e16
100GbE テスト

千代浩司
KEK/IPNS
2023-05-24
2023-06-08

セットアップ

- 2023/05/17 東海２号館204実験室搬入（四日市さん、市川さん）
- 2023/05/17 JLAN intra 接続申請
- 2023/05/23 JLAN intra 接続きょか
- 機材
  - Dell PowerEdge T430
  - ASUS ESC 4000A
  - FS.com 100GbEスイッチ


仮置き
ハドロン Exp
千代 2023-6-10迄

アイドル時の消費電力測定

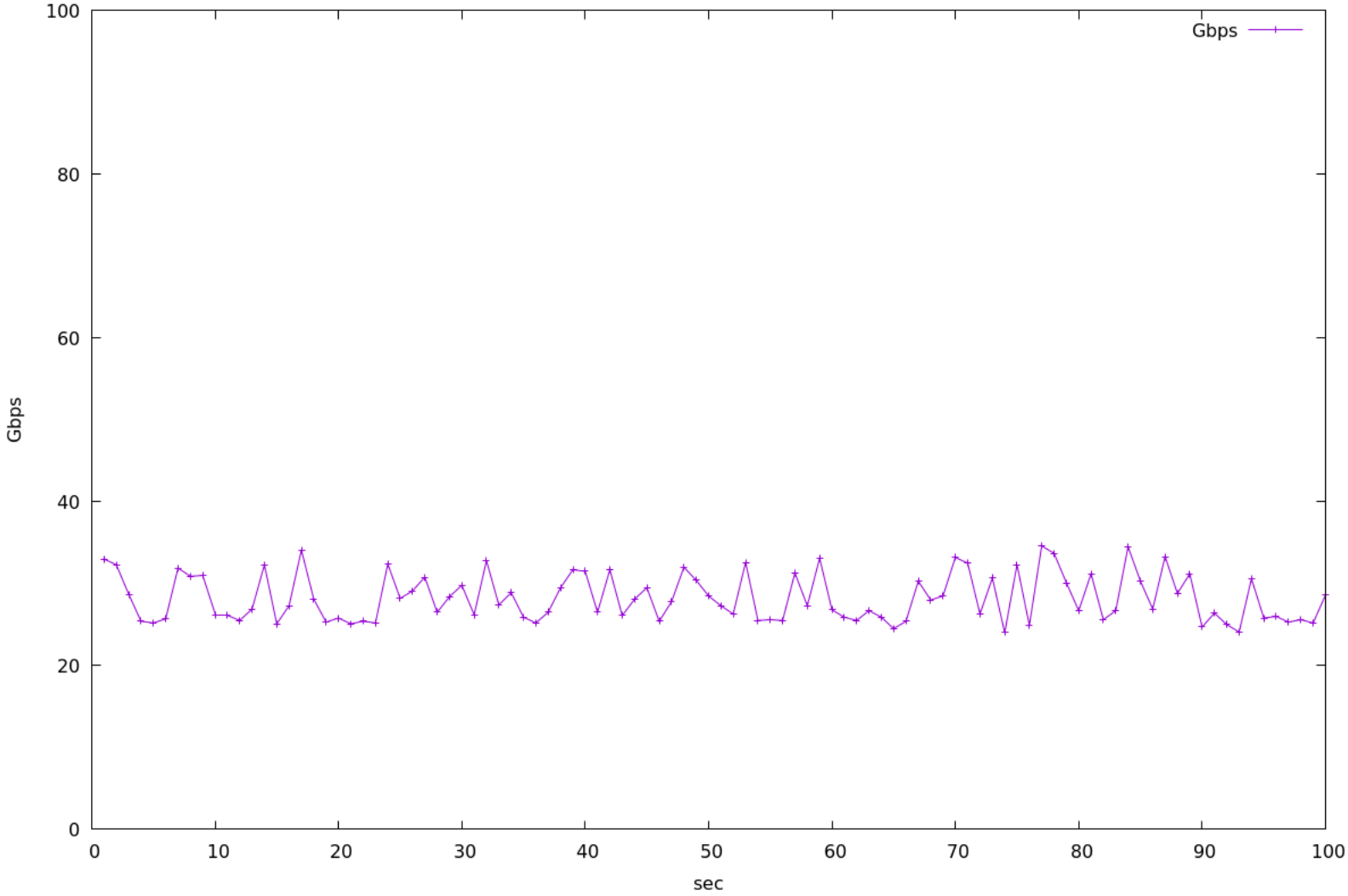
- 五十嵐さんからmeterk MK3USを拝借
- Dell PowerEdge
  - Power Supply #1 (裏面向かって左):
    OS起動後アイドル時 200 W
  - Power Supply #2 (裏面向かって右):
    OS起動後アイドル時　 0 W
- ASUS 2Uサーバー
  - Power Supply #1 (裏面向かって左):
    OS起動後アイドル時 80W
  - Power Supply #2 (裏面向かって右):
    OS起動後アイドル時 80W

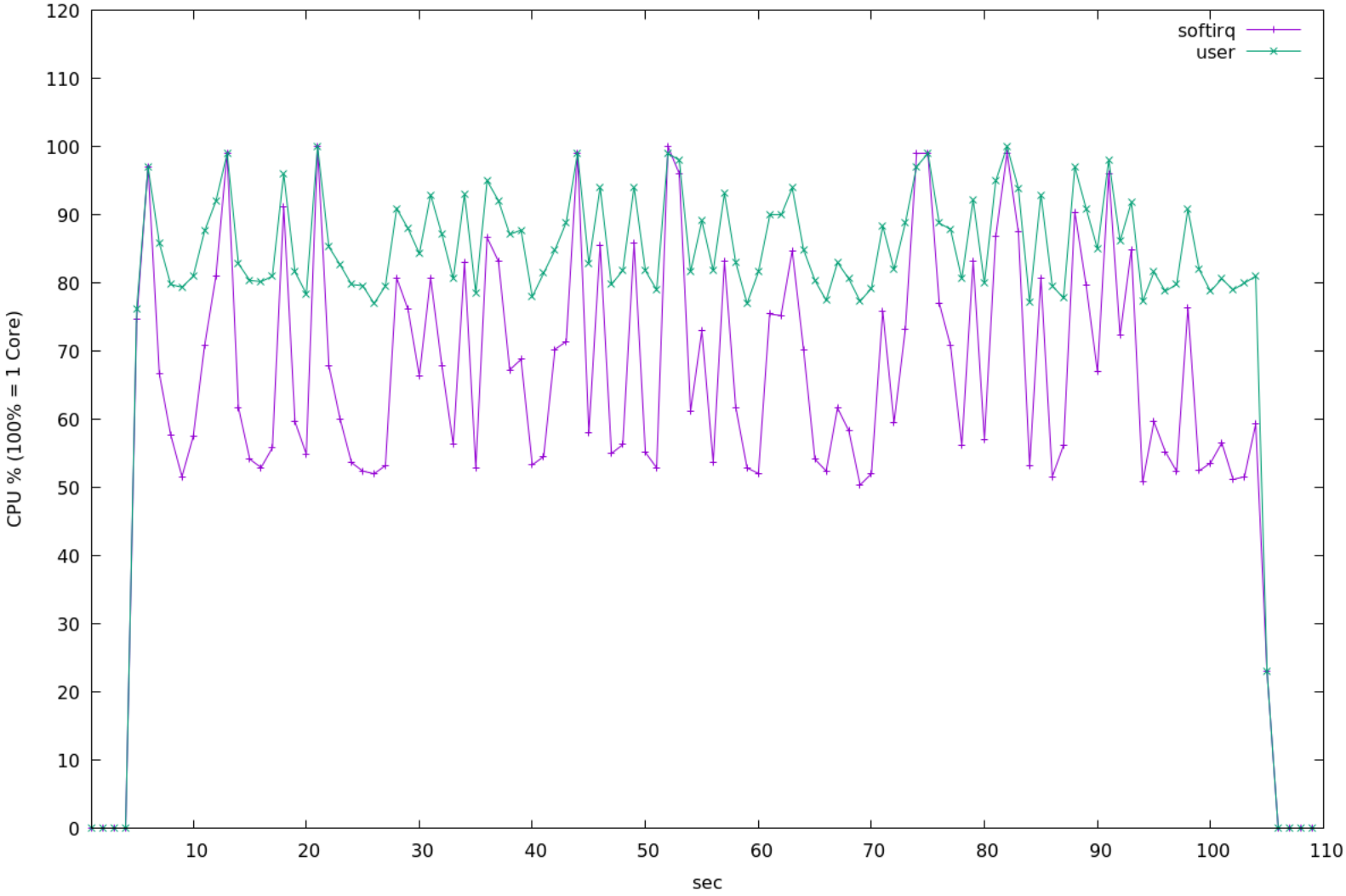
画像出典
https://diy.senka.biz/maker/Meterk/2541712051/B07CNW48JR.html

OS, スイッチ設定

- OS: AlmaLinux 9.1 / 9.2
  - ASUS 2UサーバーがJLAN intra接続前に電源をいれたらshutdown時にアップデートが走り9.2に
- スイッチの設定
  - ケーブルを接続しただけではリンクアップしなかった
  - スイッチマネージメントポートにLANケーブルをさし、http://192.168.1.1/ にアクセス (admin/admin)
  - ポートスピードでautoから100GbEに変更
  - 保存、スイッチリスタート
  - リンクアップした

```
[root@e16evb03 ~]# ethtool enp2s0f0np0
Settings for enp2s0f0np0:
    Supported ports: [ Backplane ]
    Supported link modes:   1000baseT/Full
(略)
 Advertised pause frame use: Symmetric
    Advertised auto-negotiation: Yes
    Advertised FEC modes: None
    Speed: 100000Mb/s
    Duplex: Full
    Auto-negotiation: on
    Port: Direct Attach Copper
    PHYAD: 0
    Transceiver: internal
    Supports Wake-on: d
    Wake-on: d
        Current message level: 0x00000004 (4)
                               link
    Link detected: yes
```


Gbps
100
80
60
40
20
0
Gbps
0
10
20
30
40
50
60
70
80
90
100
sec

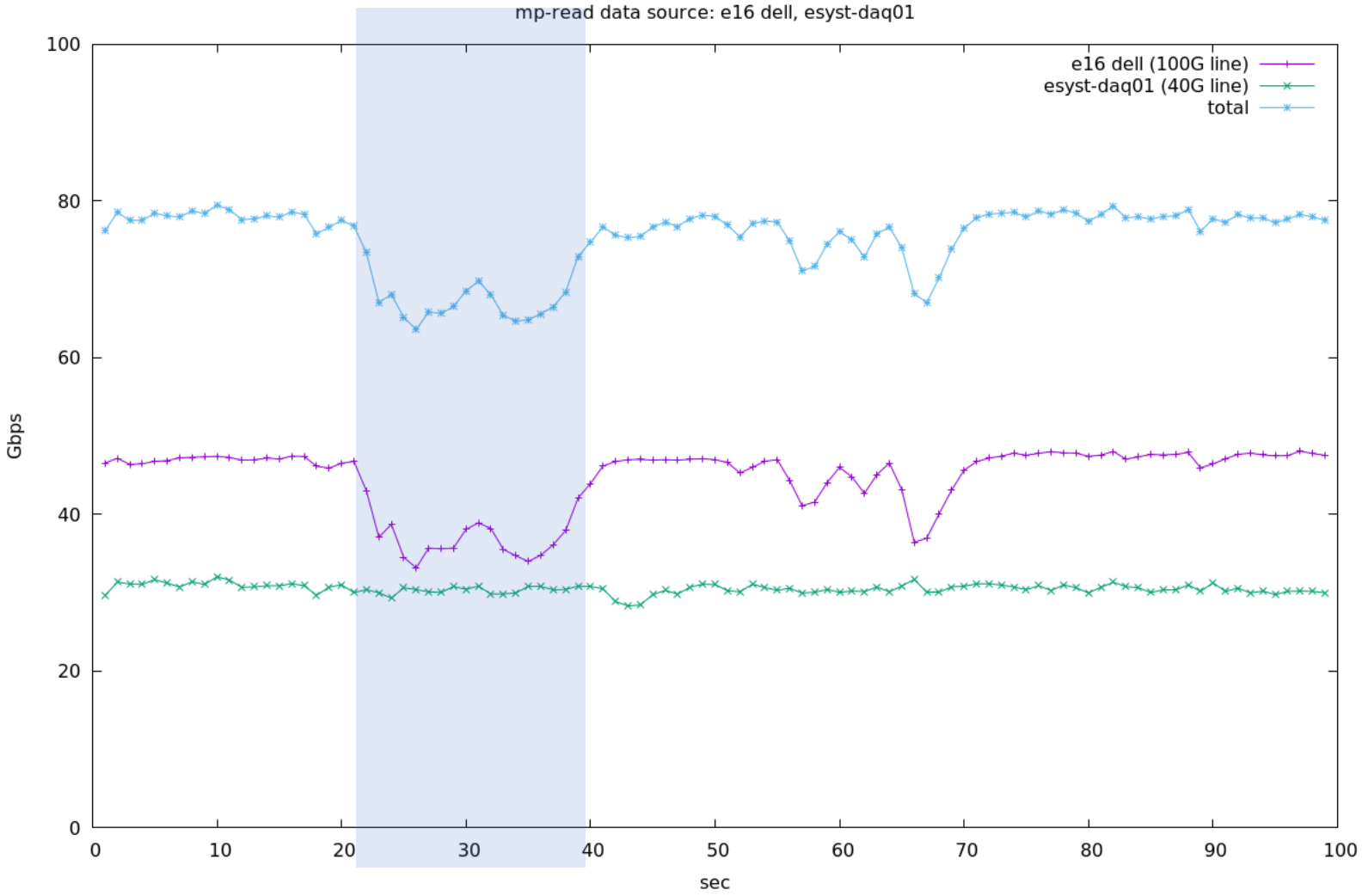

softirq
user
CPU % (100% = 1 Core)
sec
0
10
20
30
40
50
60
70
80
90
100
110
120

これから

- まずは100GbE <-> 100GbEでiperf3で測定
- 100Gb <-> (40 Gb) x 2 で測定

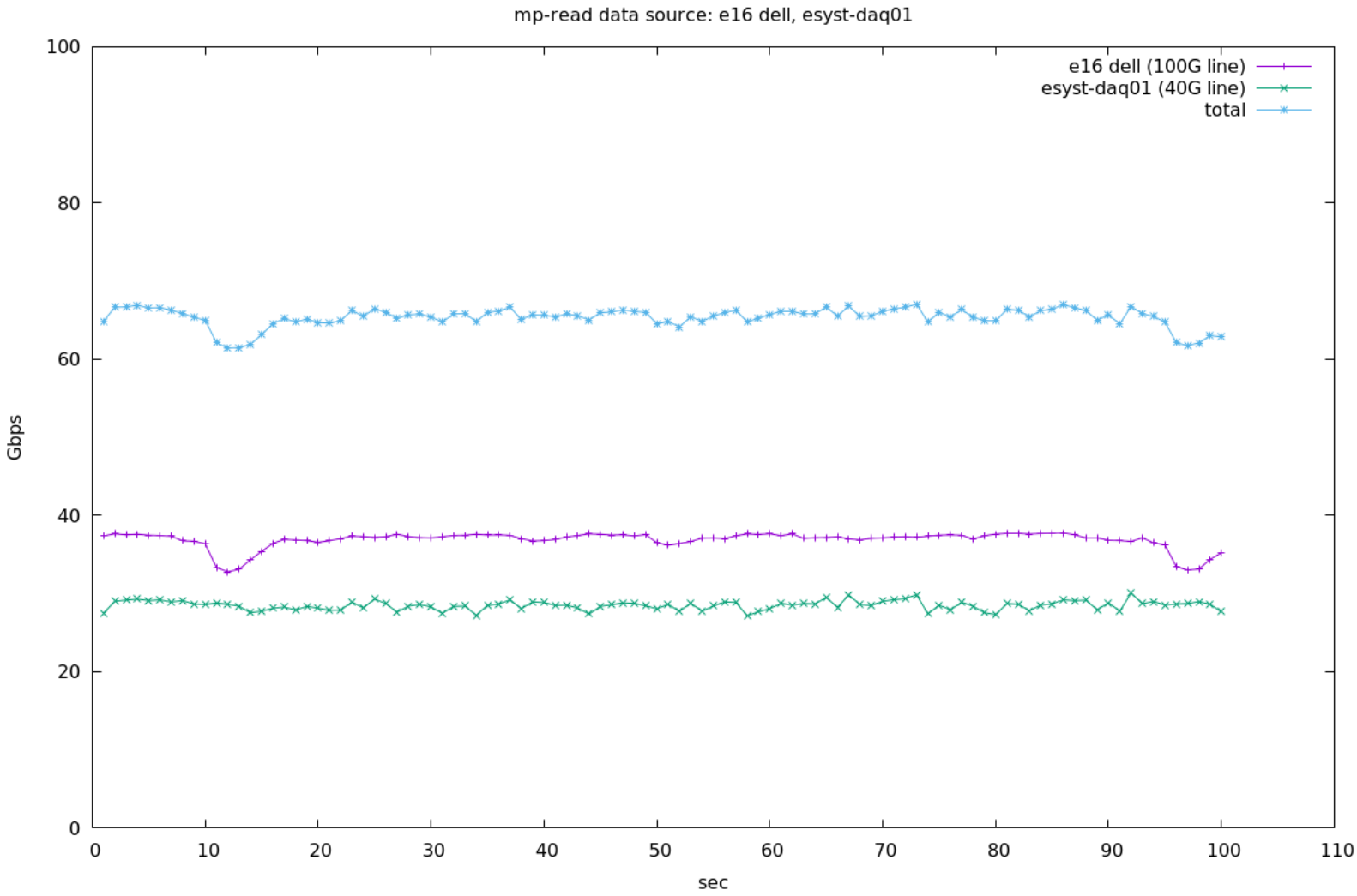
- このスライドまで2023-05-24 E16 elec. meeting で発表

run #1



多分このへんで稼働しているCPUコアがかわった？

run #2



データ生成サーバー、読み出しプログラムのCPUをtasksetですべて固定してみた

今回のテスト

- テストの目的
  - スイッチが100Gb/sでデータ転送できるか
  - TCPで計測するとイーサネットヘッダ、トレーラー、フレーム間ギャップ、IPヘッダ、TCPヘッダ、TCP timestampsオプションで100Gb/s * 0.941になる
- まずは1TCPセッションで安定して通信できることをめざす

