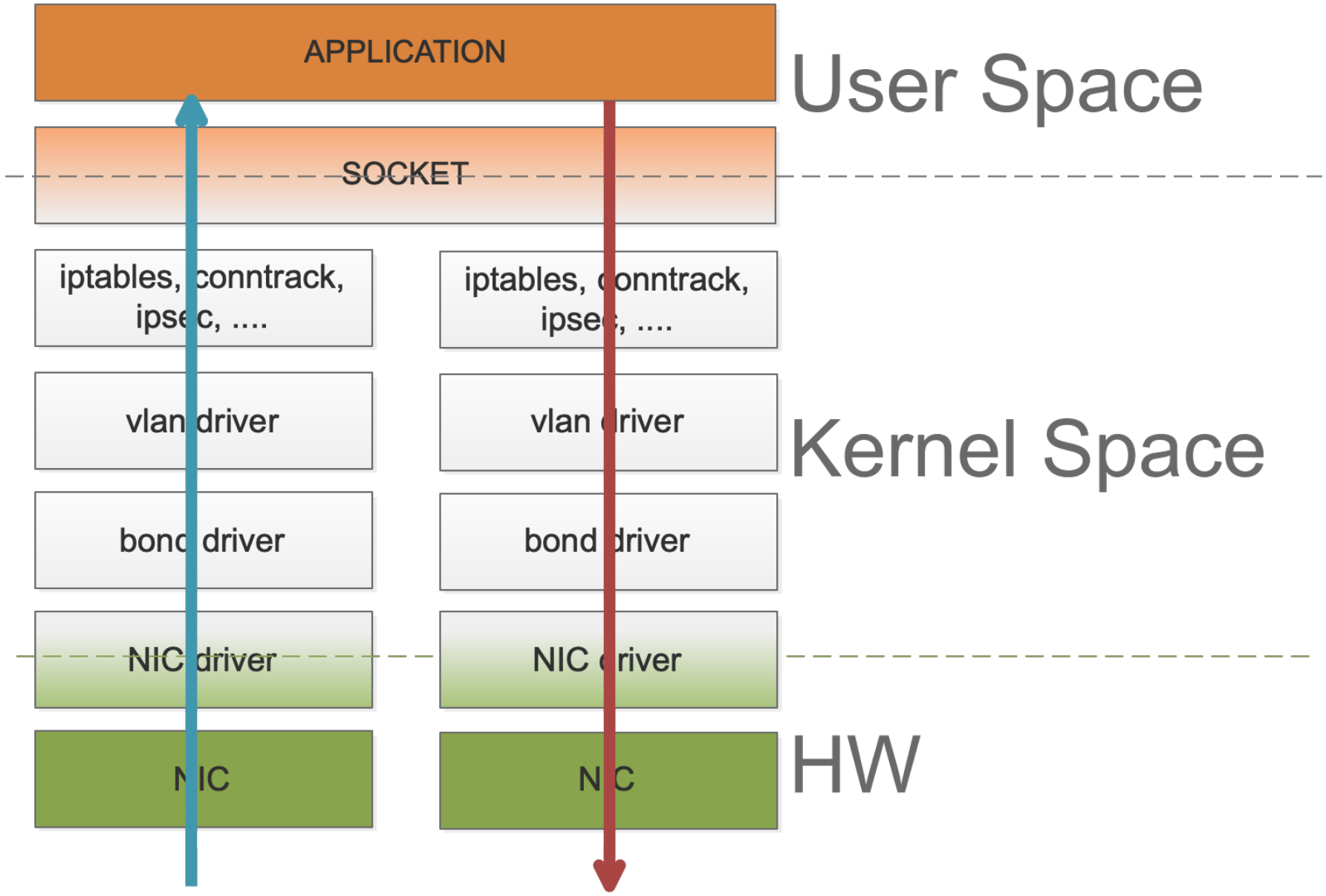
NA DORAZ II

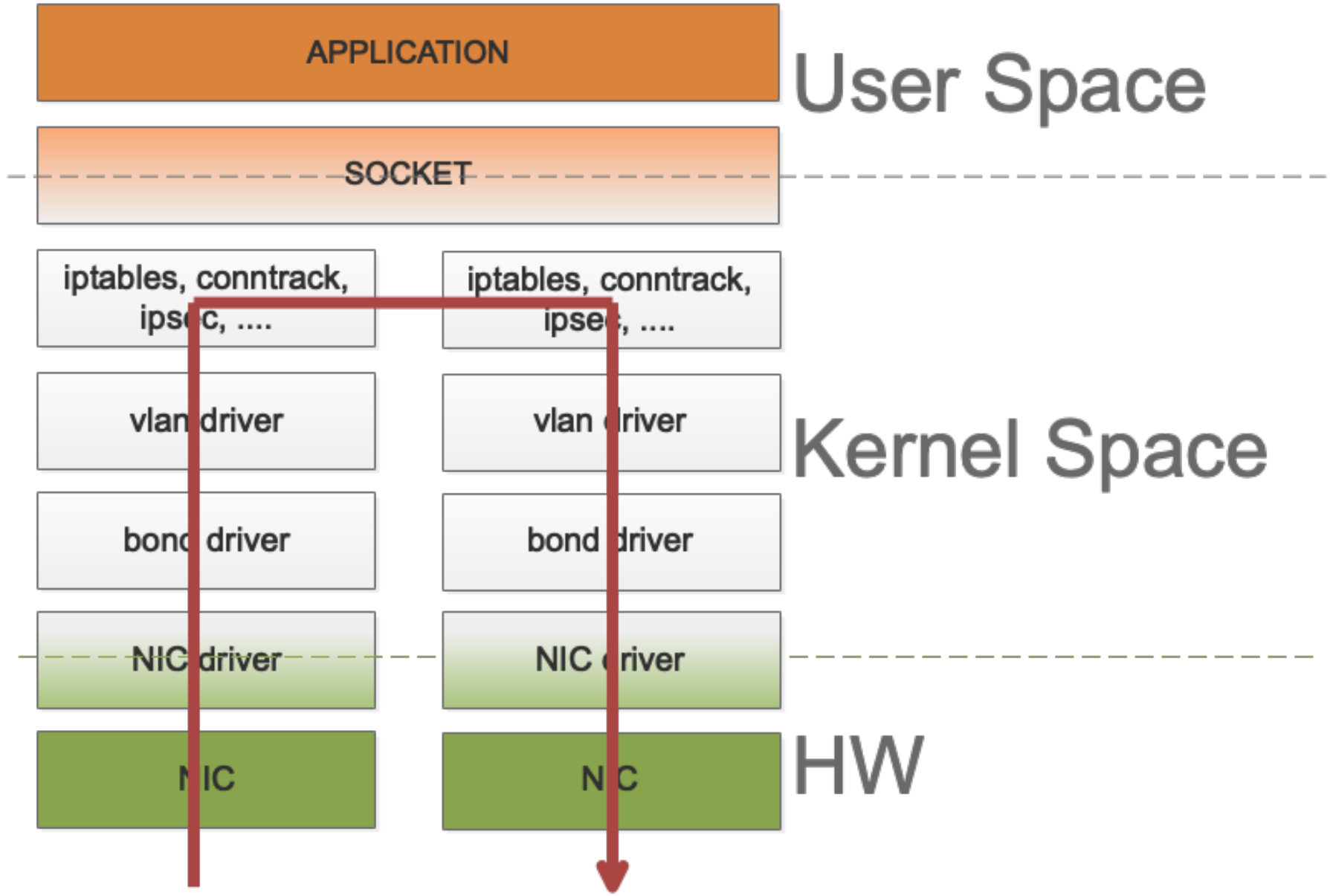
kam až sahají meze síťového subsystému Linuxového jádra tentokrát o plánovači paketů

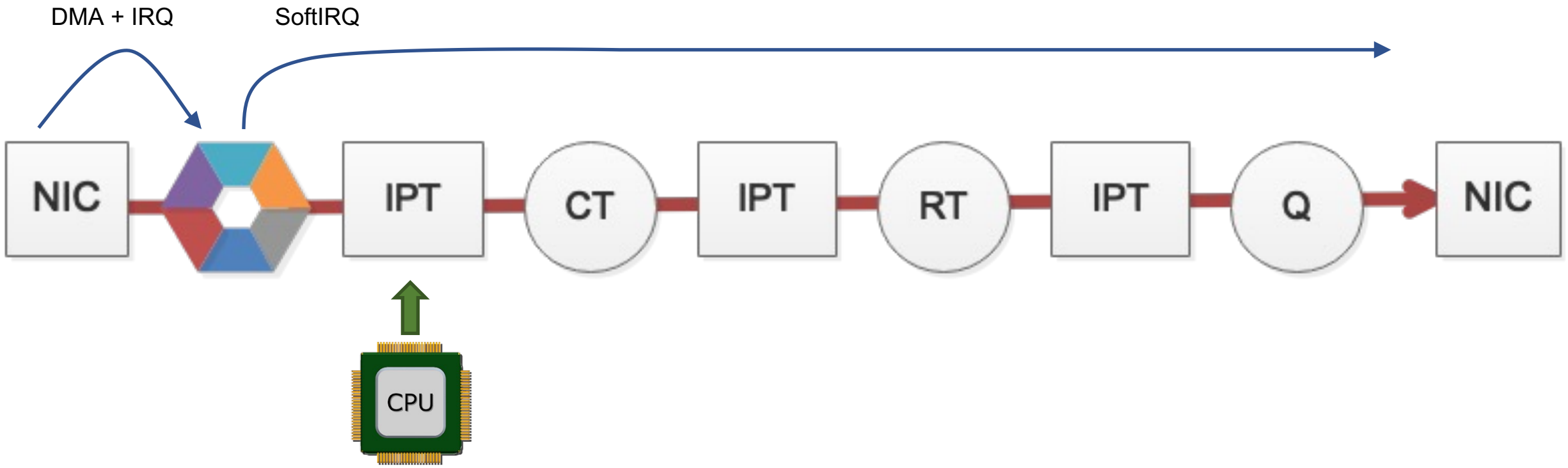
Tomáš Podermański
tpoder@vut.cz

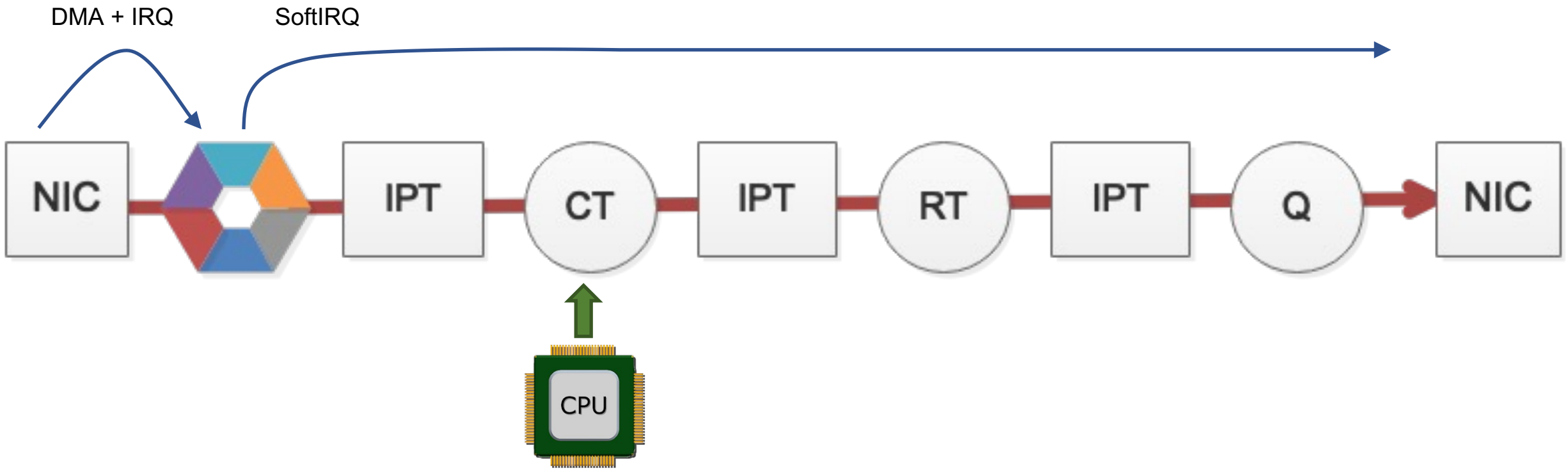
Matěj Grégr
gregr@netx.as

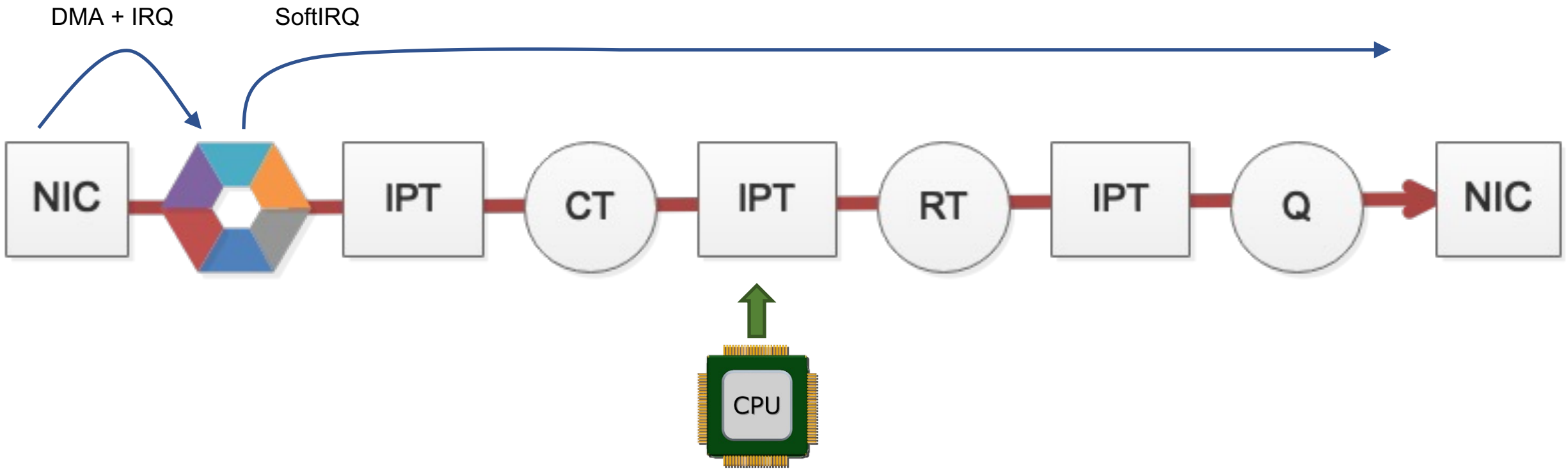
Peter Nagy
pater.nagy@vut.cz

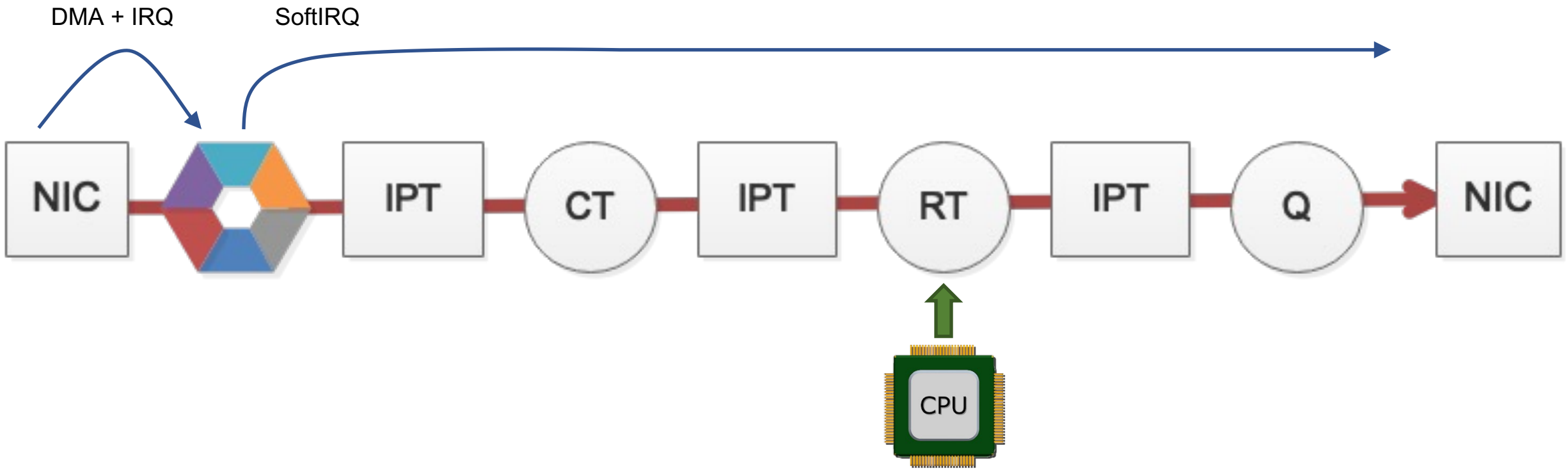

APPLICATION
SOCKET
iptables, conntrack, ipsec, ...
iptables, conntrack, ipsec, ...
vlan driver
vlan driver
bond driver
bond driver
NIC driver
NIC driver
NIC
NIC
User Space
Kernel Space
HW

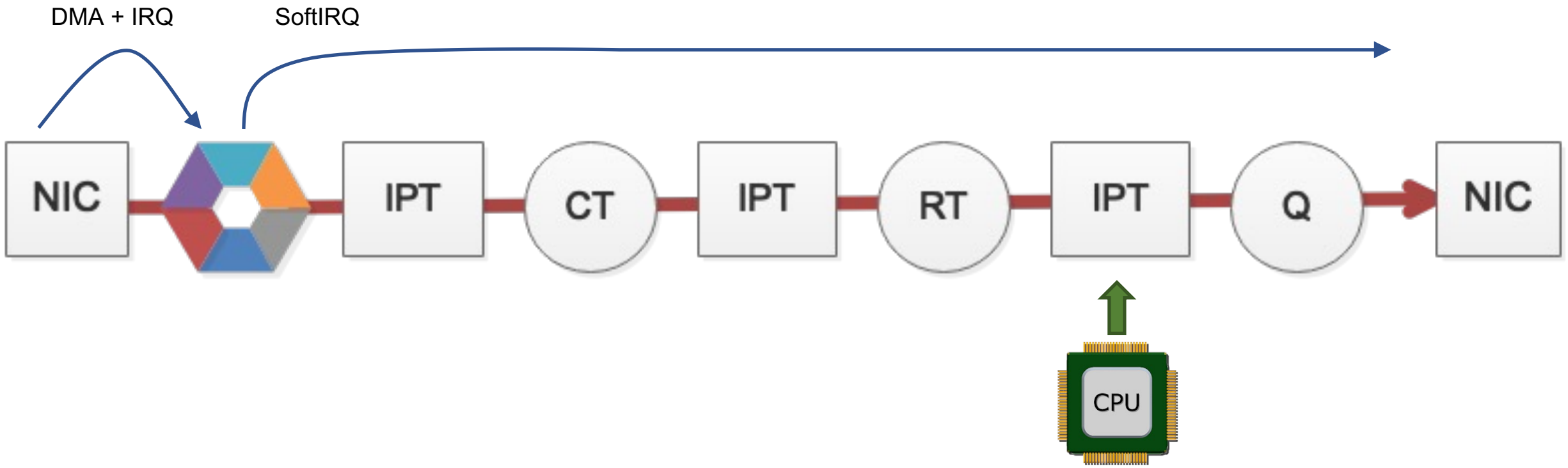

APPLICATION
User Space
SOCKET
iptables, conntrack, ipsec, ...
iptables, conntrack, ipsec, ...
vlan driver
vlan driver
bond driver
bond driver
NIC driver
NIC driver
NIC
NIC
Kernel Space
HW


DMA + IRQ
SoftIRQ
NIC
IPT
CT
IPT
RT
IPT
Q
NIC
CPU


DMA + IRQ
SoftIRQ
NIC
IPT
CT
IPT
RT
IPT
Q
NIC
CPU


DMA + IRQ
SoftIRQ
NIC
IPT
CT
IPT
RT
IPT
Q
NIC
CPU


DMA + IRQ
SoftIRQ
NIC
IPT
CT
IPT
RT
IPT
Q
NIC
CPU


DMA + IRQ
SoftIRQ
NIC
IPT
CT
IPT
RT
IPT
Q
NIC
CPU

40GbE směrovač pro operační systém GNU/Linux

Towards 40GbE GNU/Linux Router

Abstract

Účelem této práce je popis protokolu 40Gb Ethernet, popis směrovacího procesu v jádře Linux a navrhnout a provést testování výkonnosti směrování se síťovým adaptérem pro 40Gb Ethernet. Výsledky a nastavení pro získání maximální výkonnosti směrování jsou dále popsány v této práci.

The purpose of this thesis is to describe 40Gb Ethernet, describe routing process in the Linux kernel and to design and perform benchmark of routing performance with a 40Gb Ethernet network interface card. The results and system settings for achieving maximum routing performance are further described in the thesis.

View/Open

review_88683.html (1.452Kb)
final-thesis.pdf (2.397Mb)
Posudek-Oponent prace-17590_o.pdf (88.48Kb)
Posudek-Vedouci prace-17590_v.pdf (85.56Kb)

Author

Lučtický, Josef

Advisor

Grégr, Matěj

Referee

Podermański, Tomáš

Grade

A

Keywords

GNU, Linux, ethernet, směrovač, software, IP, síť, měření, propustnost, operační systém, GNU, Linux, ethernet, router, software, IP, network, measurement, throughput, operating system

Language

angličtina (English)