SMT (Simultaneous Multi Threading) (Hyper Threading)をoffにする

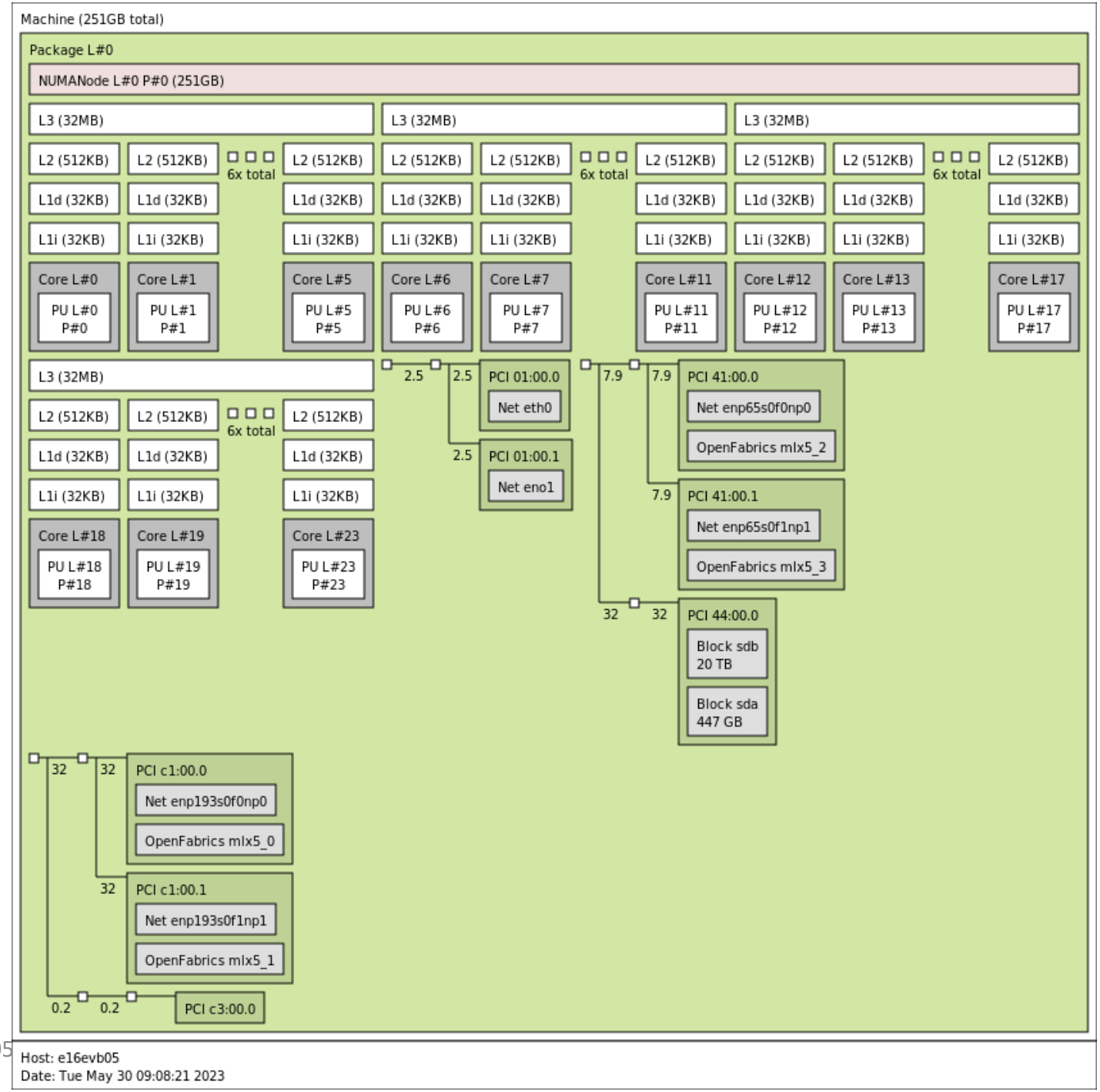
- CPU数がありすぎて（私が）テストで混乱する
- いくつかスループットテストをみたがhyperthreadをオフにする、と書いてあるところが散見される

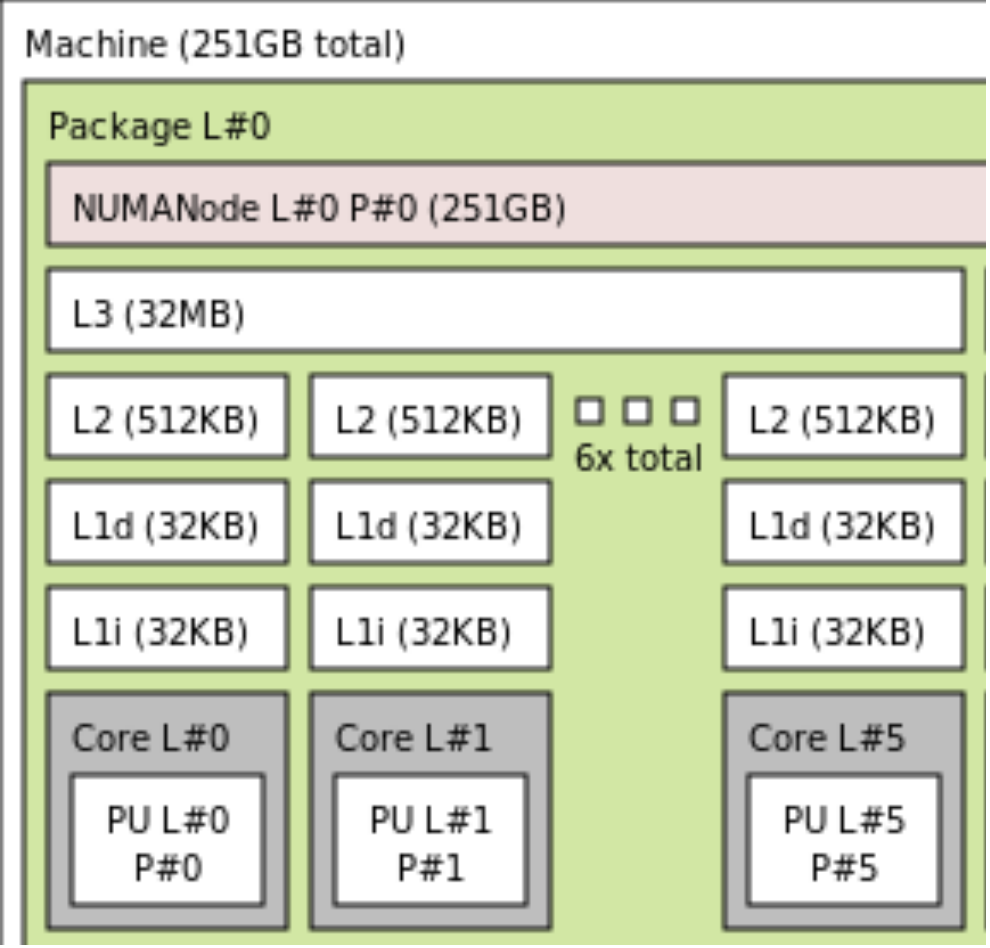
- OFFにする方法いろいろ
  - BIOSあるいはUEFIでOFFにする
  - 起動時のカーネル引数にnosmtを追加する(今回はこれをつかった)
    - grub画面でeをおして編集、編集後C-x。grub.confには保存はされない
  - echo off > /sys/devices/system/cpu/smt/controlを実行する
  - echo 0 > /sys/devices/system/cpu/cpuN/onlineを必要な数だけ繰り返す。NにHyperThreadに対応するCPUコア番号を入れる。

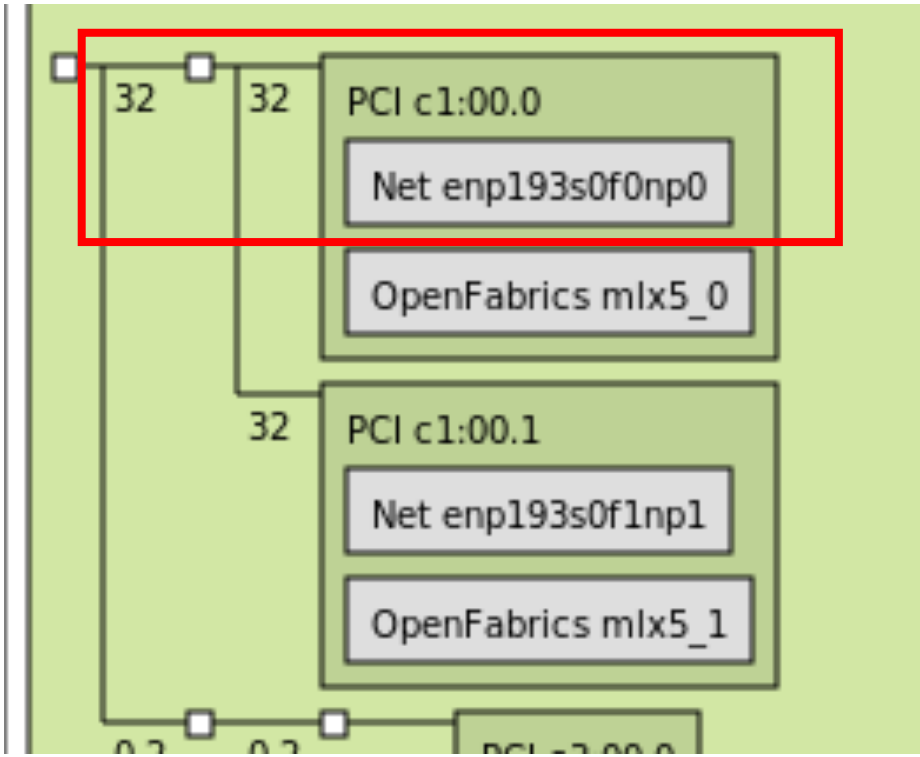
ハードウェアの把握
hwloc-ls, lstopo

- hwloc-ls でテキスト出力
- lstopo machine.png でmachine.pngという画像ファイルができる
- lstopo machine.pdfでpdfファイルができる

ASUS
ESC4000







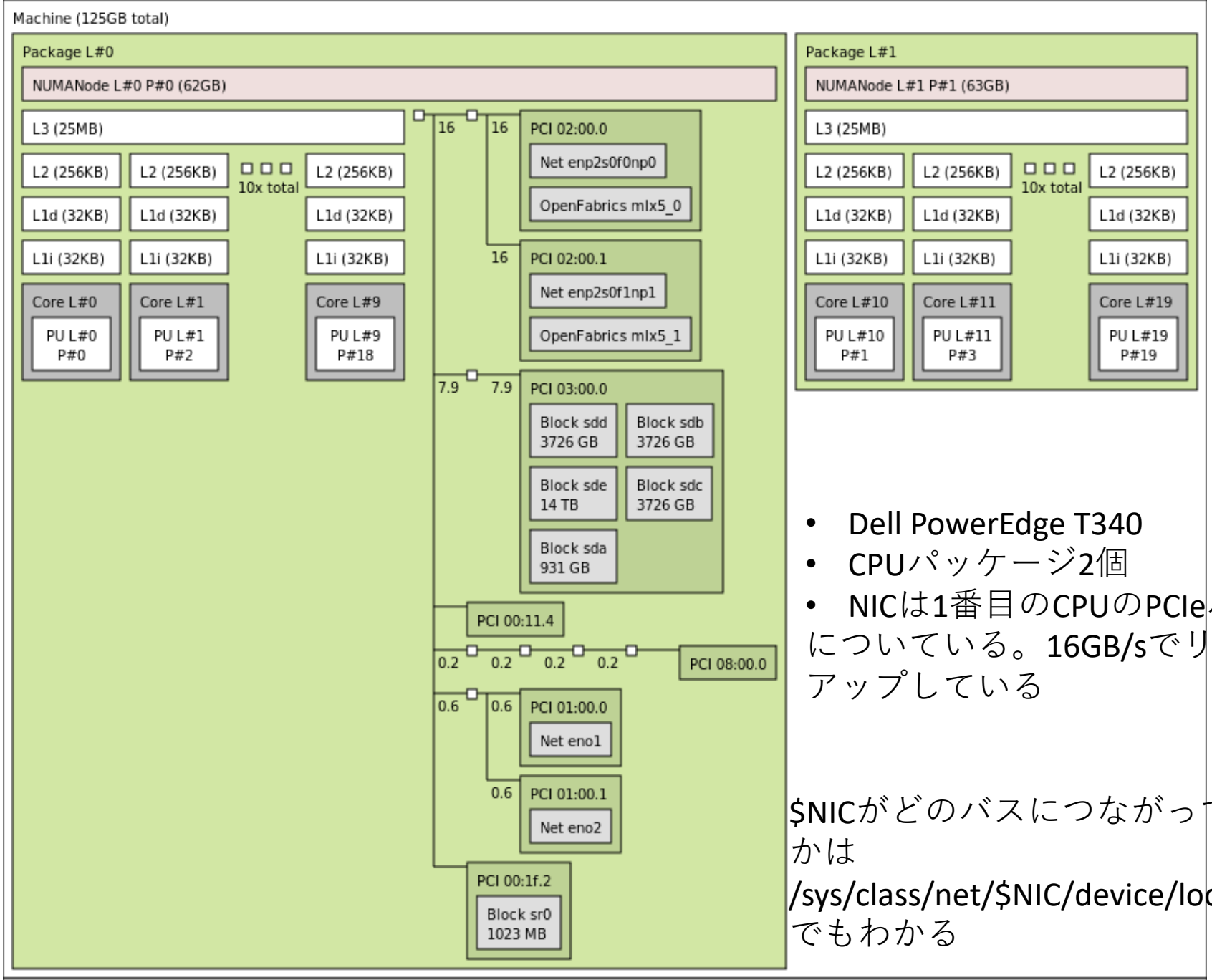
このPCではL3を 6 コアで共有する

NIC PCIe 32GB/sでリンクアップしている

```
Machine (251GB total) + Package L#0
 NUMANode L#0 (P#0 251GB)
 L3 L#0 (32MB)
   L2 L#0 (512KB) + L1d L#0 (32KB) + L1i L#0 (32KB) + Core L#0 + PU L#0 (P#0)
   L2 L#1 (512KB) + L1d L#1 (32KB) + L1i L#1 (32KB) + Core L#1 + PU L#1 (P#1)
   L2 L#2 (512KB) + L1d L#2 (32KB) + L1i L#2 (32KB) + Core L#2 + PU L#2 (P#2)
   L2 L#3 (512KB) + L1d L#3 (32KB) + L1i L#3 (32KB) + Core L#3 + PU L#3 (P#3)
   L2 L#4 (512KB) + L1d L#4 (32KB) + L1i L#4 (32KB) + Core L#4 + PU L#4 (P#4)
   L2 L#5 (512KB) + L1d L#5 (32KB) + L1i L#5 (32KB) + Core L#5 + PU L#5 (P#5)
 L3 L#1 (32MB)
   L2 L#6 (512KB) + L1d L#6 (32KB) + L1i L#6 (32KB) + Core L#6 + PU L#6 (P#6)
   L2 L#7 (512KB) + L1d L#7 (32KB) + L1i L#7 (32KB) + Core L#7 + PU L#7 (P#7)
   L2 L#8 (512KB) + L1d L#8 (32KB) + L1i L#8 (32KB) + Core L#8 + PU L#8 (P#8)
   L2 L#9 (512KB) + L1d L#9 (32KB) + L1i L#9 (32KB) + Core L#9 + PU L#9 (P#9)
   L2 L#10 (512KB) + L1d L#10 (32KB) + L1i L#10 (32KB) + Core L#10 + PU L#10 (P#10)
   L2 L#11 (512KB) + L1d L#11 (32KB) + L1i L#11 (32KB) + Core L#11 + PU L#11 (P#11)
 L3 L#2 (32MB)
   L2 L#12 (512KB) + L1d L#12 (32KB) + L1i L#12 (32KB) + Core L#12 + PU L#12 (P#12)
   L2 L#13 (512KB) + L1d L#13 (32KB) + L1i L#13 (32KB) + Core L#13 + PU L#13 (P#13)
   L2 L#14 (512KB) + L1d L#14 (32KB) + L1i L#14 (32KB) + Core L#14 + PU L#14 (P#14)
   L2 L#15 (512KB) + L1d L#15 (32KB) + L1i L#15 (32KB) + Core L#15 + PU L#15 (P#15)
   L2 L#16 (512KB) + L1d L#16 (32KB) + L1i L#16 (32KB) + Core L#16 + PU L#16 (P#16)
   L2 L#17 (512KB) + L1d L#17 (32KB) + L1i L#17 (32KB) + Core L#17 + PU L#17 (P#17)
 L3 L#3 (32MB)
   L2 L#18 (512KB) + L1d L#18 (32KB) + L1i L#18 (32KB) + Core L#18 + PU L#18 (P#18)
   L2 L#19 (512KB) + L1d L#19 (32KB) + L1i L#19 (32KB) + Core L#19 + PU L#19 (P#19)
   L2 L#20 (512KB) + L1d L#20 (32KB) + L1i L#20 (32KB) + Core L#20 + PU L#20 (P#20)
   L2 L#21 (512KB) + L1d L#21 (32KB) + L1i L#21 (32KB) + Core L#21 + PU L#21 (P#21)
   L2 L#22 (512KB) + L1d L#22 (32KB) + L1i L#22 (32KB) + Core L#22 + PU L#22 (P#22)
   L2 L#23 (512KB) + L1d L#23 (32KB) + L1i L#23 (32KB) + Core L#23 + PU L#23 (P#23)
```

```
HostBridge
  PCIBridge
    PCI 01:00.0 (Ethernet)
      Net "eth0"
    PCI 01:00.1 (Ethernet)
      Net "eno1"
 HostBridge
  PCIBridge
    PCI 41:00.0 (Ethernet)
      Net "enp65s0f0np0"
      OpenFabrics "mlx5_2"
    PCI 41:00.1 (Ethernet)
      Net "enp65s0f1np1"
      OpenFabrics "mlx5_3"
  PCIBridge
    PCI 44:00.0 (SATA)
      Block(Disk) "sdb"
      Block(Disk) "sda"
 HostBridge
  PCIBridge
    PCI c1:00.0 (Ethernet)
      Net "enp193s0f0np0"
      OpenFabrics "mlx5_0"
    PCI c1:00.1 (Ethernet)
      Net "enp193s0f1np1"
      OpenFabrics "mlx5_1"
  PCIBridge
    PCIBridge
      PCI c3:00.0 (VGA)
```