Study brunch

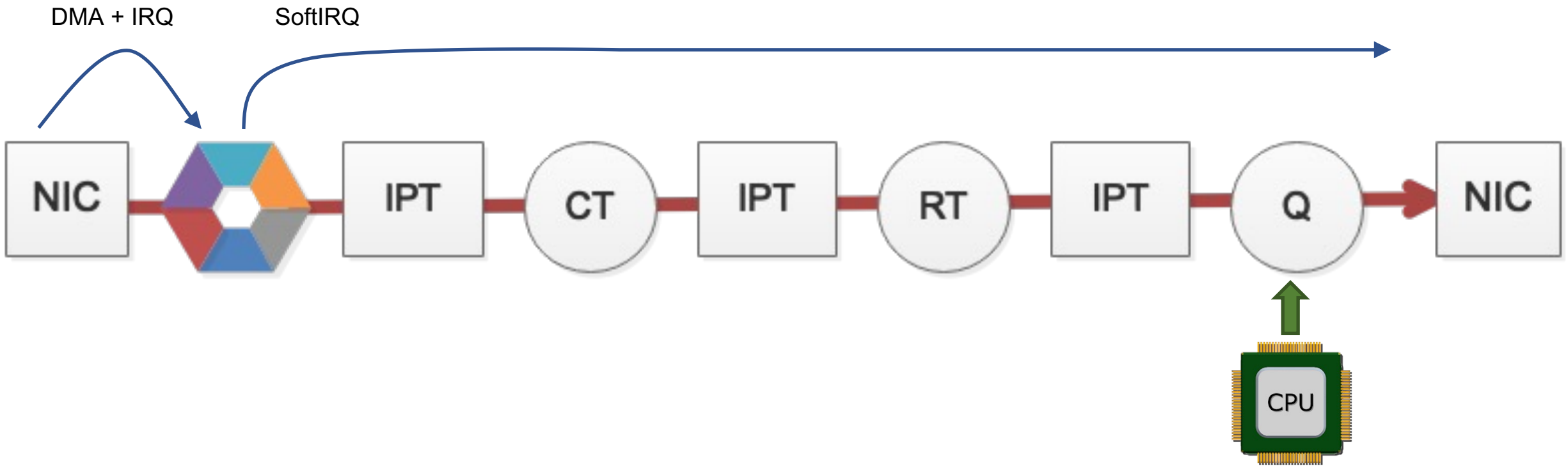
Počítačové sítě a komunikace

Composition of Committee

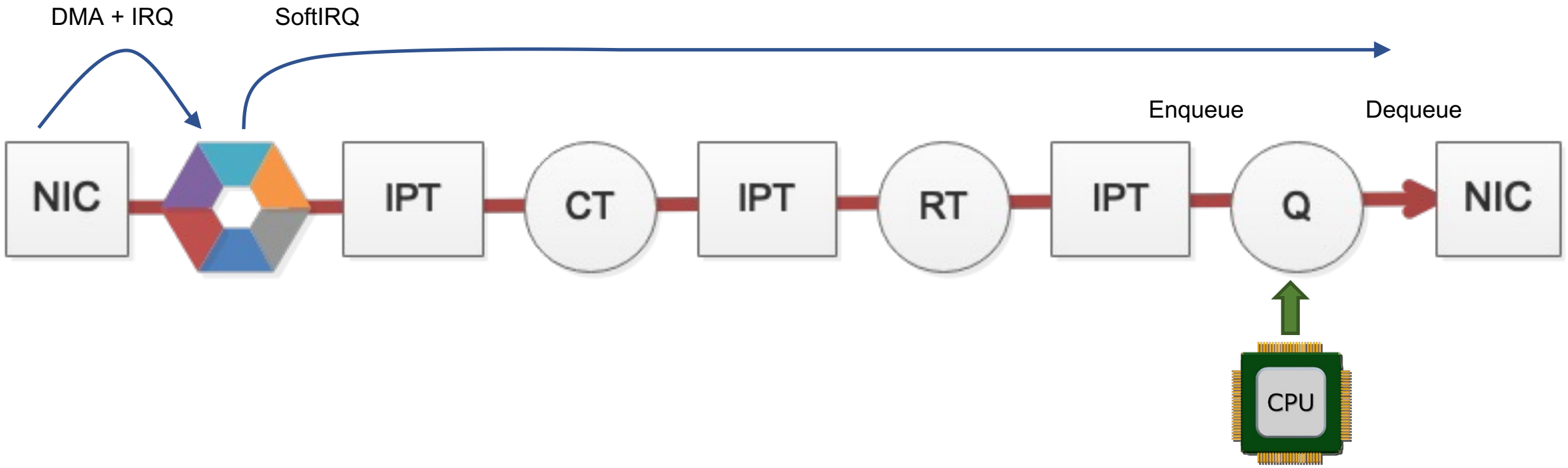
prof. Ing. Miroslav Švéda, CSc. (předseda) doc. Ing. Zdeněk Kotásek, CSc. (místopředseda) doc. Dr. Ing. Otto Fučík (člen) doc. Ing. Vladimír Janoušek, Ph.D. (člen) doc. Ing. Jiří Jaroš, Ph.D. (člen) doc. Ing. Stanislav Racek, CSc. (člen)

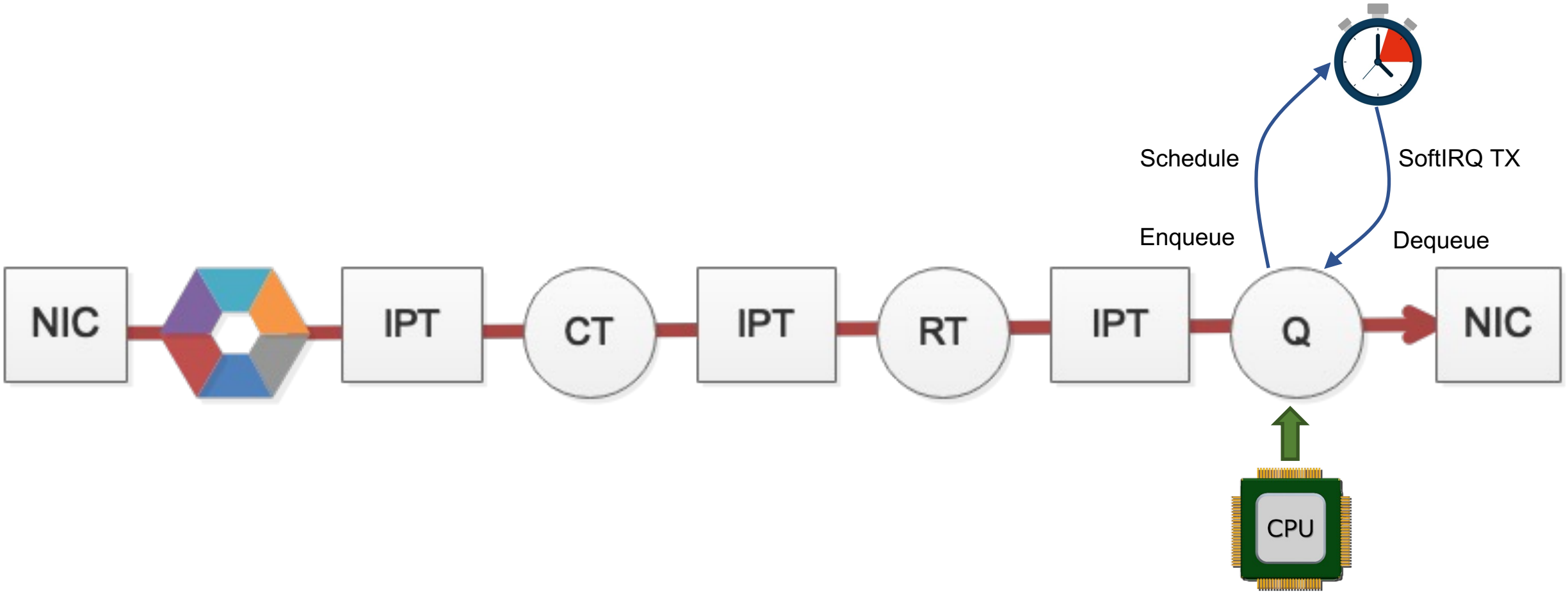
Date of defence

2015-06-25


DMA + IRQ
SoftIRQ
NIC
IPT
CT
IPT
RT
IPT
Q
NIC
CPU


enqueue
dequeue


DMA + IRQ
SoftIRQ
Enqueue
Dequeue
NIC
IPT
CT
IPT
RT
IPT
Q
NIC
CPU


Schedule
SoftIRQ TX
Enqueue
Dequeue
NIC
IPT
CT
IPT
RT
IPT
Q
NIC
CPU

tc qdisc add dev eth0 root **pfifo**

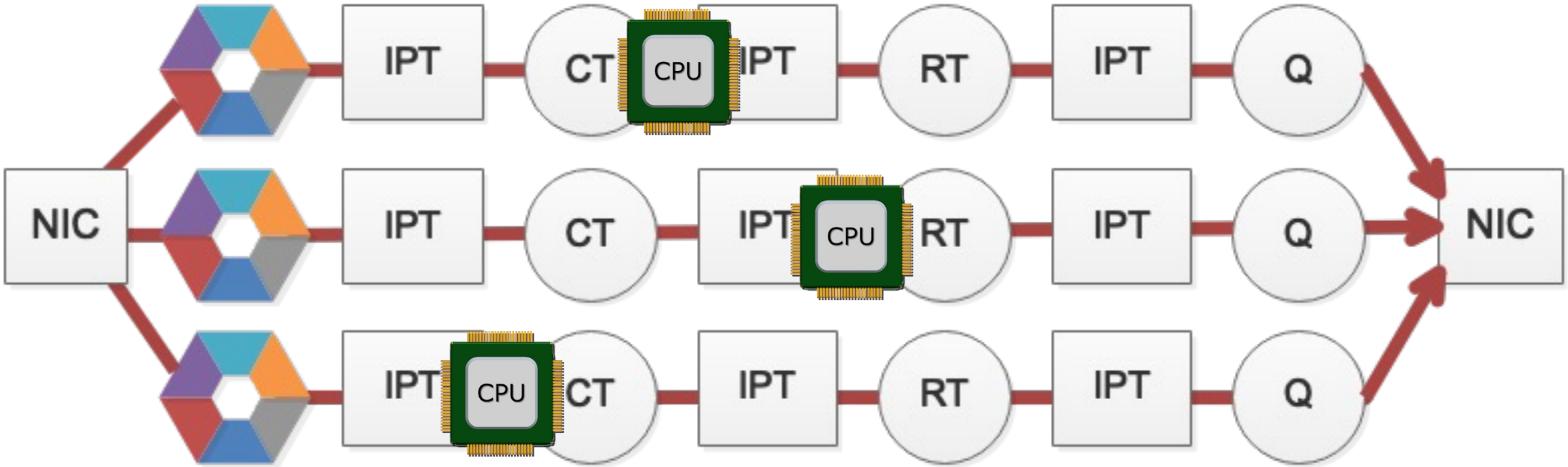
tc qdisc add dev eth0 root **pfifo_fast**