P#NがtasksetーcNで指定するCPUコア番号



- Dell PowerEdge T340
- CPUパッケージ2個
- NICは1番目のCPUのPCIeバスについている。16GB/sでリンクアップしている

$NICがどのバスにつながっているかは
/sys/class/net/$NIC/device/local_cpulist
でもわかる

```
Machine (125GB total)
  Package L#0
    NUMANode L#0 (P#0 62GB)
    L3 L#0 (25MB)
      L2 L#0 (256KB) + L1d L#0 (32KB) + L1i L#0 (32KB) + Core L#0 + PU L#0 (P#0)
      L2 L#1 (256KB) + L1d L#1 (32KB) + L1i L#1 (32KB) + Core L#1 + PU L#1 (P#2)
      L2 L#2 (256KB) + L1d L#2 (32KB) + L1i L#2 (32KB) + Core L#2 + PU L#2 (P#4)
      L2 L#3 (256KB) + L1d L#3 (32KB) + L1i L#3 (32KB) + Core L#3 + PU L#3 (P#6)
      L2 L#4 (256KB) + L1d L#4 (32KB) + L1i L#4 (32KB) + Core L#4 + PU L#4 (P#8)
      L2 L#5 (256KB) + L1d L#5 (32KB) + L1i L#5 (32KB) + Core L#5 + PU L#5 (P#10)
      L2 L#6 (256KB) + L1d L#6 (32KB) + L1i L#6 (32KB) + Core L#6 + PU L#6 (P#12)
      L2 L#7 (256KB) + L1d L#7 (32KB) + L1i L#7 (32KB) + Core L#7 + PU L#7 (P#14)
      L2 L#8 (256KB) + L1d L#8 (32KB) + L1i L#8 (32KB) + Core L#8 + PU L#8 (P#16)
      L2 L#9 (256KB) + L1d L#9 (32KB) + L1i L#9 (32KB) + Core L#9 + PU L#9 (P#18)
    HostBridge
      PCIBridge
        PCI 02:00.0 (Ethernet)
          Net "enp2s0f0np0"
          OpenFabrics "mlx5_0"
        PCI 02:00.1 (Ethernet)
          Net "enp2s0f1np1"
          OpenFabrics "mlx5_1"
      PCIBridge
        PCI 03:00.0 (RAID)
          Block(Disk) "sdd"
          Block(Disk) "sdb"
          Block(Disk) "sde"
          Block(Disk) "sdc"
          Block(Disk) "sda"
      PCI 00:11.4 (SATA)
      PCIBridge
        PCIBridge
          PCIBridge
            PCIBridge
              PCI 08:00.0 (VGA)
      PCIBridge
        PCI 01:00.0 (Ethernet)
          Net "eno1"
        PCI 01:00.1 (Ethernet)
          Net "eno2"
      PCI 00:1f.2 (SATA)
        Block(Removable Media Device) "sr0"
  Package L#1
    NUMANode L#1 (P#1 63GB)
    L3 L#1 (25MB)
      L2 L#10 (256KB) + L1d L#10 (32KB) + L1i L#10 (32KB) + Core L#10 + PU L#10 (P#1)
      L2 L#11 (256KB) + L1d L#11 (32KB) + L1i L#11 (32KB) + Core L#11 + PU L#11 (P#3)
      L2 L#12 (256KB) + L1d L#12 (32KB) + L1i L#12 (32KB) + Core L#12 + PU L#12 (P#5)
      L2 L#13 (256KB) + L1d L#13 (32KB) + L1i L#13 (32KB) + Core L#13 + PU L#13 (P#7)
      L2 L#14 (256KB) + L1d L#14 (32KB) + L1i L#14 (32KB) + Core L#14 + PU L#14 (P#9)
      L2 L#15 (256KB) + L1d L#15 (32KB) + L1i L#15 (32KB) + Core L#15 + PU L#15 (P#11)
      L2 L#16 (256KB) + L1d L#16 (32KB) + L1i L#16 (32KB) + Core L#16 + PU L#16 (P#13)
      L2 L#17 (256KB) + L1d L#17 (32KB) + L1i L#17 (32KB) + Core L#17 + PU L#17 (P#15)
      L2 L#18 (256KB) + L1d L#18 (32KB) + L1i L#18 (32KB) + Core L#18 + PU L#18 (P#17)
      L2 L#19 (256KB) + L1d L#19 (32KB) + L1i L#19 (32KB) + Core L#19 + PU L#19 (P#19)
```

読み出しPCでCPUを使うところ

- ユーザープログラム
- ネットワークキューの処理
  - IP, TCPスタックの処理
  - mpstatコマンドでsoftirqの欄にでているもの
  - 1 GbE ではほとんど見えない
  - 10 GbE あたりからみえてくる
    (CPUにもよるがユーザープロセスsys 20%, softirq 15% - 20 %程度)

パケット到着後の処理

- パケットを入れるキューが複数ある（NICによる）
- NICによってキューの最大数（キューの長さではなく）がきまる
  - 例: igb (Intel 1GbE)で８個
  - 今回読み出しに使ったPCでは24個（CPU数と同じ）が出現していた
  - CPUコア数より多く設定されることはない
- 受信パケットがNICによりキューにおかれる
  - 複数あるキューのどれにいれるかきめるロジックがある
  - ユーザーが指定することも可能（後述）
- 一定量たまったらirqでCPUに知らせる
- CPUがIP TCP スタック処理
- キューにはいったパケットを処理する(softirq)CPUコアを指定することができる
  - デフォルトではirqbalanceデーモンが起動していてCPUによって処理数にばらつきがでないようにときどき調整する
  - どのCPUがわりあてられているかは/proc/interruptsからirq番号を取得して/proc/irq/irq番号/smp_affinity_listをみる。ここにecho 3とするとCPU #3がそのキューを処理するようになる

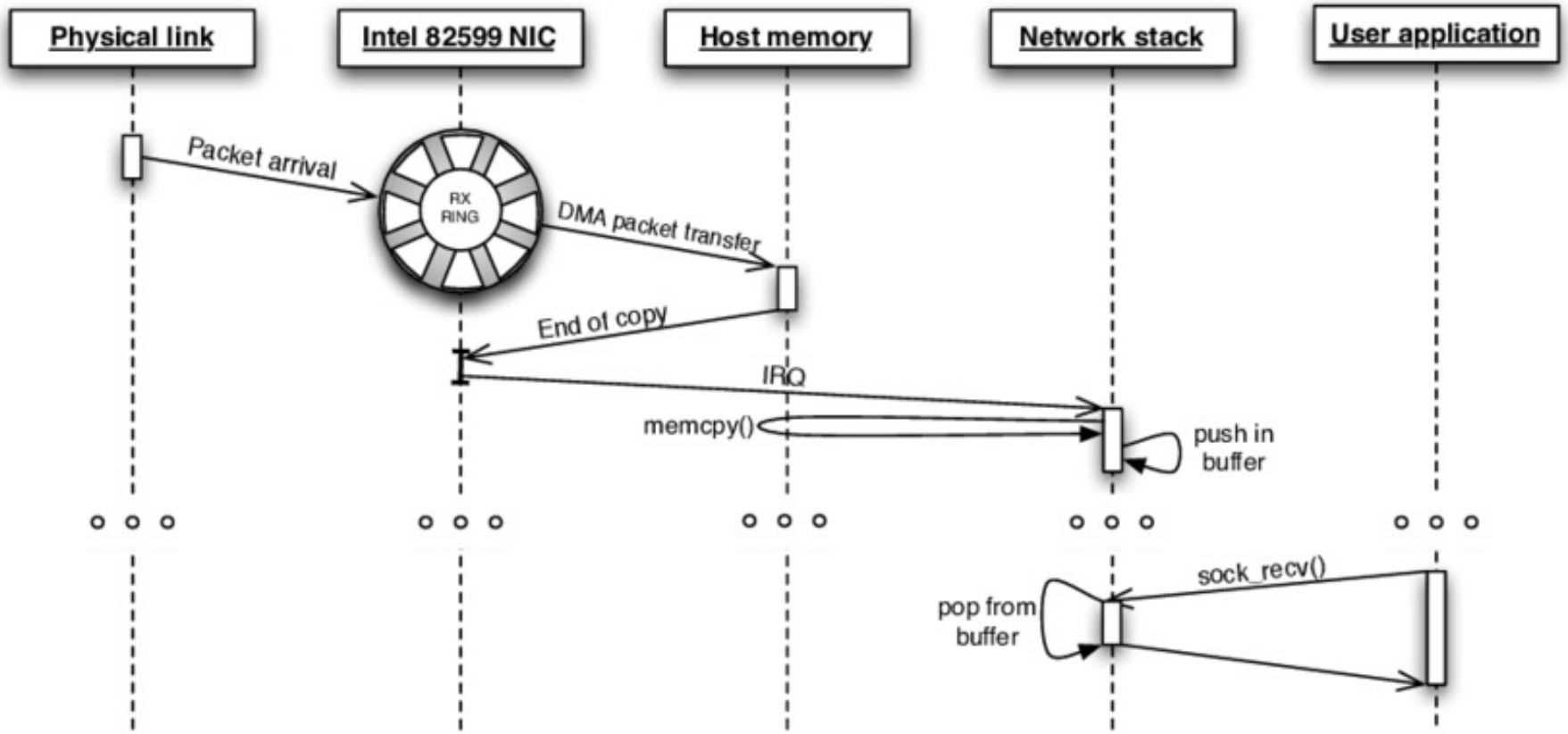
CPU0	CPU1	CPU2	CPU3	CPU4	CPU5	CPU6	CPU7	CPU8	CPU>	
109:	113	0	124	0	0	2	0	0	0	>
110:	0	250581	0	6	0	0	0	0	0	>
111:	0	0	6116633	0	1675288	0	0	0	0	>
112:	0	0	0	289506	0	6	0	0	0	>
113:	459412	0	0	0	1973029	0	0	0	0	>
114:	0	10522154	0	0	0	9197536	0	0	3202892	>
115:	0	0	0	0	0	0	9230935	0	6	>
116:	0	0	0	0	0	0	5305	1468501	0	80305>
117:	0	0	0	0	0	0	0	695511	1096274	>
118:	0	0	0	0	0	0	0	0	0	271369>
119:	0	0	0	0	0	0	8	0	0	>
120:	0	0	0	0	0	0	0	9215573	0	>
121:	0	0	1	0	0	72713	0	6	0	>
122:	0	0	0	0	0	0	0	0	0	>
123:	166981	0	0	0	0	0	0	0	0	>
124:	0	0	0	0	0	0	0	0	0	>
125:	0	0	0	0	0	0	0	0	0	111677>
126:	0	0	0	0	0	0	0	0	0	>
127:	0	0	0	0	0	168850	0	0	0	>
128:	0	0	0	0	8077	0	0	0	0	>
129:	0	0	0	225632	0	0	0	0	0	>
130:	0	0	0	0	0	0	0	0	0	>
131:	0	0	0	0	0	0	0	0	0	>
132:	0	0	0	0	0	0	0	0	0	>
133:	0	0	0	0	0	455717	8841	168	210	18>

横に長いので中略

最初の数字がIRQ番号

最後のがキューの名称

CPU20	CPU21	CPU22	CPU23			
0	0	0	0	IR-PCI-MSI	101187584-edge	mlx5_comp0@pci:0000:c1:00.0
0	0	0	0	IR-PCI-MSI	101187585-edge	mlx5_comp1@pci:0000:c1:00.0
0	0	0	0	IR-PCI-MSI	101187586-edge	mlx5_comp2@pci:0000:c1:00.0
0	0	0	0	IR-PCI-MSI	101187587-edge	mlx5_comp3@pci:0000:c1:00.0
0	0	0	0	IR-PCI-MSI	101187588-edge	mlx5_comp4@pci:0000:c1:00.0
0	0	0	0	IR-PCI-MSI	101187589-edge	mlx5_comp5@pci:0000:c1:00.0
0	0	0	0	IR-PCI-MSI	101187590-edge	mlx5_comp6@pci:0000:c1:00.0
0	0	0	0	IR-PCI-MSI	101187591-edge	mlx5_comp7@pci:0000:c1:00.0
0	0	0	0	IR-PCI-MSI	101187592-edge	mlx5_comp8@pci:0000:c1:00.0
0	0	0	0	IR-PCI-MSI	101187593-edge	mlx5_comp9@pci:0000:c1:00.0
0	0	0	0	IR-PCI-MSI	101187594-edge	mlx5_comp10@pci:0000:c1:00.0
0	0	0	0	IR-PCI-MSI	101187595-edge	mlx5_comp11@pci:0000:c1:00.0
11	17	0	0	IR-PCI-MSI	101187596-edge	mlx5_comp12@pci:0000:c1:00.0
366089	0	0	0	IR-PCI-MSI	101187597-edge	mlx5_comp13@pci:0000:c1:00.0
0	0	0	0	IR-PCI-MSI	101187598-edge	mlx5_comp14@pci:0000:c1:00.0
0	1	0	0	IR-PCI-MSI	101187599-edge	mlx5_comp15@pci:0000:c1:00.0
0	0	0	0	IR-PCI-MSI	101187600-edge	mlx5_comp16@pci:0000:c1:00.0
0	0	0	0	IR-PCI-MSI	101187601-edge	mlx5_comp17@pci:0000:c1:00.0
0	0	0	0	IR-PCI-MSI	101187602-edge	mlx5_comp18@pci:0000:c1:00.0
0	0	0	0	IR-PCI-MSI	101187603-edge	mlx5_comp19@pci:0000:c1:00.0
7	0	0	0	IR-PCI-MSI	101187604-edge	mlx5_comp20@pci:0000:c1:00.0
0	2	728919	0	IR-PCI-MSI	101187605-edge	mlx5_comp21@pci:0000:c1:00.0
0	0	2	331080	IR-PCI-MSI	101187606-edge	mlx5_comp22@pci:0000:c1:00.0
0	41	0	3	IR-PCI-MSI	101187607-edge	mlx5_comp23@pci:0000:c1:00.0
8459	21868	0	0	IR-PCI-MSI	101187608-edge	mlx5_async24@pci:0000:c1:00.0



https://medium.com/coccoc-engineering-blog/linux-network-ring-buffers-cea7ead0b8e8

RX ringは一つしかかいていないが実際は複数ある(NICによる)

受信デスクリプタキューの構造 Intel NIC datasheet

- NIC上のレジスタ
  - Receive Descriptor Base Address registers (RDBA0, RDBA1) 64ビット
    ホストPC上のメモリデスクリプタリングバッファの開始位置を示す
  - Receive Descriptor Length Registers (RDLEN0, RDLEN1)
    単位：バイト。128の倍数。128はキャッシュラインの最大値。デスクリプタは16バイトなので128/16=8でこの値は8の倍数になる (ethtool –g/-G $nicで取得・セットできるやつ。ethtool –g $nicの出力数字の単位は個)
  - Receive Descriptor Head Registers (RDH0, RDH1)
    デスクリプタ総数は64kBなので64kB/16B=4096個のデスクリプタを保持できる
  - Receive Descriptor Tail Registers (RDT0, RDT1)

レジスタダンプ

- ethtool –d $nic (e1000eの例)

```
% sudo ethtool -d exp0
MAC Registers
-------------
0x00000: CTRL (Device control register)  0x580C0241
      Endian mode (buffers):             little
      Link reset:                        normal
      Set link up:                       1
      Invert Loss-Of-Signal:             no
      Receive flow control:              enabled
      Transmit flow control:             enabled
      VLAN mode:                         enabled
      Auto speed detect:                 disabled
      Speed select:                      1000Mb/s
      Force speed:                       no
      Force duplex:                      no
0x00008: STATUS (Device status register) 0x00080383
      Duplex:                            full
      Link up:                           link config
      TBI mode:                          disabled
      Link speed:                        1000Mb/s
      Bus type:                          PCI Express
      Port number:                       0
0x00100: RCTL (Receive control register) 0x04008002
      Receiver:                          enabled
      Store bad packets:                 disabled
      Unicast promiscuous:               disabled
      Multicast promiscuous:             disabled
      Long packet:                       disabled
      Descriptor minimum threshold size: 1/2
      Broadcast accept mode:             accept
      VLAN filter:                       disabled
      Canonical form indicator:          disabled
      Discard pause frames:              filtered
      Pass MAC control frames:           don't pass
      Receive buffer size:               2048
```

```
0x02808: RDLEN (Receive desc length)     0x00001000
0x02810: RDH   (Receive desc head)       0x00000040
0x02818: RDT   (Receive desc tail)       0x00000030
0x02820: RDTR  (Receive delay timer)     0x00000020
0x00400: TCTL (Transmit ctrl register)   0x3103F0FA
      Transmitter:                       enabled
      Pad short packets:                 enabled
      Software XOFF Transmission:        disabled
      Re-transmit on late collision:     enabled
0x03808: TDLEN (Transmit desc length)    0x00001000
0x03810: TDH   (Transmit desc head)      0x00000045
0x03818: TDT   (Transmit desc tail)      0x00000045
0x03820: TIDV  (Transmit delay timer)    0x00000008
PHY type:                                unknown
```

7.1.2.3 Queue Configuration Registers

Configuration registers (CSRs) that control queue operation are replicated per queue (total of 8 copies of each register per port). Each of the replicated registers correspond to a queue such that the queue index equals the serial number of the register (such as register 0 corresponds to queue 0, etc.). Registers included in this category are:

- *RDBAL* and *RDBAH* — Rx Descriptor Base
- *RDLEN* — RX Descriptor Length
- *RDH* — RX Descriptor Head
- *RDT* — RX Descriptor Tail
- *RXDCTL* — Receive Descriptor Control
- *RXCTL* — Rx DCA Control
- *SRRCTL* — Split and Replication Receive Control

Intel I350 (igb)
queueは8個

The receive descriptor rings are described by the following registers:

- Receive Descriptor Base Address (*RDBA7* to *RDBA0*) register:
  This register indicates the start of the descriptor ring buffer. This 64-bit address is aligned on a 16-byte boundary and is stored in two consecutive 32-bit registers. Note that hardware ignores the lower 4 bits.
- Receive Descriptor Length (*RDLEN7* to *RDLEN0*) registers:
  This register determines the number of bytes allocated to the circular buffer. This value must be a multiple of 128 (the maximum cache-line size). Since each descriptor is 16 bytes in length, the total number of receive descriptors is always a multiple of eight.
- Receive Descriptor Head (*RDH7* to *RDH0*) registers:
  This register holds a value that is an offset from the base and indicates the in-progress descriptor. There can be up to 64 KB, 8 KB descriptors in the circular buffer. Hardware maintains a shadow copy that includes those descriptors completed but not yet stored in memory.
- Receive Descriptor Tail (*RDT7* to *RDT0*) registers:
  This register holds a value that is an offset from the base and identifies the location beyond the last descriptor hardware can process. This is the location where software writes the first new descriptor.