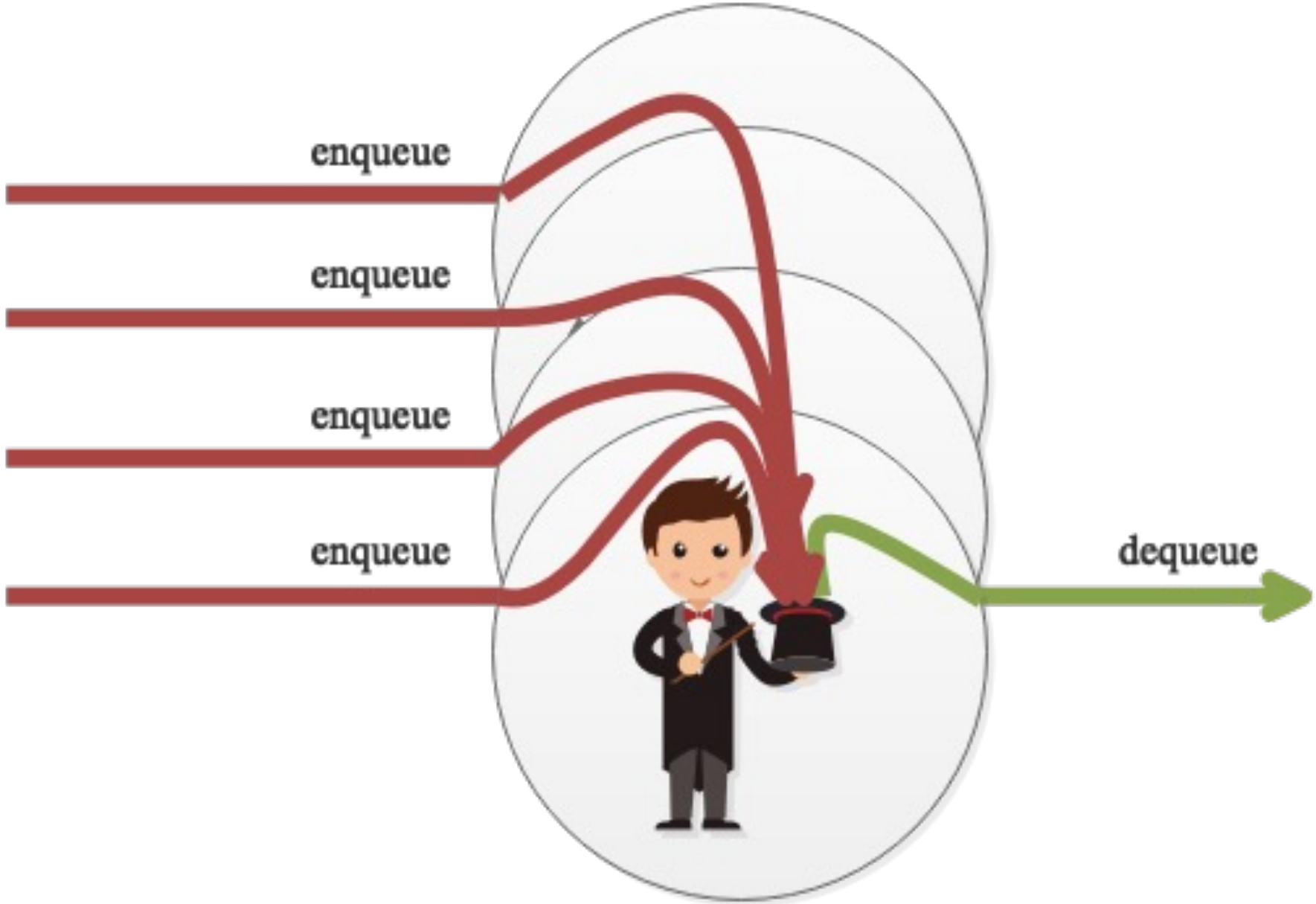
tc qdisc add dev eth0 root **sfq**

tc qdisc add dev eth0 root handle 1: **htb**
tc class add dev eth0 parent 1: classid 1:1 htb rate 100Mbit
tc class add dev eth0 parent 1: classid 1:2 htb rate 100Mbit
tc class add dev eth0 parent 1:1 classid 1:11 htb rate 60Mbit
tc class add dev eth0 parent 1:1 classid 1:12 htb rate 60Mbit

sysctl -w **net.core.dev_weight=64**

tc qdisc add dev eth0 root pfifo **limit 1000**

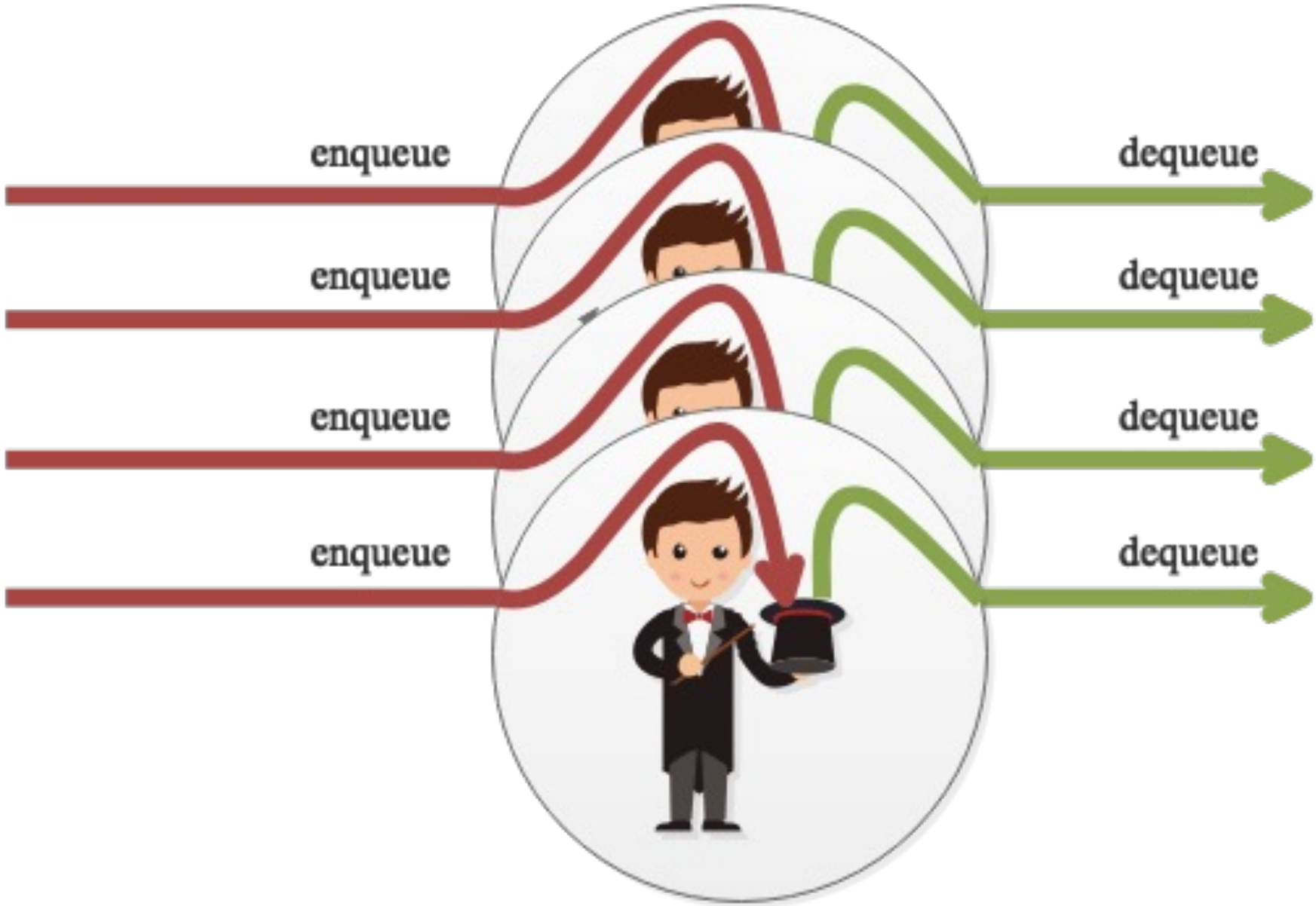

NIC
IPT
CT
CPU
IPT
RT
IPT
Q
IPT
CT
IPT
CPU
RT
IPT
Q
IPT
CPU
CT
IPT
RT
IPT
Q
NIC


enqueue
enqueue
enqueue
enqueue
dequeue

Filter tags

- v6
  - v6.1
    - v6.1-rc4
    - v6.1-rc3
    - v6.1-rc2
    - v6.1-rc1
  - v6.0
- v5
- v4
- v3
- v2
- v1
- v0

```
3848
3849        spin_lock(root_lock);
3850        if (unlikely(test_bit(__QDISC_STATE_DEACTIVATED, &q->state))) {
3851                __qdisc_drop(skb, &to_free);
3852                rc = NET_XMIT_DROP;
3853        } else if ((q->flags & TCQ_F_CAN_BYPASS) && !qdisc_qlen(q) &&
3854                   qdisc_run_begin(q)) {
3855                /*
3856                 * This is a work-conserving queue; there are no old skbs
3857                 * waiting to be sent out; and the qdisc is not running -
3858                 * xmit the skb directly.
3859                 */
3860
3861                qdisc_bstats_update(q, skb);
3862
3863                if (sch_direct_xmit(skb, q, dev, txq, root_lock, true)) {
3864                        if (unlikely(contended)) {
3865                                spin_unlock(&q->busylock);
3866                                contended = false;
3867                        }
3868                        __qdisc_run(q);
3869                }
3870
3871                qdisc_run_end(q);
3872                rc = NET_XMIT_SUCCESS;
3873        } else {
3874                rc = dev_qdisc_enqueue(skb, q, &to_free, txq);
3875                if (qdisc_run_begin(q)) {
3876                        if (unlikely(contended)) {
3877                                spin_unlock(&q->busylock);
3878                                contended = false;
3879                        }
3880                        __qdisc_run(q);
3881                        qdisc_run_end(q);
3882                }
3883        }
3884        spin_unlock(root_lock);
```


enqueue
enqueue
enqueue
enqueue
dequeue
dequeue
dequeue
dequeue

V některých případech řešitelné přes:

tc qdisc add dev eth0 root **mq**

Nicméně...

```
# tc qdisc add dev eth0 root handle 1:  htb
# tc class add dev eth0 parent 1: classid 1:1 htb rate 100Mbit
# tc class add dev eth0 parent 1: classid 1:2 htb rate 100Mbit
# tc class add dev eth0 parent 1:1 classid 1:11 htb rate 60Mbit
# tc class add dev eth0 parent 1:1 classid 1:12 htb rate 60Mbit
```

Dopad na výkon?

Intel(R) Xeon(R) Gold 6248R CPU @ 3.00GHz, 48 jader, pakety 100B

tc qdisc add dev eth0 root **mq:** propustnost: **14,1 Mp/s**

tc qdisc add dev eth0 root **pfifo:** propustnost: **2,5 Mp/s**

tc qdisc add dev eth0 root **sfq:** propustnost: **2,4 Mp/s**

tc qdisc add dev eth0 root **htb:** propustnost: **2,4 Mp/s**

```
 1  [|                    1.3%]   13 [|                    1.2%]   25 [|                    3.4%]   37 [|                    1.0%]
 2  [|                    4.6%]   14 [|                    1.2%]   26 [|                    2.4%]   38 [                     0.0%]
 3  [|                    1.2%]   15 [                     0.0%]   27 [|                    1.2%]   39 [|                    2.9%]
 4  [|                    1.2%]   16 [|                    3.6%]   28 [|                    2.3%]   40 [|                    1.0%]
 5  [|                    2.4%]   17 [|                    3.5%]   29 [|                    1.2%]   41 [|                    1.0%]
 6  [|                    2.4%]   18 [|                    2.3%]   30 [|                    1.2%]   42 [|                    1.0%]
 7  [|                    2.4%]   19 [|                    1.2%]   31 [                     0.0%]   43 [|                    2.0%]
 8  [                     0.0%]   20 [|                    2.4%]   32 [                     0.0%]   44 [|                    1.0%]
 9  [|                    2.4%]   21 [                     0.0%]   33 [|                    1.2%]   45 [|                    1.0%]
10  [|                    1.2%]   22 [|                    2.3%]   34 [|                    1.2%]   46 [|                    1.0%]
11  [                     0.0%]   23 [|                    2.4%]   35 [|                    1.2%]   47 [|                    2.0%]
12  [|                    2.4%]   24 [                     0.0%]   36 [|                    2.0%]   48 [|                    2.0%]
Mem[||||||||                                      4.64G/45.2G]   Tasks: 38, 13 thr; 4 running
Swp[                                                 0K/16.0G]   Load average: 15.58 9.55 4.33
                                                                 Uptime: 00:23:19

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
19870 root       20   0  120M  4180  3152 R  2.2  0.0  0:00.58 htop
 5448 root       20   0  143M  7276  4148 S  0.0  0.0  0:00.73 /usr/bin/perl /usr/bin/cpuload
 5467 root       20   0  4604  1724  1484 S  0.0  0.0  0:00.19 /usr/bin/turbostat --quiet -i 5 --show CPU Busy%
  834 root       20   0 39152  8068  7692 S  0.0  0.0  0:00.11 /usr/lib/systemd/systemd-journald
    1 root       20   0  186M  5408  3984 S  0.0  0.0  0:01.14 /usr/lib/systemd/systemd --switched-root --system --deser
  860 root       20   0 46724  4668  2964 S  0.0  0.0  0:00.97 /usr/lib/systemd/systemd-udevd
1070 root       16  -4 55540  2324  1916 S  0.0  0.0  0:00.00 /sbin/auditd
1069 root       16  -4 55540  2324  1916 S  0.0  0.0  0:00.00 /sbin/auditd
1098 polkitd    20   0  598M 15768  9052 S  0.0  0.0  0:00.00 /usr/lib/polkit-1/polkitd --no-debug
1100 polkitd    20   0  598M 15768  9052 S  0.0  0.0  0:00.00 /usr/lib/polkit-1/polkitd --no-debug
1102 polkitd    20   0  598M 15768  9052 S  0.0  0.0  0:00.00 /usr/lib/polkit-1/polkitd --no-debug
1104 polkitd    20   0  598M 15768  9052 S  0.0  0.0  0:00.00 /usr/lib/polkit-1/polkitd --no-debug
1106 polkitd    20   0  598M 15768  9052 S  0.0  0.0  0:00.00 /usr/lib/polkit-1/polkitd --no-debug
1113 polkitd    20   0  598M 15768  9052 S  0.0  0.0  0:00.00 /usr/lib/polkit-1/polkitd --no-debug
1093 polkitd    20   0  598M 15768  9052 S  0.0  0.0  0:00.02 /usr/lib/polkit-1/polkitd --no-debug
1096 dbus       20   0 58120  4240  3704 S  0.0  0.0  0:00.06 /usr/bin/dbus-daemon --system --address=systemd: --nofork
```