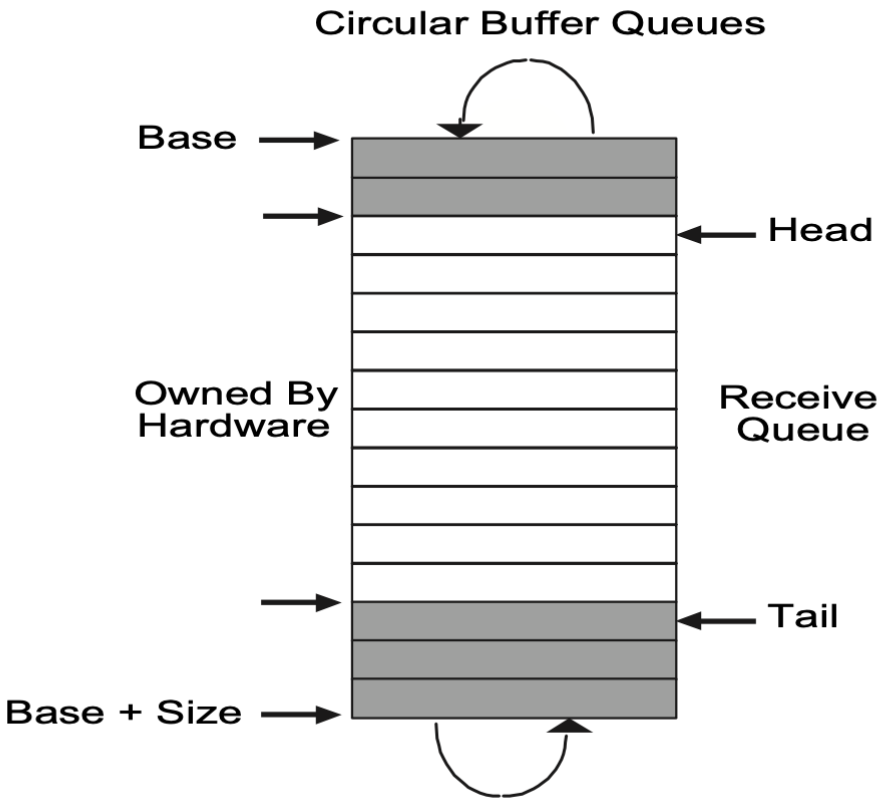
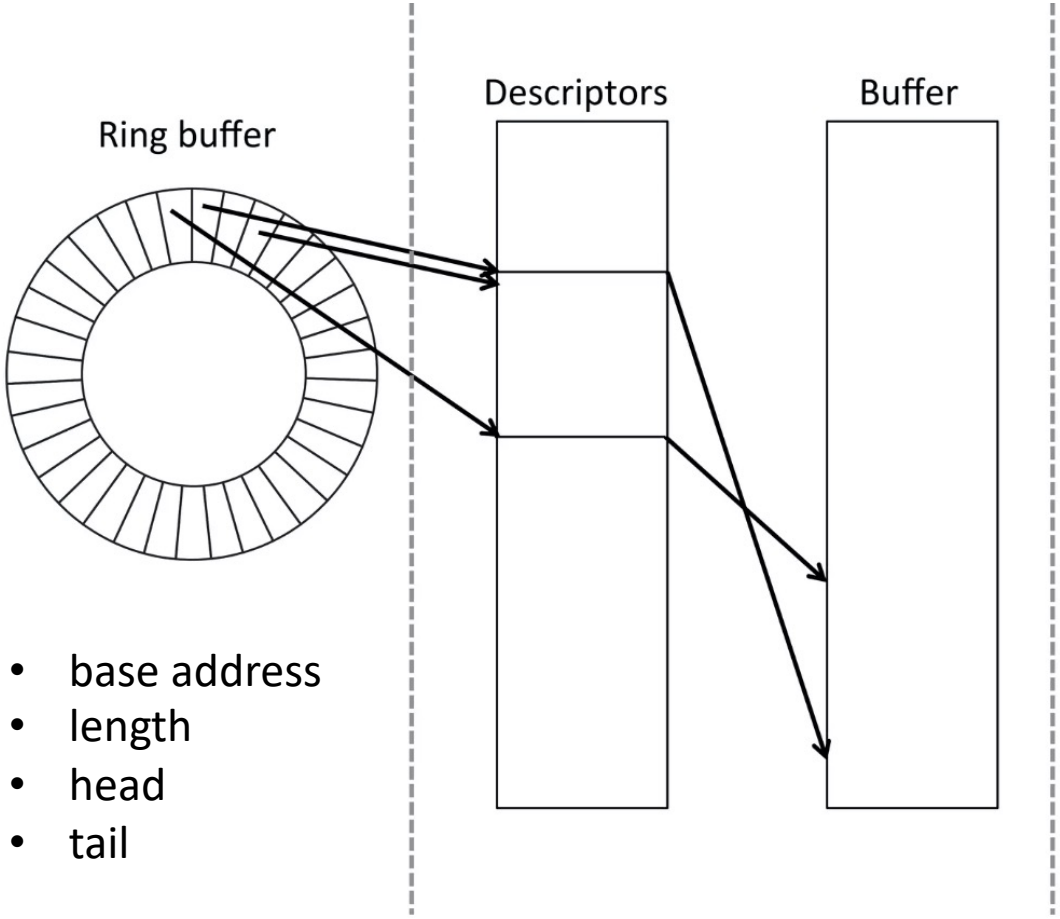
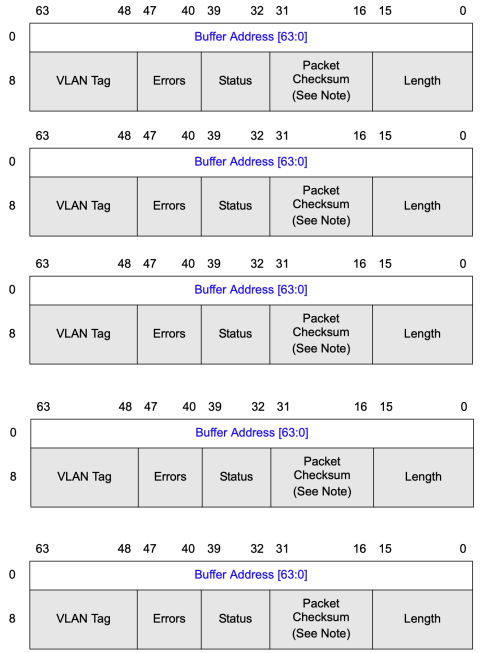


Table 3-1. Receive Descriptor (RDESC) Layout

	63 48	47 40	39 32	31 16	15 0
0	Buffer Address [63:0]				
8	VLAN Tag	Errors	Status	Packet Checksum (See Note)	Length

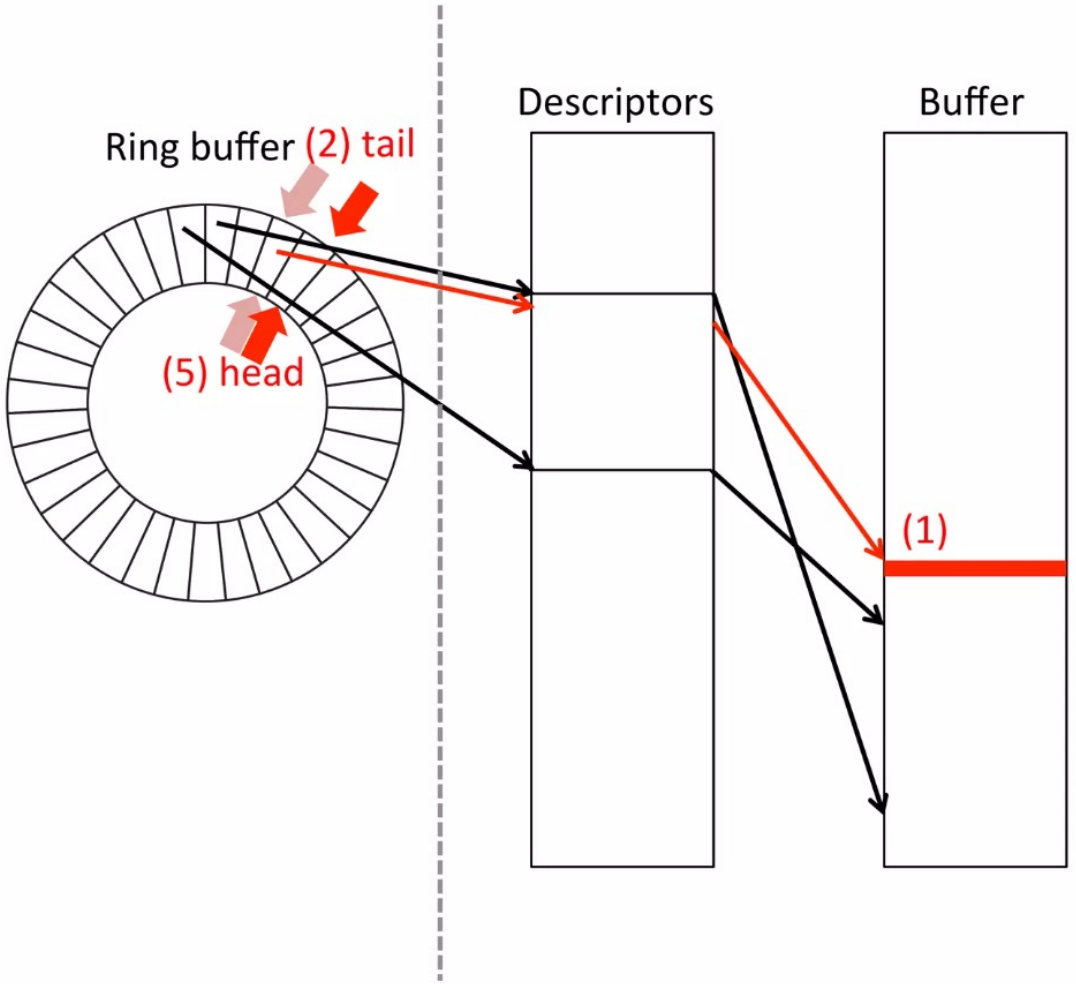


- base address
- length
- head
- tail



Descriptors

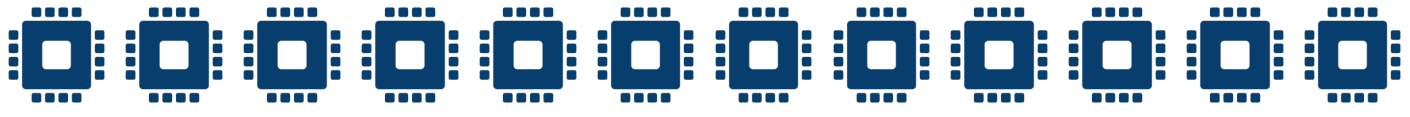
https://www.slideshare.net/hirochikasai/kernel-vm20140525



Packet transmission

1. Software writes a packet to a buffer in RAM
2. Software proceeds the tail pointer to commit the packet
3. NIC transfer the packet data from the buffer in RAM via DMA
4. NIC transmit the packet
5. NIC proceeds the head pointer to notify the packet is transmitted

https://www.slideshare.net/hirochikasai/kernel-vm20140525



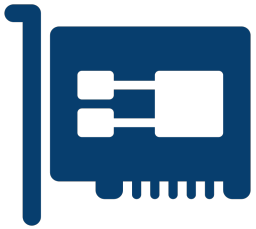
L3 cache

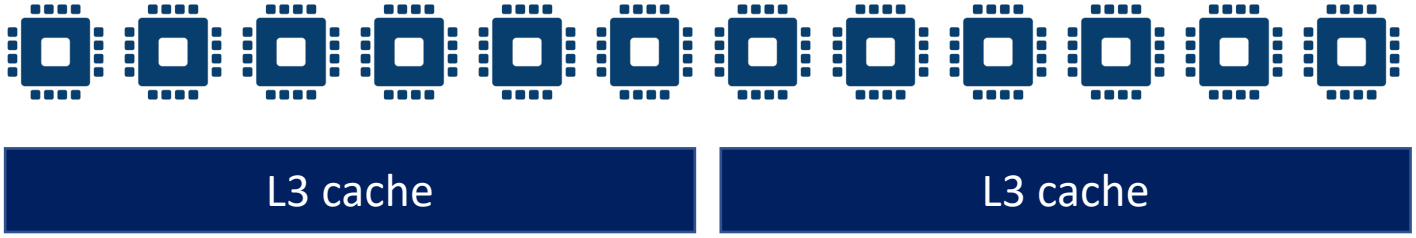
L3 cache

PCメモリ

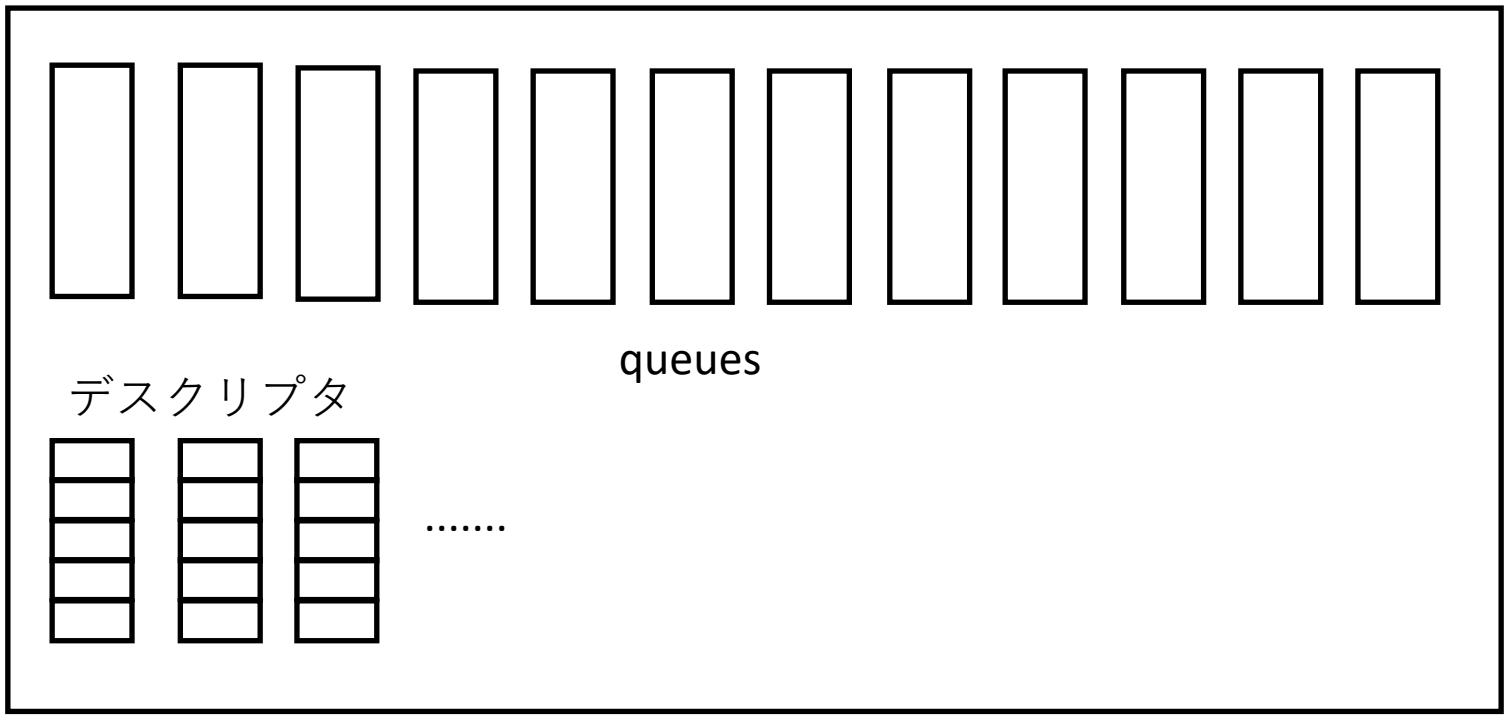
NIC上のレジスタ
- キューベースアドレス
- キュー長さ
- head
- tail

のセットがキューの数だけある
昨今のNICだと
キューの数
＝CPUの数

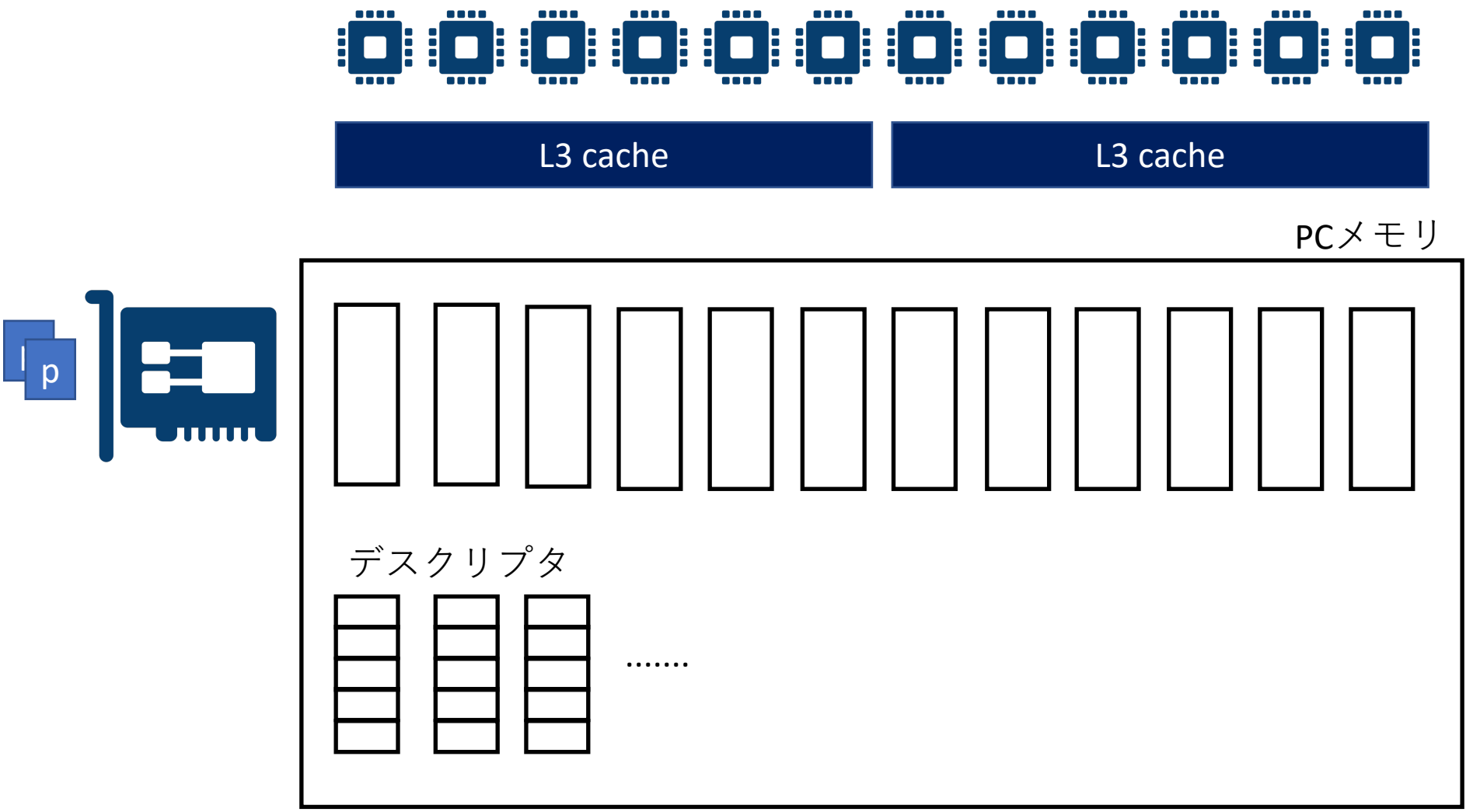


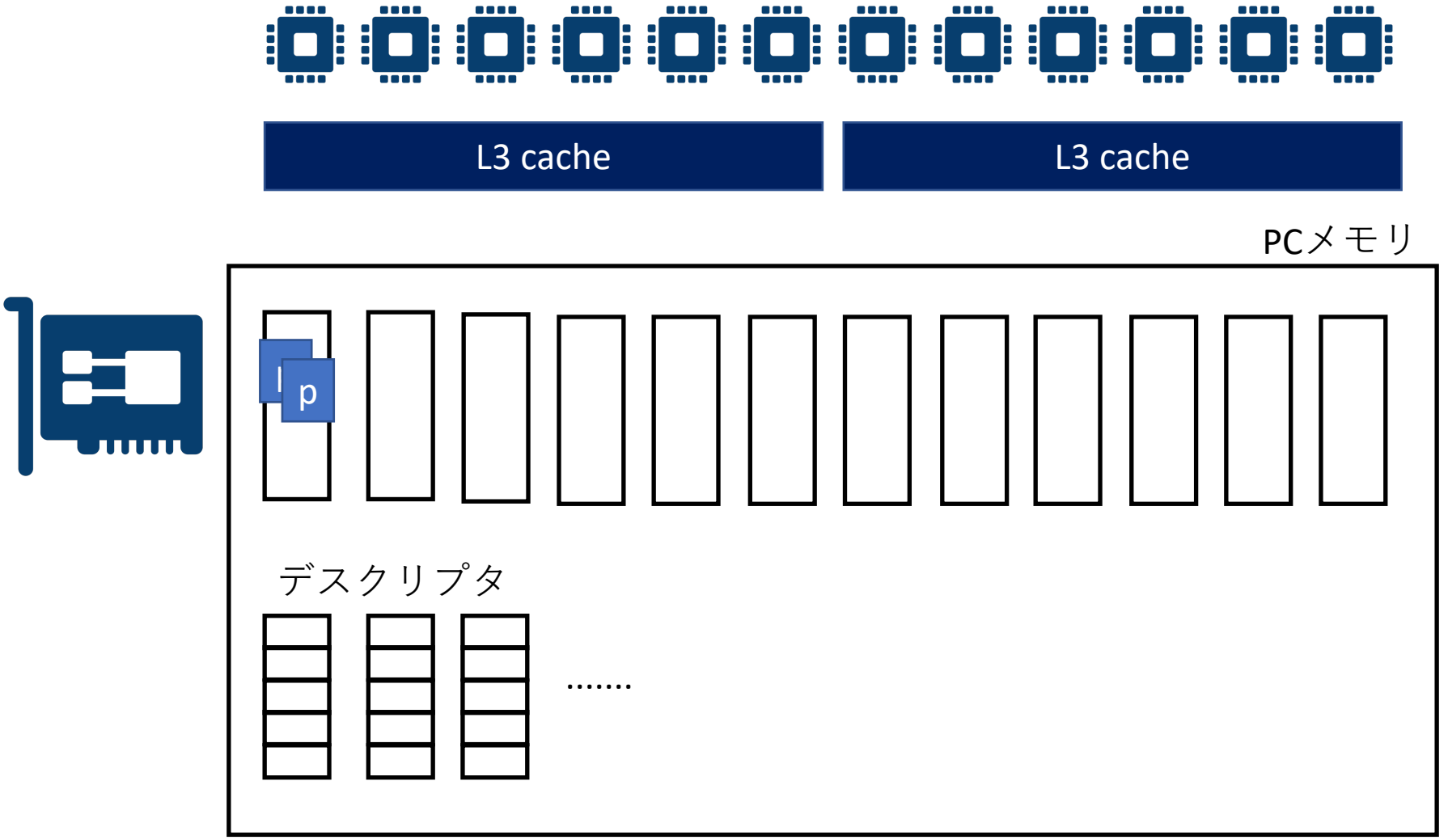


PCメモリ

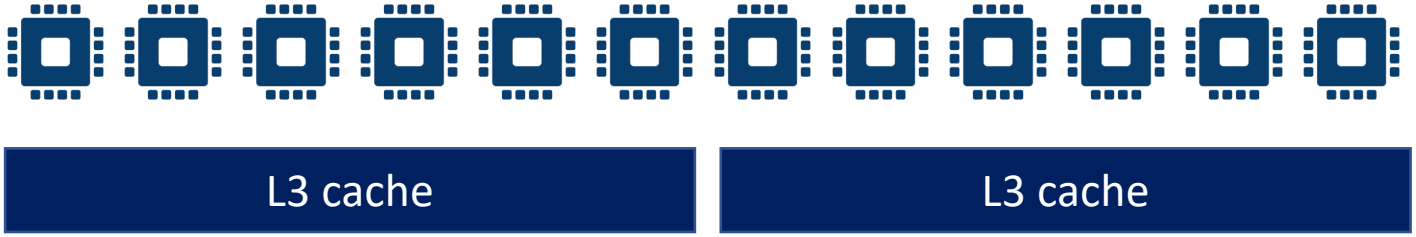


ethernetフレームが
到着したら中身をみて
IPアドレス、TCP, UDPなら
ポート番号からハッシュ関数
を使っていれるキューを決める

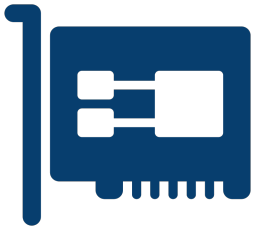


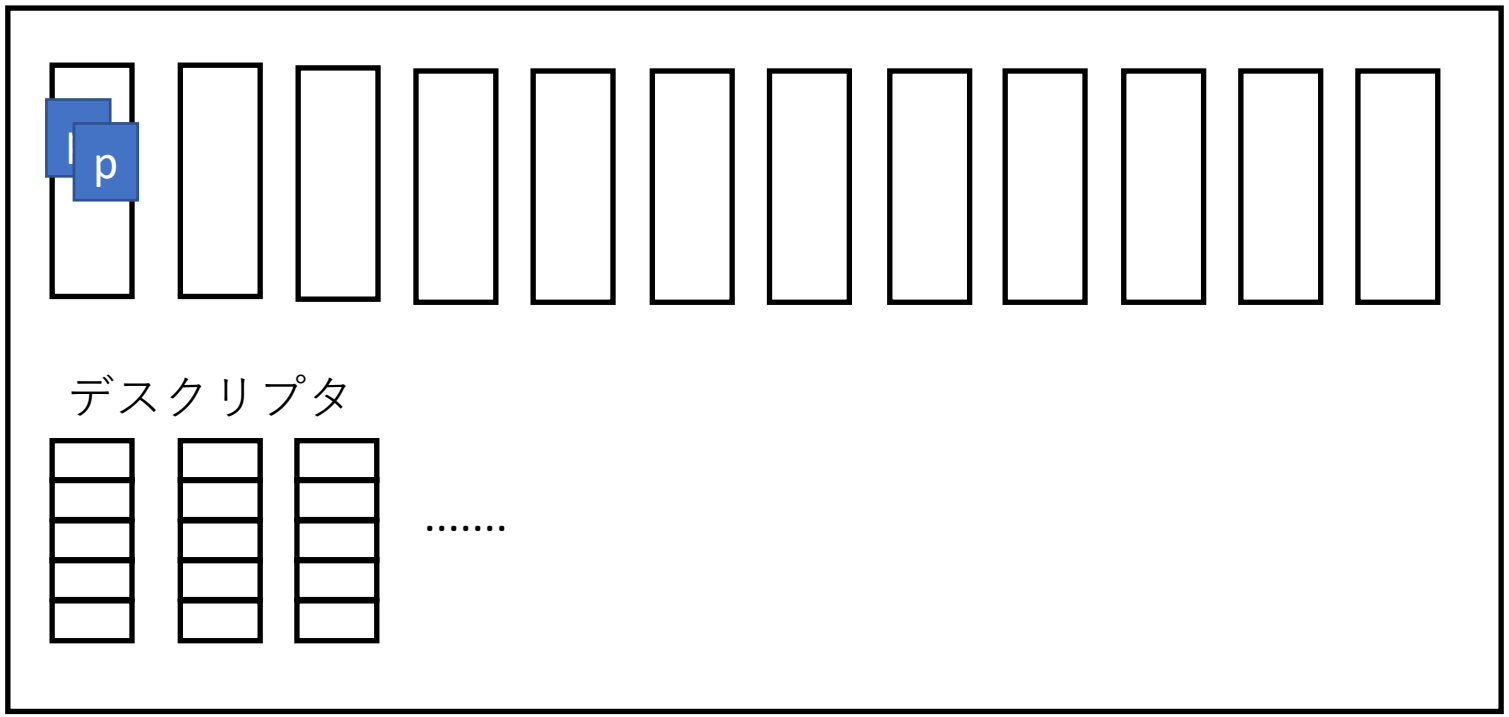

L3 cache
L3 cache
PCメモリ
p
デスクリプタ
.....

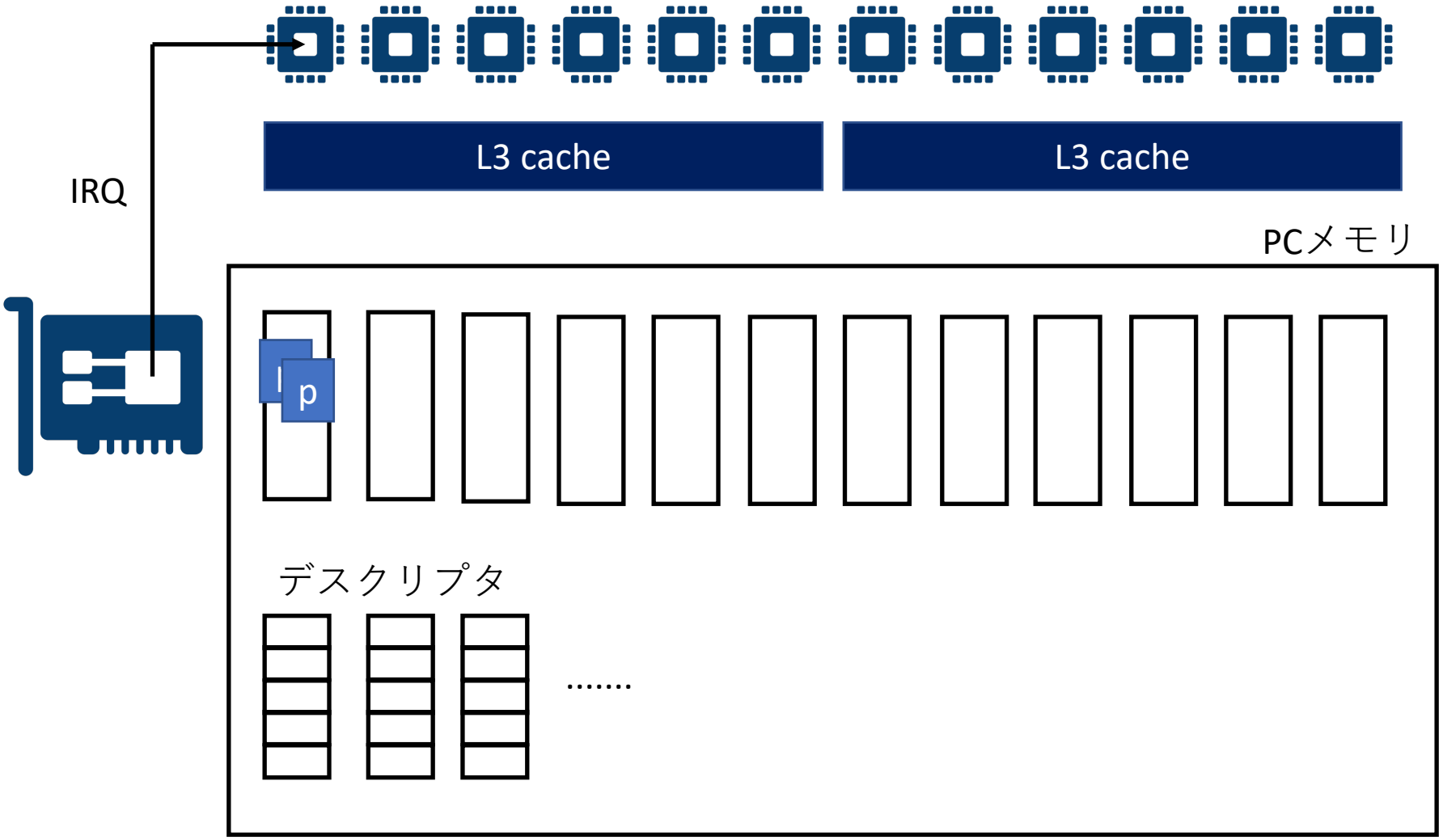
NICがデータをDMA転送
CPUはまだきづいていない
L3キャッシュにも
はいっていない