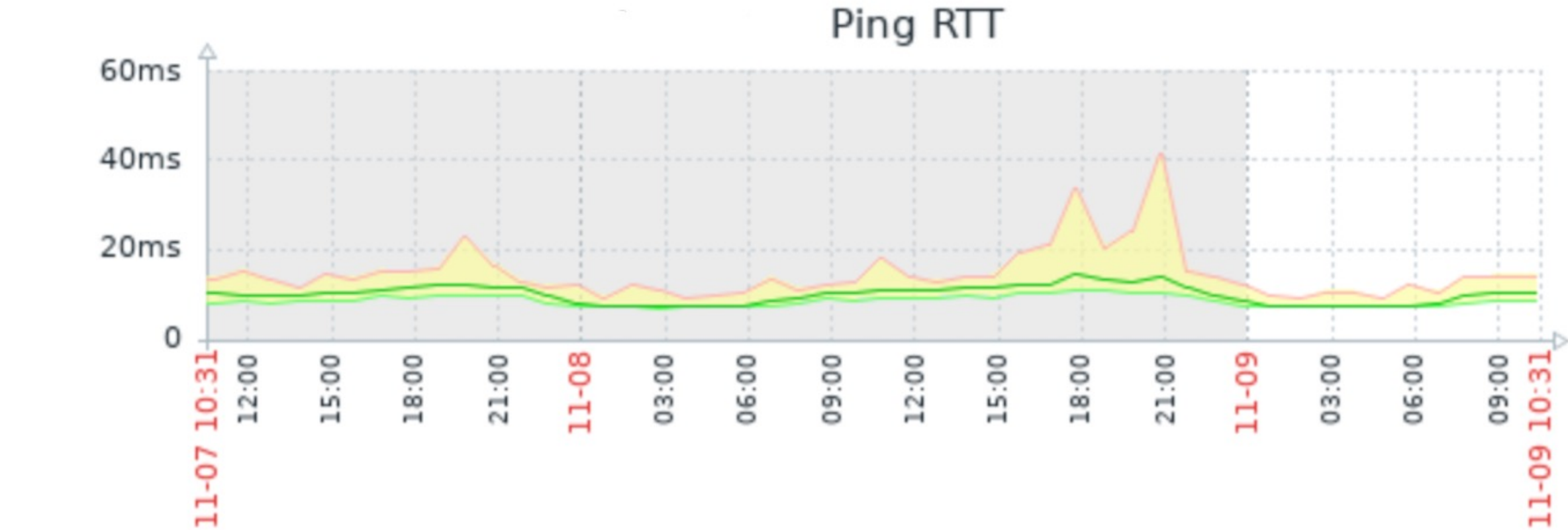
```
Samples: 4M of event 'cycles', 4000 Hz, Event count (approx.): 1659274460875 lo
Overhead  Shared Object             Symbol
  86.98%  [kernel]                  [k] native_queued_spin_lock_slowpath
   1.29%  [kernel]                  [k] _raw_spin_lock
   0.89%  [kernel]                  [k] ipt_do_table
   0.72%  [kernel]                  [k] __dev_queue_xmit
   0.41%  [kernel]                  [k] ___bpf_prog_run
   0.30%  [kernel]                  [k] page_frag_free
   0.27%  [kernel]                  [k] mlx5e_skb_from_cqe_mpwrq_linear
   0.26%  [kernel]                  [k] mlx5e_poll_tx_cq
   0.25%  [kernel]                  [k] ip_forward
   0.24%  [kernel]                  [k] mlx5e_sq_xmit
   0.24%  [kernel]                  [k] __netif_receive_skb_core
   0.21%  [kernel]                  [k] htb_dequeue
   0.20%  [kernel]                  [k] mlx5_eq_comp_int
   0.20%  [kernel]                  [k] mlx5e_poll_rx_cq
   0.18%  [kernel]                  [k] udp_v4_early_demux
   0.18%  [kernel]                  [k] htb_enqueue
   0.17%  [kernel]                  [k] fib_table_lookup
   0.15%  [kernel]                  [k] __qdisc_run
   0.14%  [kernel]                  [k] build_skb
   0.13%  [kernel]                  [k] netif_skb_features
   0.13%  [kernel]                  [k] mlx5e_build_rx_skb
   0.13%  [kernel]                  [k] ip_route_input_slow
   0.12%  [kernel]                  [k] ip_sublist_rcv
   0.12%  [kernel]                  [k] tasklet_action_common.isra.0
   0.12%  [kernel]                  [k] vlan_do_receive
   0.11%  [kernel]                  [k] dev_gro_receive
   0.11%  [kernel]                  [k] ip_finish_output2
   0.11%  [kernel]                  [k] read_tsc
   0.11%  [kernel]                  [k] nf_hook_slow
   0.10%  [kernel]                  [k] validate_xmit_skb
   0.10%  [kernel]                  [k] __local_bh_enable_ip
   0.10%  [kernel]                  [k] mlx5e_xmit
```

Jak takový stav poznat ?

```
# ethtool -S eth0 | grep disc
     rx_discards_phy: 1276981171
     tx_discards_phy: 0

# ethtool -S eth0 | grep drop
     rx_xdp_drop: 0
     tx_queue_dropped: 0
     rx_oversize_pkts_sw_drop: 0
     rx_xsk_xdp_drop: 0
     rx_xsk_oversize_pkts_sw_drop: 0

# ip -s link show eth0
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFAULT group
    link/ether 08:c0:eb:48:6a:07 brd ff:ff:ff:ff:ff:ff
    RX:      bytes     packets errors  dropped  missed   mcast
    25129218934472 16842643303      0 50058429       0    3189
    TX:      bytes     packets errors  dropped carrier collsns
           699808        3219      0        0       0       0
```



		last	min	avg	max
Ping RTT	[all]	10ms	6.9ms	9.87ms	41.3ms

A jak je na tom MikroTik ?

CCR2116-12G-4S+, ROS 7, pakety 100B

only-hardware-queue, fast-path: propustnost: **8,6 Mp/s**

sfq: propustnost: **8,6 Mp/s**

pcq propustnost: **0,9 Mp/s**

cake: propustnost: **0,5 Mp/s**

queue tree: propustnost: **1,2 Mp/s**

simple queue : propustnost: **2,1 Mp/s**

Bude líp?

linux-2.1.65.tar.bz2	18-Nov-1997 03:00	7M
linux-2.1.65.tar.gz	18-Nov-1997 03:00	9M
linux-2.1.65.tar.sign	08-Aug-2013 19:18	665
linux-2.1.65.tar.xz	18-Nov-1997 03:00	6M
linux-2.1.66.tar.bz2	26-Nov-1997 00:15	7M
linux-2.1.66.tar.gz	26-Nov-1997 00:15	9M
linux-2.1.66.tar.sign	08-Aug-2013 19:18	665
linux-2.1.66.tar.xz	26-Nov-1997 00:15	6M
linux-2.1.67.tar.bz2	29-Nov-1997 19:18	7M
linux-2.1.67.tar.gz	29-Nov-1997 19:18	9M
linux-2.1.67.tar.sign	08-Aug-2013 19:18	665
linux-2.1.67.tar.xz	29-Nov-1997 19:18	6M
linux-2.1.68.tar.bz2	30-Nov-1997 23:15	7M
linux-2.1.68.tar.gz	30-Nov-1997 23:15	9M
linux-2.1.68.tar.sign	08-Aug-2013 19:18	665
linux-2.1.68.tar.xz	30-Nov-1997 23:15	6M
linux-2.1.69.tar.bz2	01-Dec-1997 21:42	7M
linux-2.1.69.tar.gz	01-Dec-1997 21:42	9M
linux-2.1.69.tar.sign	08-Aug-2013 19:18	665
linux-2.1.69.tar.xz	01-Dec-1997 21:42	6M
linux-2.1.7.tar.bz2	01-Nov-1996 14:38	5M
linux-2.1.7.tar.gz	01-Nov-1996 14:38	6M
linux-2.1.7.tar.sign	08-Aug-2013 19:18	665

Pentium II

From Wikipedia, the free encyclopedia

The **Pentium II**[2] brand refers to Intel's sixth-generation microarchitecture ("P6") and x86-compatible microprocessors introduced on May 7, 1997. Containing 7.5 million transistors (27.4 million in the case of the mobile Dixon with 256 KB L2 cache), the Pentium II featured an improved version of the first *P6*-generation core of the Pentium Pro, which contained 5.5 million transistors. However, its L2 cache subsystem was a downgrade when compared to the Pentium Pros. It is a single-core microprocessor.