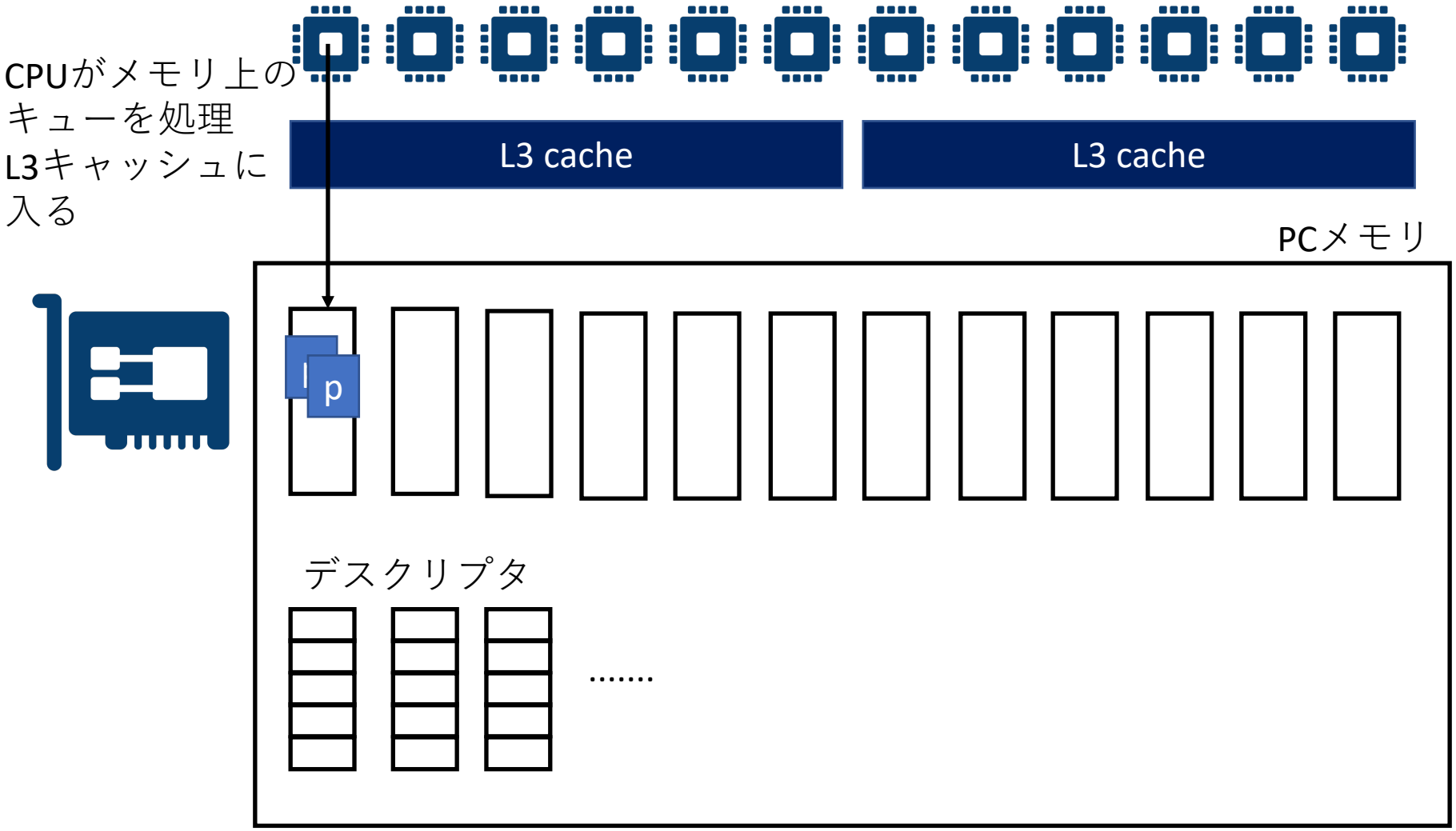


PCメモリ



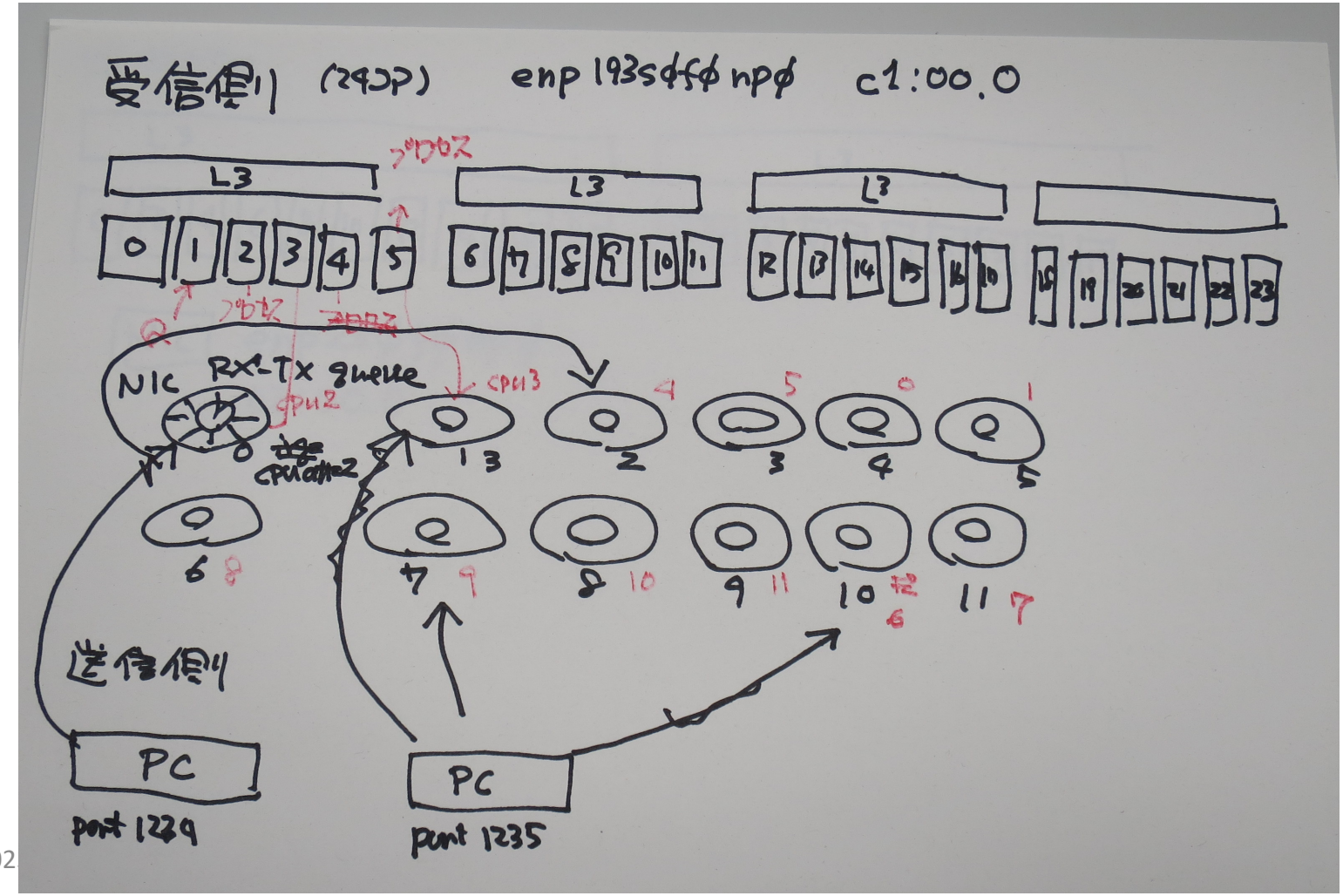



L3 cache
L3 cache
IRQ
PCメモリ
p
デスクリプタ
.....


CPUがメモリ上の
キューを処理
L3キャッシュに
入る
L3 cache
L3 cache
PCメモリ
p
デスクリプタ
.....

キューを処理するCPUの割り当て

- まずirqbalanceをとめる
- わりあてを考える



irqbalanaceが10秒に一度起動して割り込み担当CPUを変更する
（割り込み処理回数の公平化をめざしている）。
手動でわりあててもそのうち変更されてしまうのでテストの間は
irqbalanceは一時的に止める: systemctl stop irqbalance

キューに担当CPUをわりあて

echo 0 > /proc/irq/$(irq番号)smp_affinity_list
echo 1 > /proc/irq/$(irq番号)smp_affinity_list
:

queue #	CPU #
0	0
1	1
2	2
3	3
4	4
5	5
:	:
23	23

- キューを処理するCPUと読み出しプロセスのCPUはL3キャッシュを共有すると速度的に有利

- デフォルトではIPアドレス、ポートなどをみて同じTCPストリームは同じキューにはいるようになっている（のではないか？）

Intel 82599 Controller
(ちょっと前の10GbEコントローラ)
Datasheet

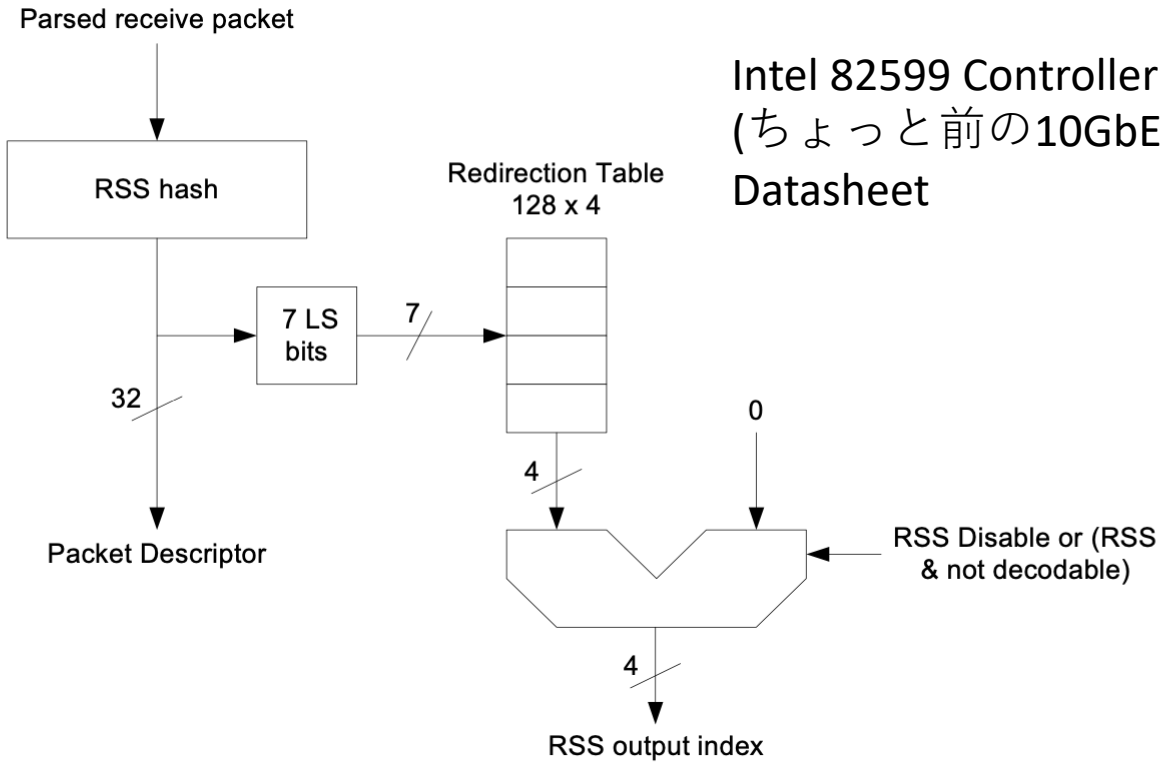


Figure 7-10 RSS Block Diagram

7.1.2.8.1 RSS Hash Function

This section provides a verification suite used to validate that the hash function is computed according to MSFT nomenclature.

The 82599's hash function follows the MSFT definition. A single hash function is defined with several variations for the following cases:

- TcpIPv4 — The 82599 parses the packet to identify an IPv4 packet containing a TCP segment per the following criteria. If the packet is not an IPv4 packet containing a TCP segment, RSS is not done for the packet.
- IPv4 — parses the packet to identify an IPv4 packet. If the packet is not an IPv4 packet, RSS is not done for the packet.

特定のパケットを特定のキューにいれる(ethtool –U)

- ethtool –U
- sudo ethtool –U $nic flow-type tcp4 src-port 1234 action 2
  ソースポート1234のtcpのパケットをキュー#2にいれる
- sudo ethtool –U $nic flow-type tcp4 src-port 1235 action 7
  ソースポート1235
- ポート以外にもIPアドレスなどが指定できる
  - なにを指定できるかはNICによる
- イーサネットなのにIPアドレスとかポートをみないとだめなのはたいへんそうだ

現在の設定をみる
ethtool –u $nic

12 RX rings available
Total 2 rules

Filter: 1022
    Rule Type: TCP over IPv4
    Src IP addr: 0.0.0.0 mask: 255.255.255.255
    Dest IP addr: 0.0.0.0 mask: 255.255.255.255
    TOS: 0x0 mask: 0xff
    Src port: 1235 mask: 0x0
    Dest port: 0 mask: 0xffff
    Action: Direct to queue 7

Filter: 1023
    Rule Type: TCP over IPv4
    Src IP addr: 0.0.0.0 mask: 255.255.255.255
    Dest IP addr: 0.0.0.0 mask: 255.255.255.255
    TOS: 0x0 mask: 0xff
    Src port: 1234 mask: 0x0
    Dest port: 0 mask: 0xffff
    Action: Direct to queue 2

消すときにはfilter番号を使う:sudo ethtool –U $nic delete 1023

ethtool –Uについて

- NICによっては（ドライバによっては?）ntuple-filtersがonになっていないと設定できないものがある。例: intel 10GbE (ixgbeドライバ)
- 今回使ったMellanoxMellanox Technologies MT2892 Family [ConnectX-6 Dx] (mlx5_coreドライバ AlmaLinux 9付属)だとntuple-filters offでも設定できた。

ixgbeの例
% sudo ethtool -K texp0 ntuple off
% sudo ethtool -U texp0 flow-type tcp4 dst-port 80 action 1
rmgr: Cannot insert RX class rule: Operation not supported
　ntupleがoffだとルールを追加できない
% sudo ethtool -K texp0 ntuple on
　onにしてルールを追加
% sudo ethtool -U texp0 flow-type tcp4 dst-port 80 action 1
Added rule with ID 2045
　追加できたかどうか確認
% sudo ethtool -u texp0
8 RX rings available
Total 1 rules

Filter: 2045
　Rule Type: TCP over IPv4
　Src IP addr: 0.0.0.0 mask: 255.255.255.255
　Dest IP addr: 0.0.0.0 mask: 255.255.255.255
　TOS: 0x0 mask: 0xff
　Src port: 0 mask: 0xffff
　Dest port: 80 mask: 0x0
　VLAN EtherType: 0x0 mask: 0xffff
　VLAN: 0x0 mask: 0xffff
　User-defined: 0x0 mask: 0xffffffffffffffff
　Action: Direct to queue 1

その他の設定

- flow controlの有効化
  ethtool –A $nic rx on tx on
  リンクパートナー（スイッチを介して接続する場合はスイッチ）もflow controlを有効にセットする必要がある。スイッチWeb UIから設定した。
- キュー長の設定
  ethtool –G $nic rx 8192 tx 8192
  ハードウェアで最大長が決まっているので
  ethtool –g $nicで調べてから最大数に設定する

ソケットレシーブバッファ、センドバッファ長の設定

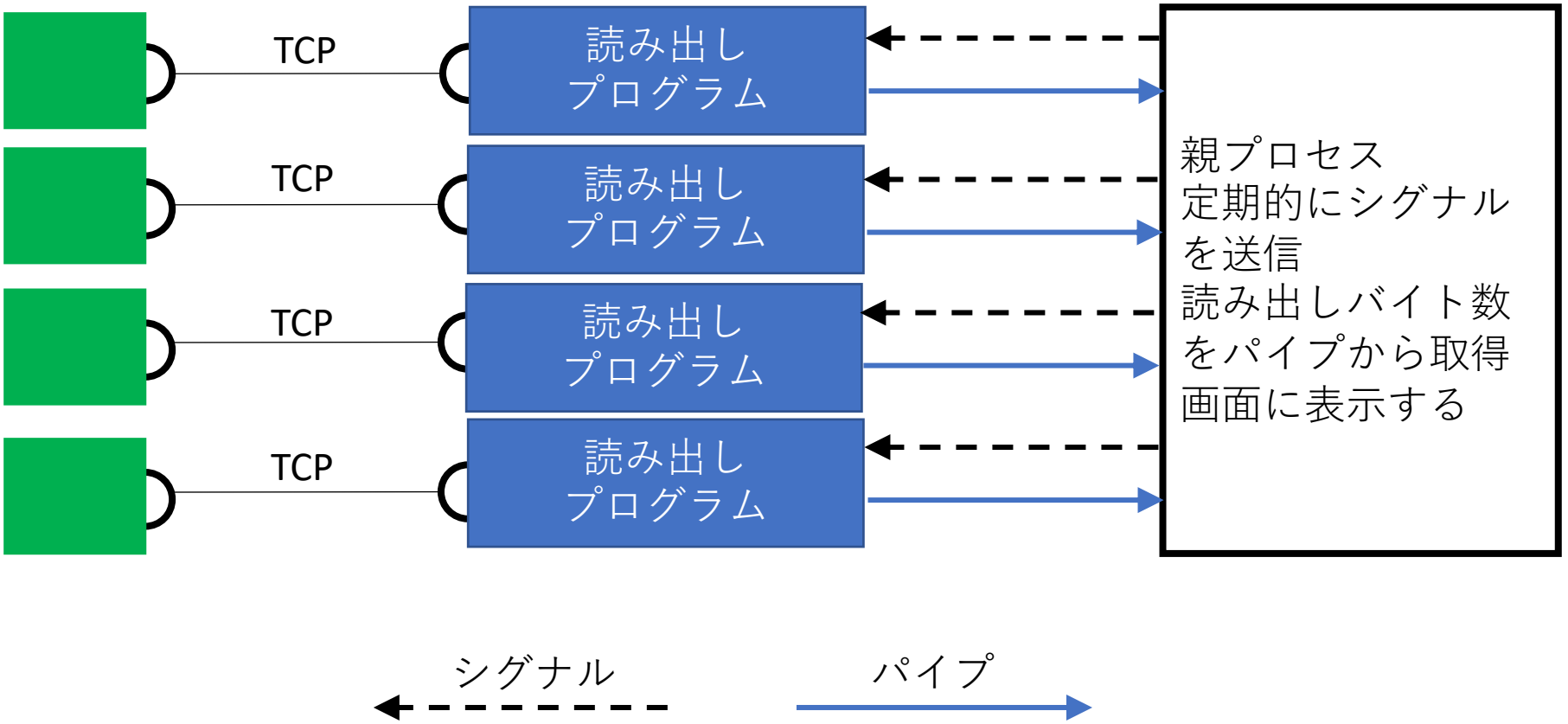
```
echo $((1024*1024*1024)) > /proc/sys/net/core/wmem_max
echo $((1024*1024*1024)) > /proc/sys/net/core/rmem_max

echo 4096   131072  $((1024*1024*1024)) > /proc/sys/net/ipv4/tcp_rmem
echo 4096   16384   $((1024*1024*1024)) > /proc/sys/net/ipv4/tcp_wmem
```

自動調節で1GBまでおおきくできるようにしてみた。
2GBにセットするとエラーがおきた。
1.5GBまで大きくできることは確認した。

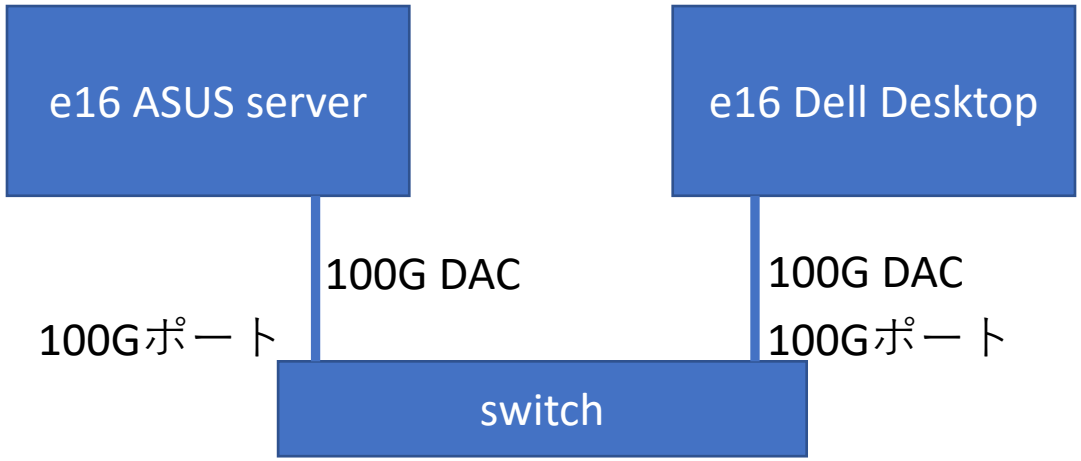
データサーバー側の設定

- HyperThread OFFで起動
  - 設定ファイルは変更していない
- センドバッファの調整（前ページ）
- irqbalanceの停止
- 停止後みるとキューを処理するCPUとして１個目のCPU (NICがぶさがっているCPU)になっていたのでとりあえず手動での設定はせず
- ethtool –G $NICで最大数にセット
- ethtool –A $NICでflow control on


TCP
TCP
TCP
TCP
読み出し
プログラム
読み出し
プログラム
読み出し
プログラム
読み出し
プログラム
親プロセス
定期的にシグナル
を送信
読み出しバイト数
をパイプから取得
画面に表示する
シグナル
パイプ

読み出しテスト

- 受信側 ASUS 2U Mellanox MT2892 Family [ConnetX-6 Dx]
- 送信側 Dell PowerEdge T340 Mellanox MT2892 Family [ConnectX-6 Dx]
- ドライバはAlmalinux 9のものを使用
-



データサーバープログラム

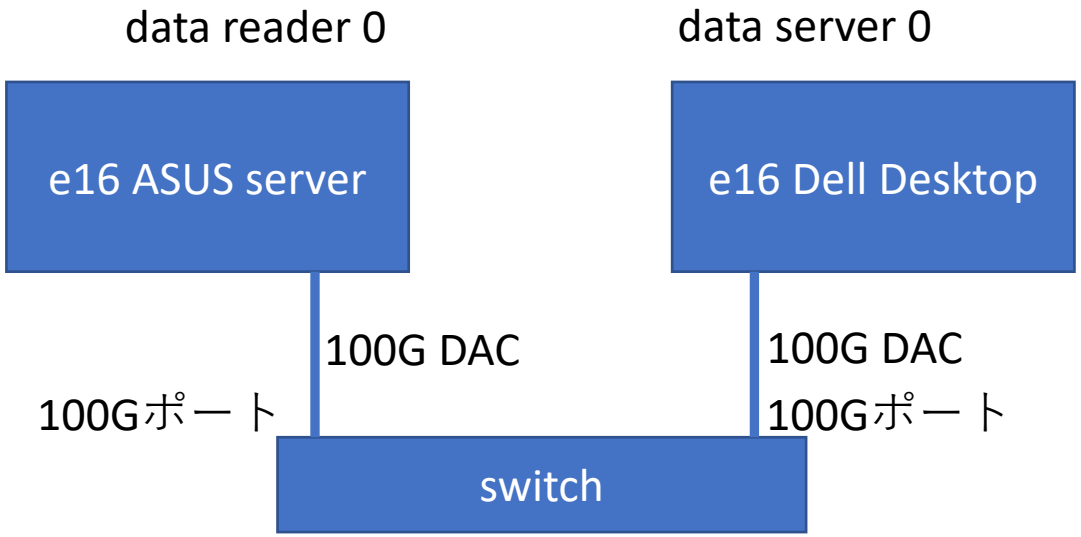
- ./server -b 128k -p 1234 -c 2 -S 512m
- ポート1234でリスン
- write(sockfd, buf, bufsize)のbufsizeは128k
- CPU#2で動かす
- ソケットセンドバッファは自動調節ではなく512MBを確保

- 2個目のサーバーはポート1235をリスン以外は同様

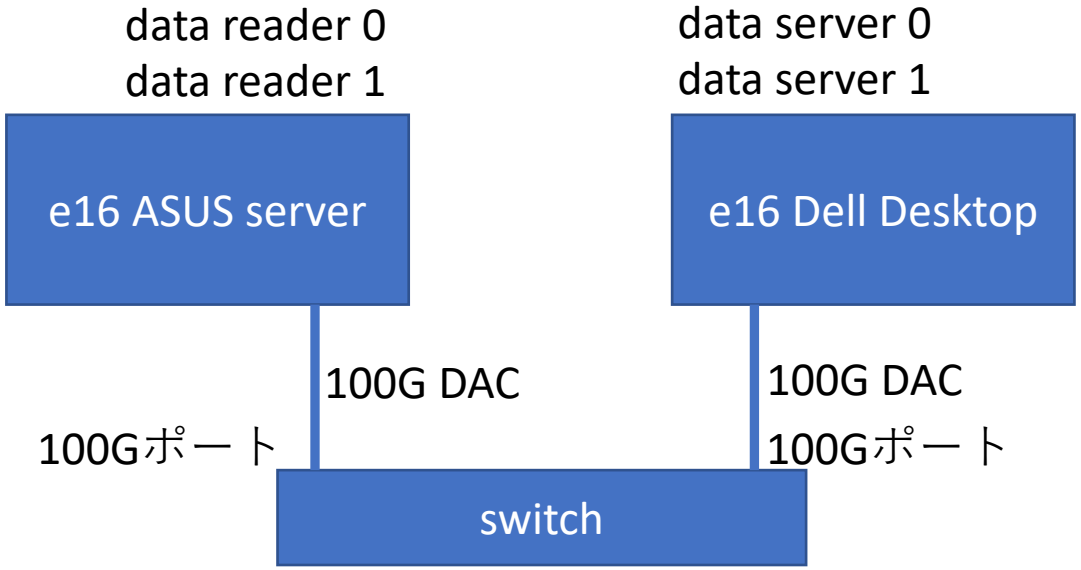
読み出しプログラム

- TCPセッション１つにつき１プロセスを起動
- 1秒に一度読み出しバイト数などを親に報告
- read(sockfd, buf, bufsize)のbufsizeは2MB。その場で読めたぶんだけ読む方式
- ソケットレシーブバッファは自動調整
- テスト1: tcp 1本でCPU #5に投入
- テスト2: tcp 1本でL3キャッシュを共有しないCPU #17に投入
- テスト3: tcp 2本でCPU #5と#11に投入

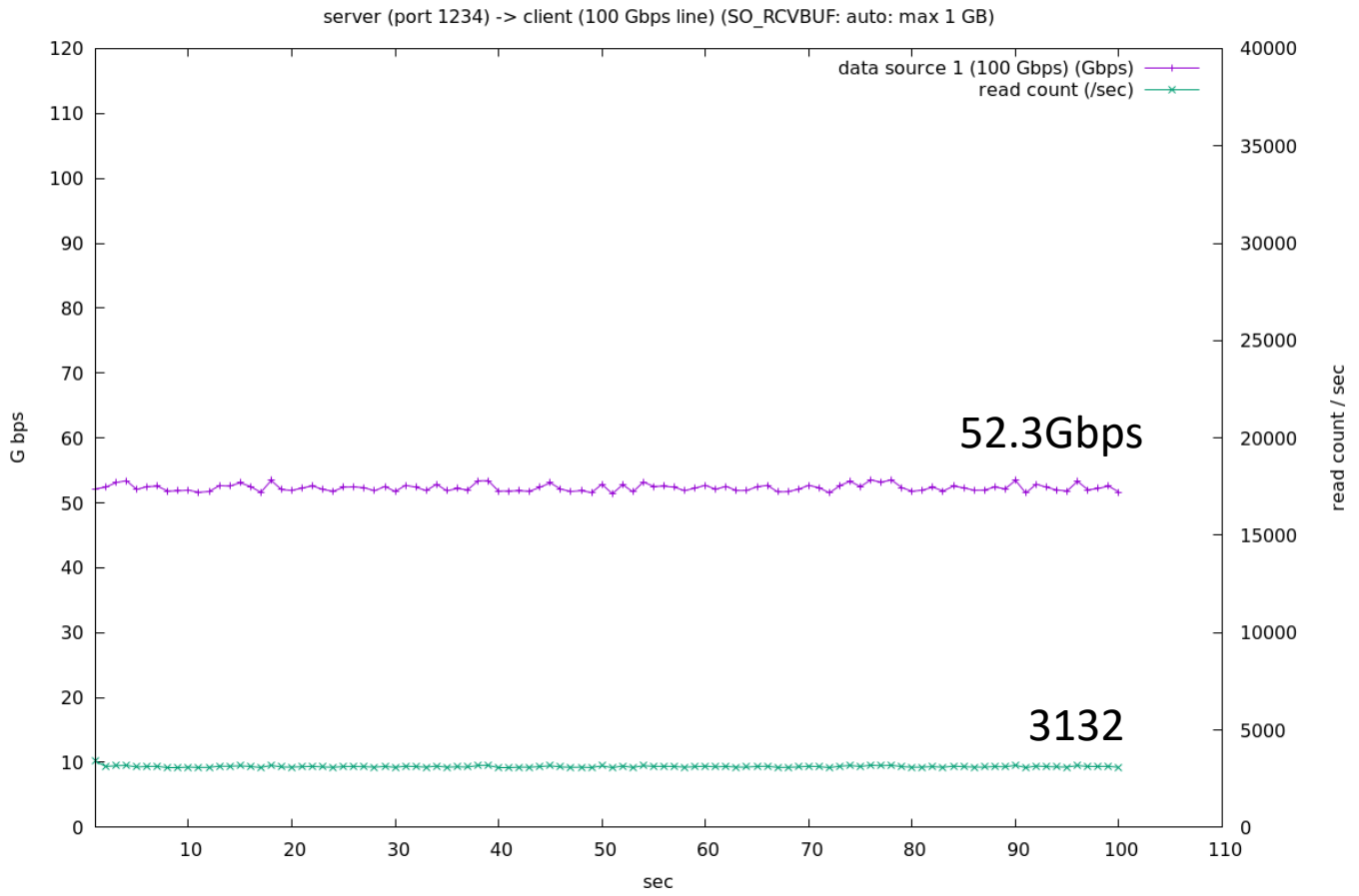
テスト1, 2


data reader 0
data server 0
e16 ASUS server
e16 Dell Desktop
100G DAC
100G DAC
100Gポート
100Gポート
switch

テスト 3

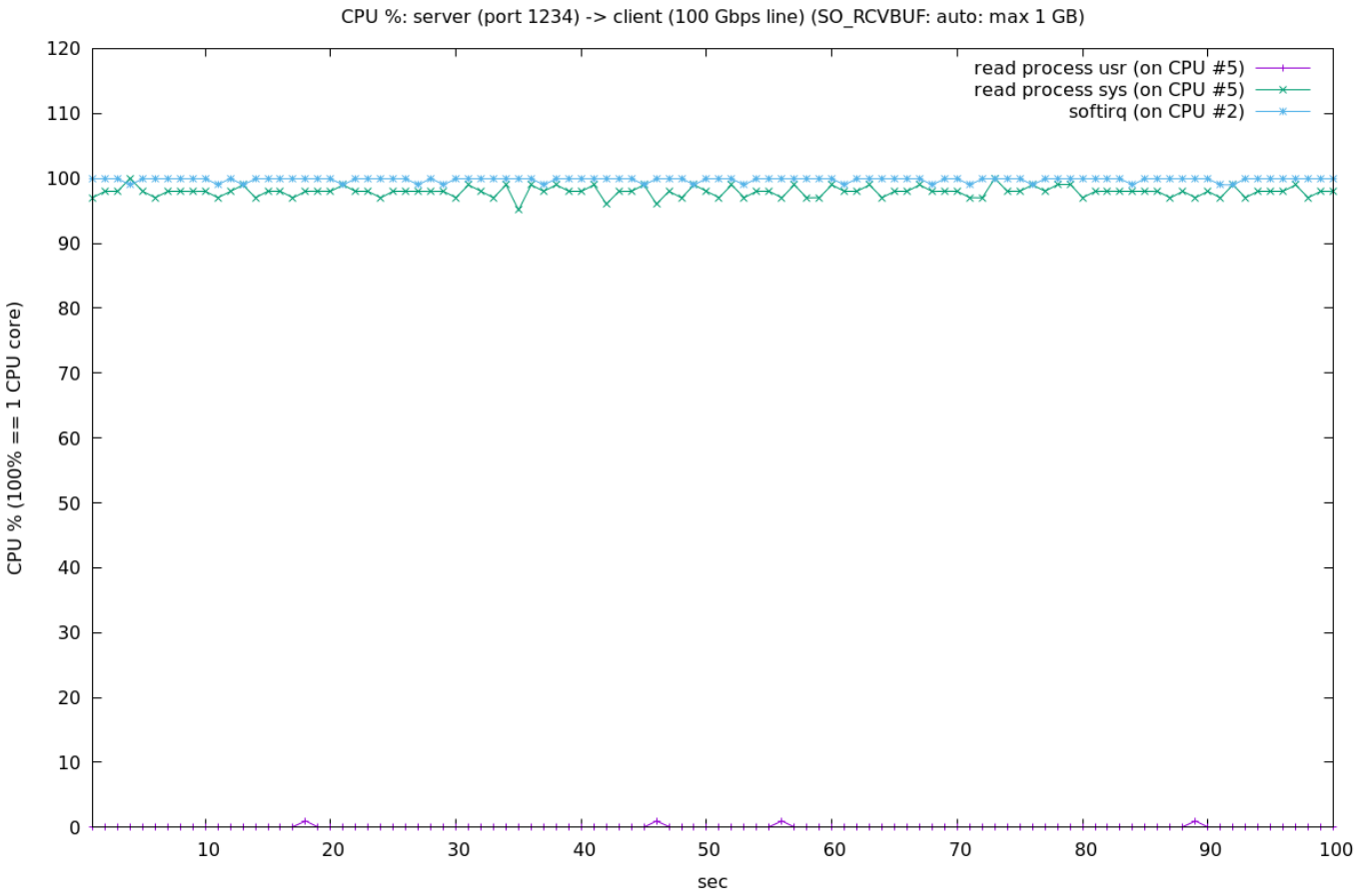

data reader 0
data reader 1
data server 0
data server 1
e16 ASUS server
e16 Dell Desktop
100G DAC
100G DAC
100Gポート
100Gポート
switch

ソケットレシーブバッファ
自動調節で読む
（TCPセッション1個）

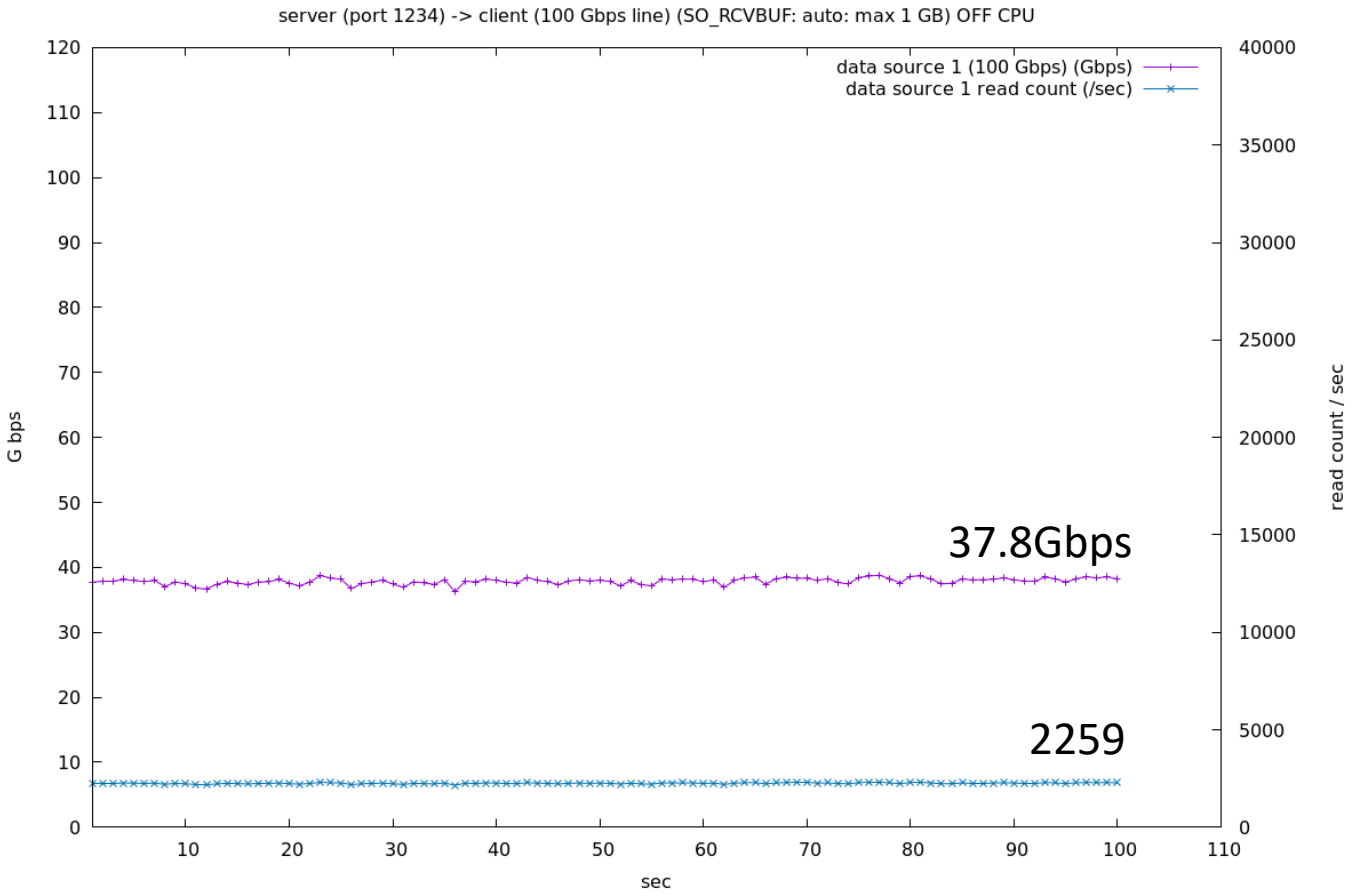


これでまあまあ安定した転送レートが
得られるようになった。

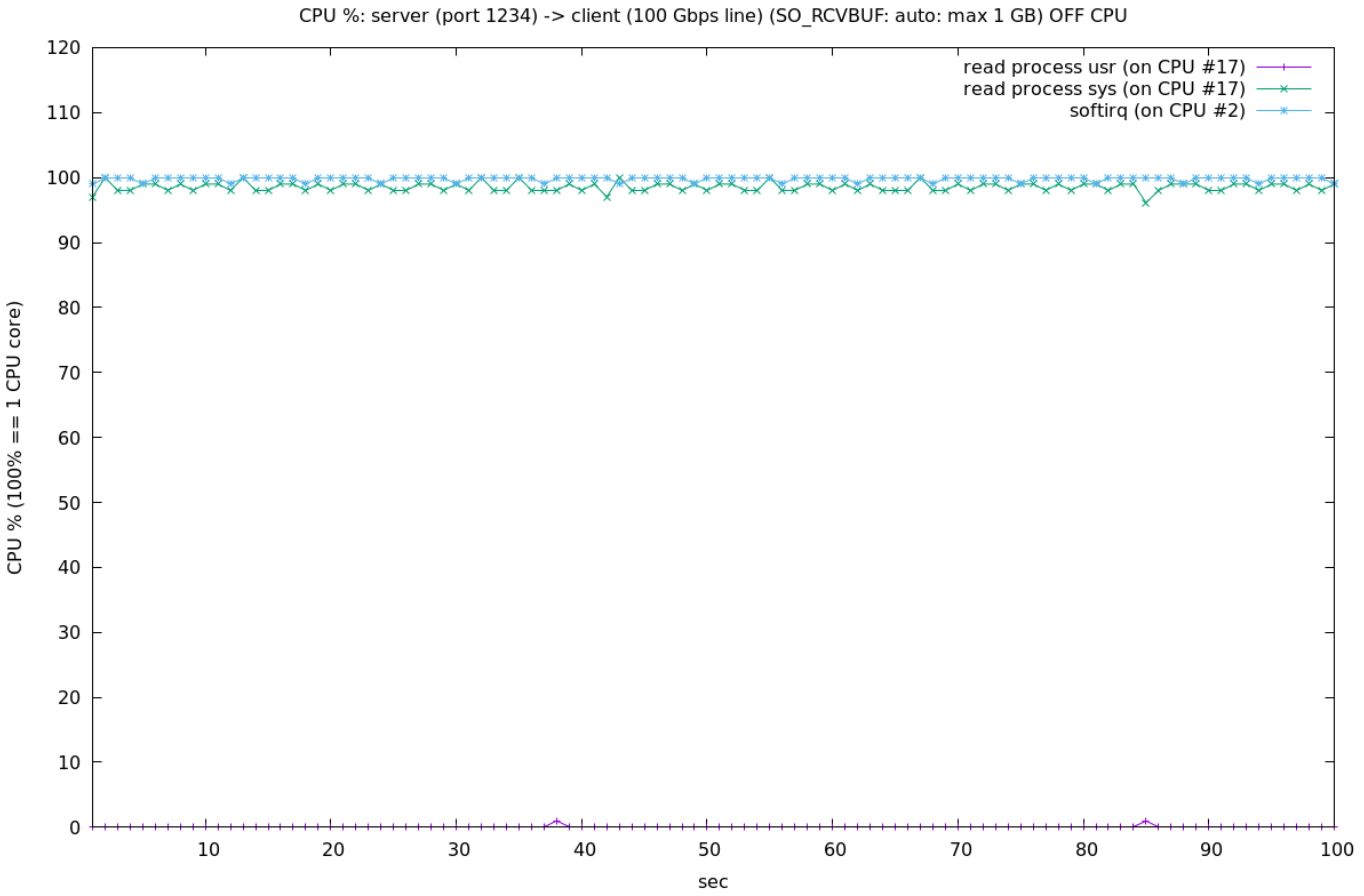
ソケットレシーブバッファ
自動調節で読む
（TCPセッション1個）CPU 消費量