In 1998, Intel stratified the Pentium II family by releasing the Pentium II-based Celeron line of processors for low-end workstations and the Pentium II Xeon line for servers and high-end workstations. The Celeron was characterized by a reduced or omitted (in some cases present but disabled) on-die full-speed L2 cache and a 66 MT/s FSB. The Xeon was characterized by a range of full-speed L2 cache (from 512 KB to 2048 KB), a 100 MT/s FSB, a different physical interface (Slot 2), and support for symmetric multiprocessing.

In February 1999, the Pentium II was replaced by the nearly identical Pentium III, which only added the then-new SSE instruction set. However, the older family would continue to be produced until June 2001 for desktop units,[3] September 2001 for mobile units,[4] and the end of 2003 for embedded devices.[1]

Pentium II

Original Pentium II MMX Case Badge

General information

Launched	May 7, 1997
Discontinued	December 26, 2003[1]
Common manufacturer(s)	Intel

Overview [edit]

The Pentium II microprocessor was largely based upon the microarchitecture of its

Filter tags

2.1.74
2.1.73
2.1.72
2.1.71
2.1.70
2.1.69
2.1.68
2.1.68pre1
2.1.67
2.1.66
2.1.65
2.1.64
2.1.63
2.1.62
2.1.61
2.1.60
2.1.59
2.1.58
2.1.57
2.1.56
2.1.56pre1
2.1.55
2.1.55pre1
2.1.54
2.1.53
2.1.52
2.1.51
2.1.51pre1
2.1.50
2.1.49
2.1.49pre1
2.1.48
2.1.48pre4
2.1.48pre3
2.1.48pre2
2.1.48pre1

```
#ifndef __NET_PKT_SCHED_H
#define __NET_PKT_SCHED_H

#include <linux/pkt_sched.h>

struct Qdisc_ops
{
        struct Qdisc_ops        *next;
        char                    id[IFNAMSIZ];
        int                     refcnt;
        int                     priv_size;
        int                     (*enqueue)(struct sk_buff *skb, struct Qdisc *);
        struct sk_buff *        (*dequeue)(struct Qdisc *);
        void                    (*reset)(struct Qdisc *);
        void                    (*destroy)(struct Qdisc *);
        int                     (*init)(struct Qdisc *, void *arg);
        int                     (*control)(struct Qdisc *, void *);
};

struct Qdisc_head
{
        struct Qdisc_head *forw;
};

extern struct Qdisc_head qdisc_head;

struct Qdisc
{
        struct Qdisc_head       h;
        int                     (*enqueue)(struct sk_buff *skb, struct Qdisc *dev);
        struct sk_buff *        (*dequeue)(struct Qdisc *dev);
        struct Qdisc_ops        *ops;
        int                     handle;
        struct Qdisc            *parent;
        struct sk_buff_head     q;
        struct device           *dev;
        struct sk_buff_head     failure_q;
        unsigned long           dropped;
        unsigned long           tx_last;
        unsigned long           tx_timeo;

        char                    data[0];
```

From: John Fastabend <john.fastabend@gmail.com>
To: daniel@iogearbox.net, eric.dumazet@gmail.com, jhs@mojatatu.com,
aduyck@mirantis.com, brouer@redhat.com, davem@davemloft.net
Cc: john.r.fastabend@intel.com, netdev@vger.kernel.org,
john.fastabend@gmail.com
Subject: [RFC PATCH 03/12] net: sched: allow qdiscs to handle locking
Date: Wed, 30 Dec 2015 09:51:59 -0800 [thread overview]
Message-ID: <20151230175159.26257.51130.stgit@john-Precision-Tower-5810> (raw)
In-Reply-To: <20151230175000.26257.41532.stgit@john-Precision-Tower-5810>

This patch adds a flag for queueing disciplines to indicate
the stack does not need to use the qdisc lock to protect
operations. This can be used to build lockless scheduling
algorithms and improving performance.

The flag is checked in the tx path and the qdisc lock is
only taken if it is not set. For now use a conditional
if statement. Later we could be more aggressive if it
proves worthwhile and use a static key or wrap this in
a likely().

Signed-off-by: John Fastabend <john.r.fastabend@intel.com>
---
include/net/sch_generic.h | 1 +
net/core/dev.c | 20 ++++++++++++--------
net/sched/sch_generic.c | 7 +++++--
3 files changed, 18 insertions(+), 10 deletions(-)

diff --git a/include/net/sch_generic.h b/include/net/sch_generic.h
index b2a8e63..c8d42c3 100644

> Sure. Seems they crept in over time. I had some plans to write a
> lockless HTB implementation. But with fq+EDT with BPF it seems that
> it is no longer needed, we have a more generic/better solution. So
> I dropped it. Also most folks should really be using fq, fq_codel,
> etc. by default anyways. Using pfifo_fast alone is not ideal IMO.

Half a year later, we still have the NOLOCK implementation
present, and pfifo_fast still does set the TCQ_F_NOLOCK flag on itself.

And we've just been bitten by this very same race which appears to be
still unfixed, with single packet being stuck in pfifo_fast qdisc
basically indefinitely due to this very race that this whole thread began
with back in 2019.

Unless there are

        (a) any nice ideas how to solve this in an elegant way without
            (re-)introducing extra spinlock (Cong's fix) or

        (b) any objections to revert as per the argumentation above

I'll be happy to send a revert of the whole NOLOCK implementation next
week.

Thanks,

--
Jiri Kosina
SUSE Labs

From: David Laight <David.Laight@ACULAB.COM>
To: 'Jakub Kicinski' <kuba@kernel.org>
Cc: "netdev@vger.kernel.org" <netdev@vger.kernel.org>,
Vladimir Oltean <vladimir.oltean@nxp.com>,
"'linyunsheng@huawei.com'" <linyunsheng@huawei.com>
Subject: RE: rawip: delayed and mis-sequenced transmits
Date: Thu, 7 Jul 2022 09:34:36 +0000 [thread overview]
Message-ID: <20b3f85b4fa24f8f86ea479383580eed@AcuMS.aculab.com> (raw)
In-Reply-To: <20220706185417.2fcbcdf0@kernel.org>

From: Jakub Kicinski
> Sent: 07 July 2022 02:54
>
> On Wed, 6 Jul 2022 15:54:18 +0000 David Laight wrote:
> > Anyone any ideas before I start digging through the kernel code?
>
> If the qdisc is pfifo_fast and kernel is old there could be races.
> But I don't think that's likely given you probably run something
> recent and next packet tx would usually flush the stuck packet.
> In any case - switching qdisc could be a useful test, also bpftrace
> is your friend for catching patckets with long sojourn time.

Reading the sources I think I've found something:
In core/dev.c line 3818 there is:

```
static inline int __dev_xmit_skb(struct sk_buff *skb, struct Qdisc *q,
                                 struct net_device *dev,
                                 struct netdev_queue *txq)
{
        spinlock_t *root_lock = qdisc_lock(q);
```

Obecná doporučení

- Pokud chci server na Linuxu:
  - pokud můžu použiju výchozí **mq** (výchozí) anebo **noqueue** qdisc
  - **pokud možno v nastavení plánovače paketů nic nevylepšovat**
  - maximálně využít offload kary
  - HyperThreading – spíše ano, ale… záleží podle situace
  - Rate limiting spojení/paketů atd. ideálně přesunout na aplikační vrstvu
- Pokud chci router na Linuxu:
  - pokud můžu použiju výchozí **mq** (výchozí) anebo **noqueue** qdisc
  - při výběru HW upřednostním menší počet jader a vyšší taktovací frekvenci CPU
  - vyhnu se použití vlan interfejsu – interně implementován jako single queue
  - HyperThreading – spíše ne, ale… záleží podle situace
- Pokud chci router na MikroTiku
  - přednostně využiju ideálně pouze **only-hardware-queue**
  - vyhnu se použití queue tree
  - když už musím dělat shaping zkusím raději využít simple queue