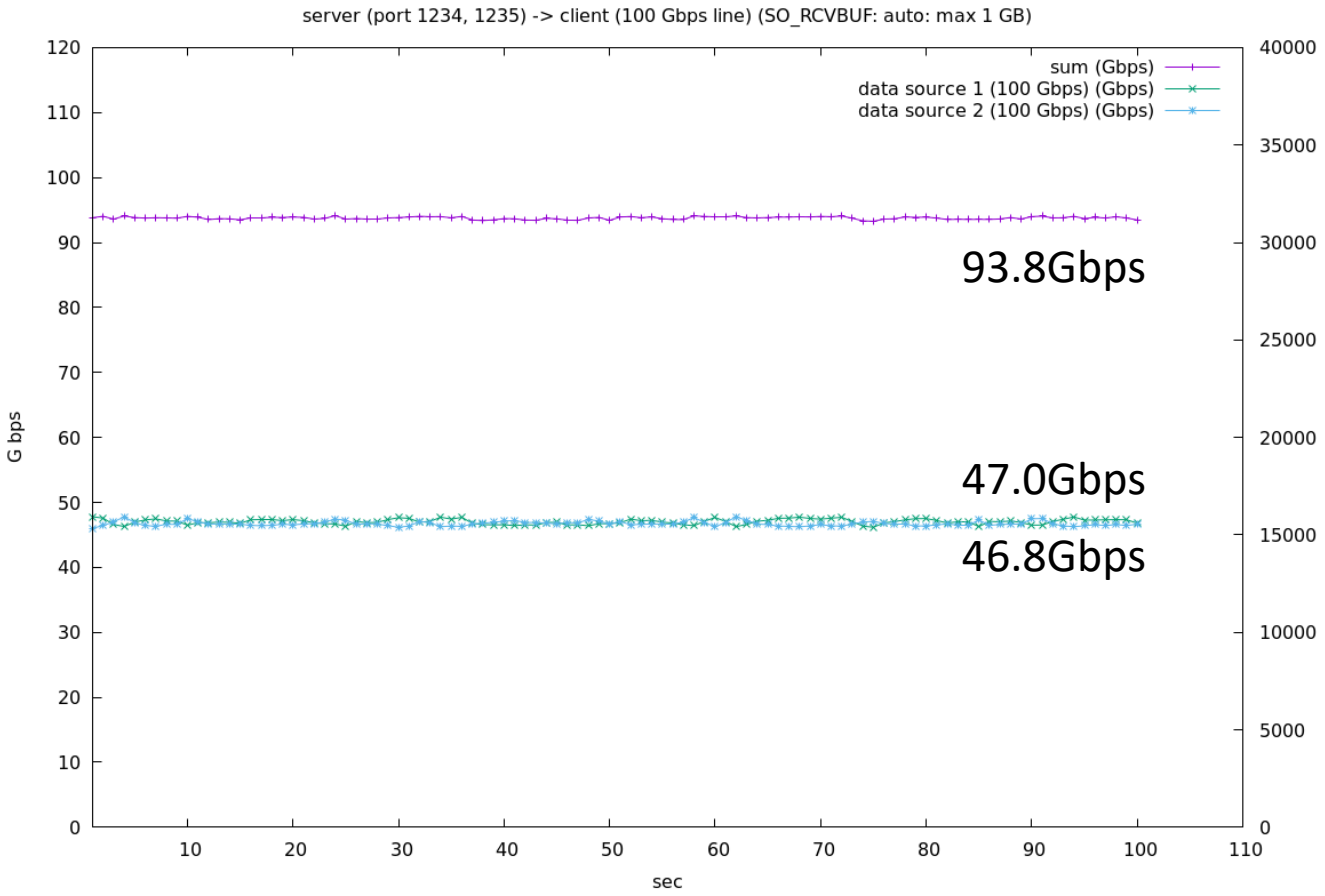
ソケットレシーブバッファ
自動調節で読む
（TCPセッション1個）
キューとL3キャッシュを共有しないCPUコアにプロセスを投入



ソケットレシーブバッファ
自動調節で読む
（TCPセッション1個）
キューとL3キャッシュを共有しないCPUコアにプロセスを投入
CPU消費量

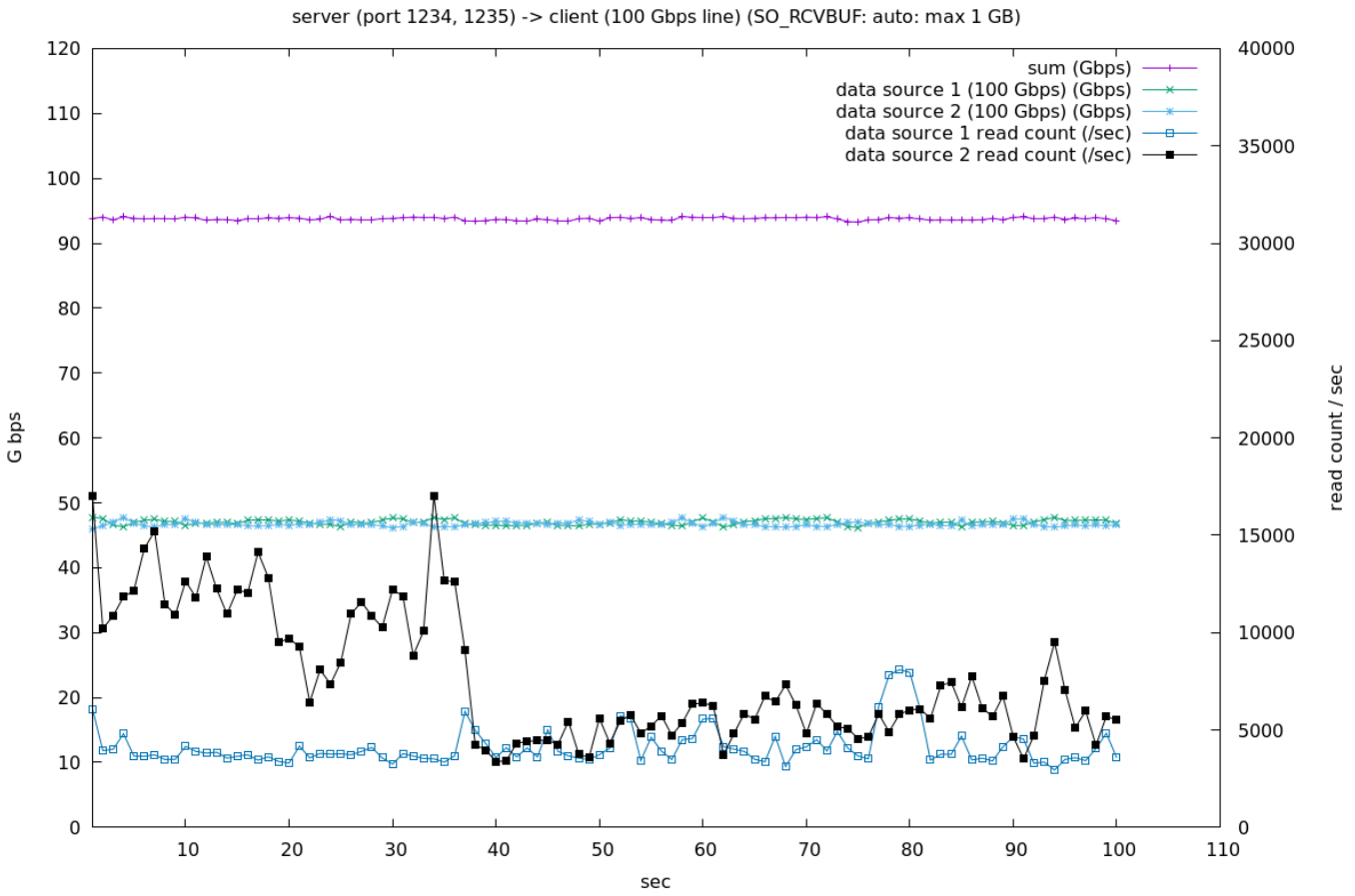


ソケットレシーブバッファ
自動調節で読む
（TCPセッション2個。データサーバーは同じものを使用）
読み側は違うL3を使うように稼働CPUコアを選択

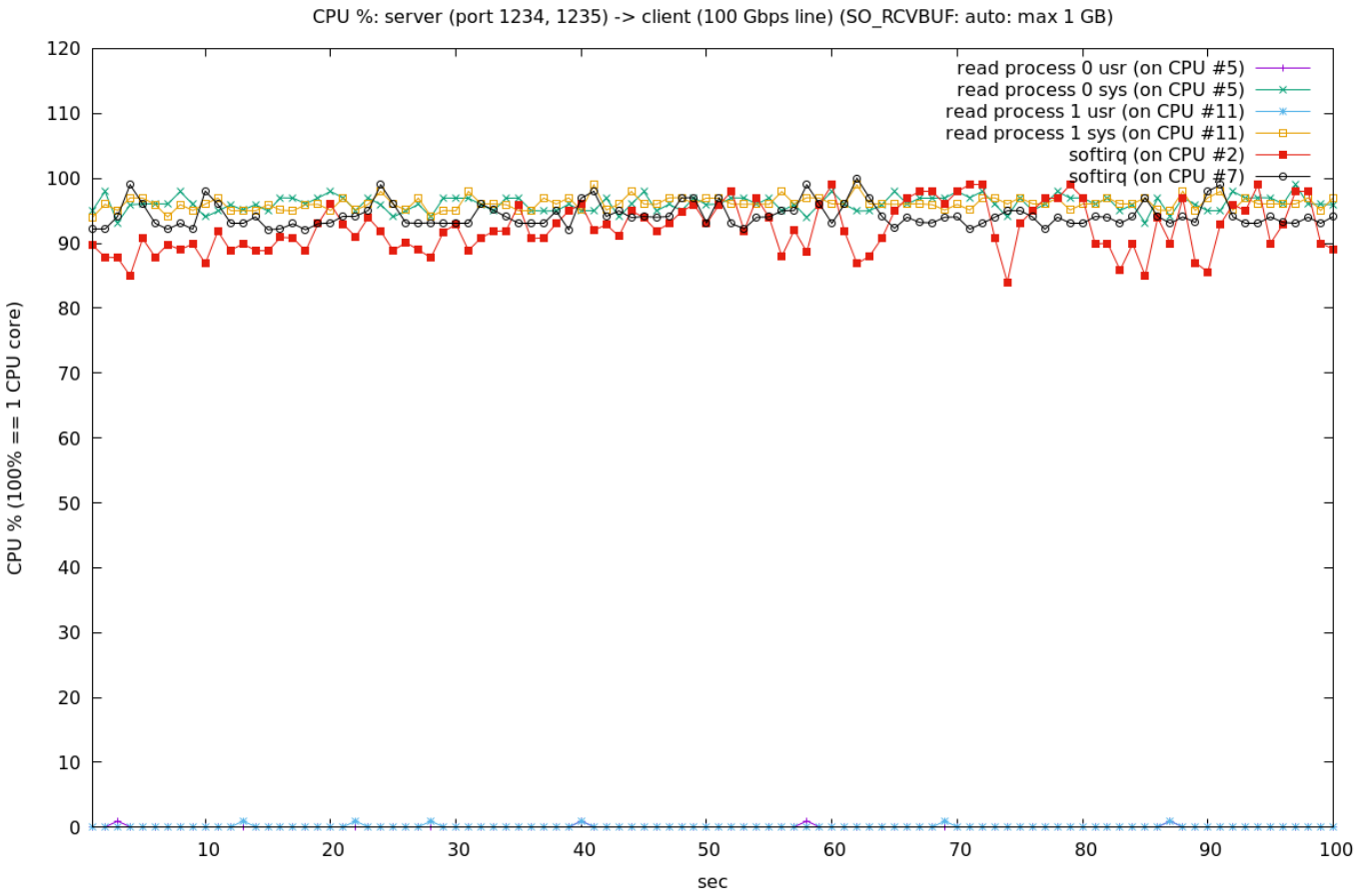


TCP Timestampオプションを有効にしていたので1448/1538 *100Gbps = 94.1 Gbpsが最大レート

ソケットレシーブバッファ
自動調節で読む
（TCPセッション2個。データサーバーは同じものを使用）
read()の回数もプロットしてみた



ソケットレシーブバッファ
自動調節で読む
（TCPセッション2個。データサーバーは同じものを使用）
CPU消費量



その他
結論

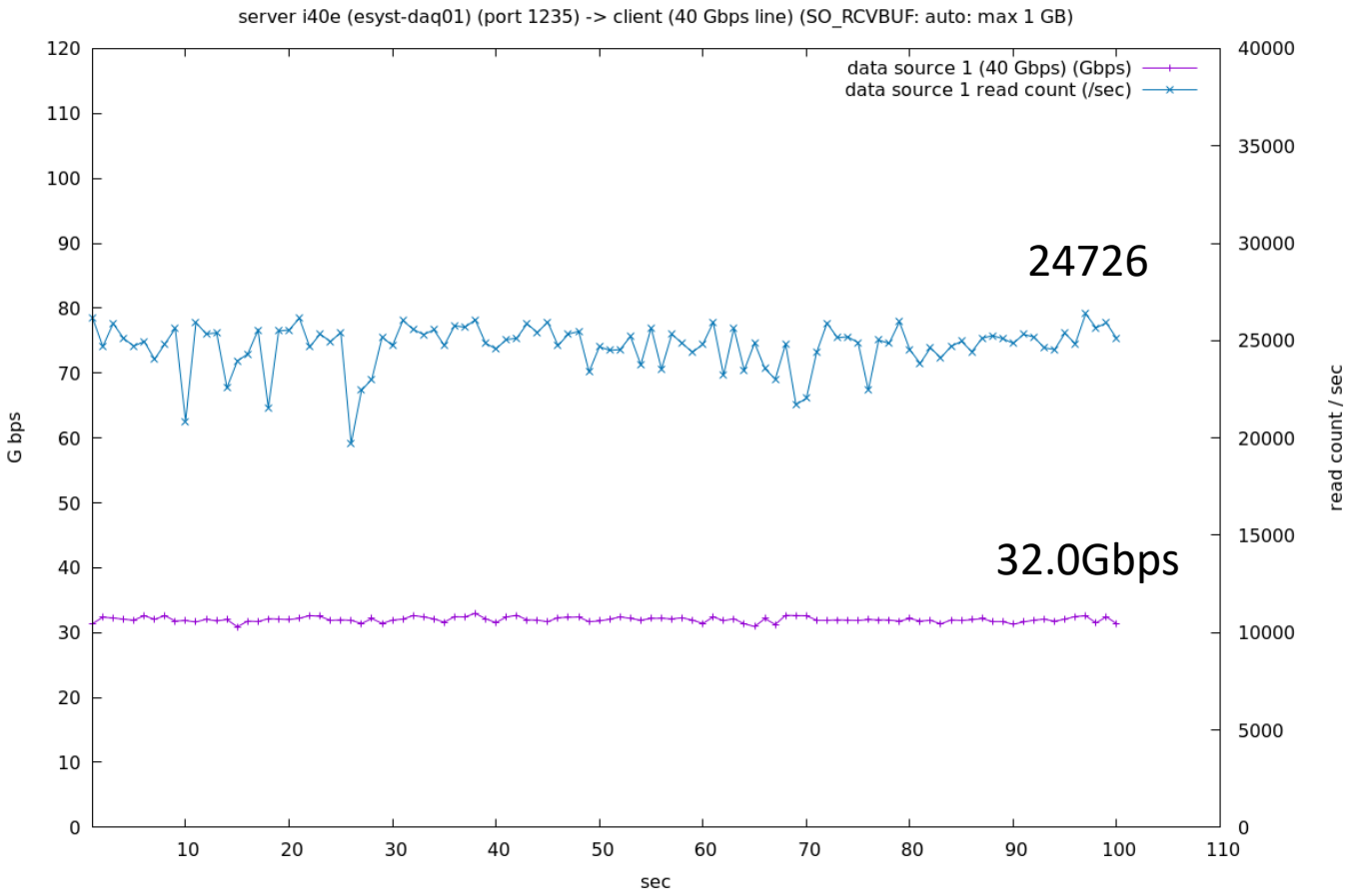
- TCPセッション１つで100秒程度読んで終了時のソケットレシーブバッファの大きさは40-50MB程度
  - 読み出し開始後1秒後には40 – 50MBになっている

- 結論
  - TCPセッション2個 + いろいろ設定でほぼ94.1Gbps (TCP Timestamps有効時の最大レート)がえられた
- スイッチは100Gbps出力可能
  - 多数束ねてだいじょうぶかどうかはテストが必要

esyst-daq01 Intel 40GbE NIC (i40e)から読み出し

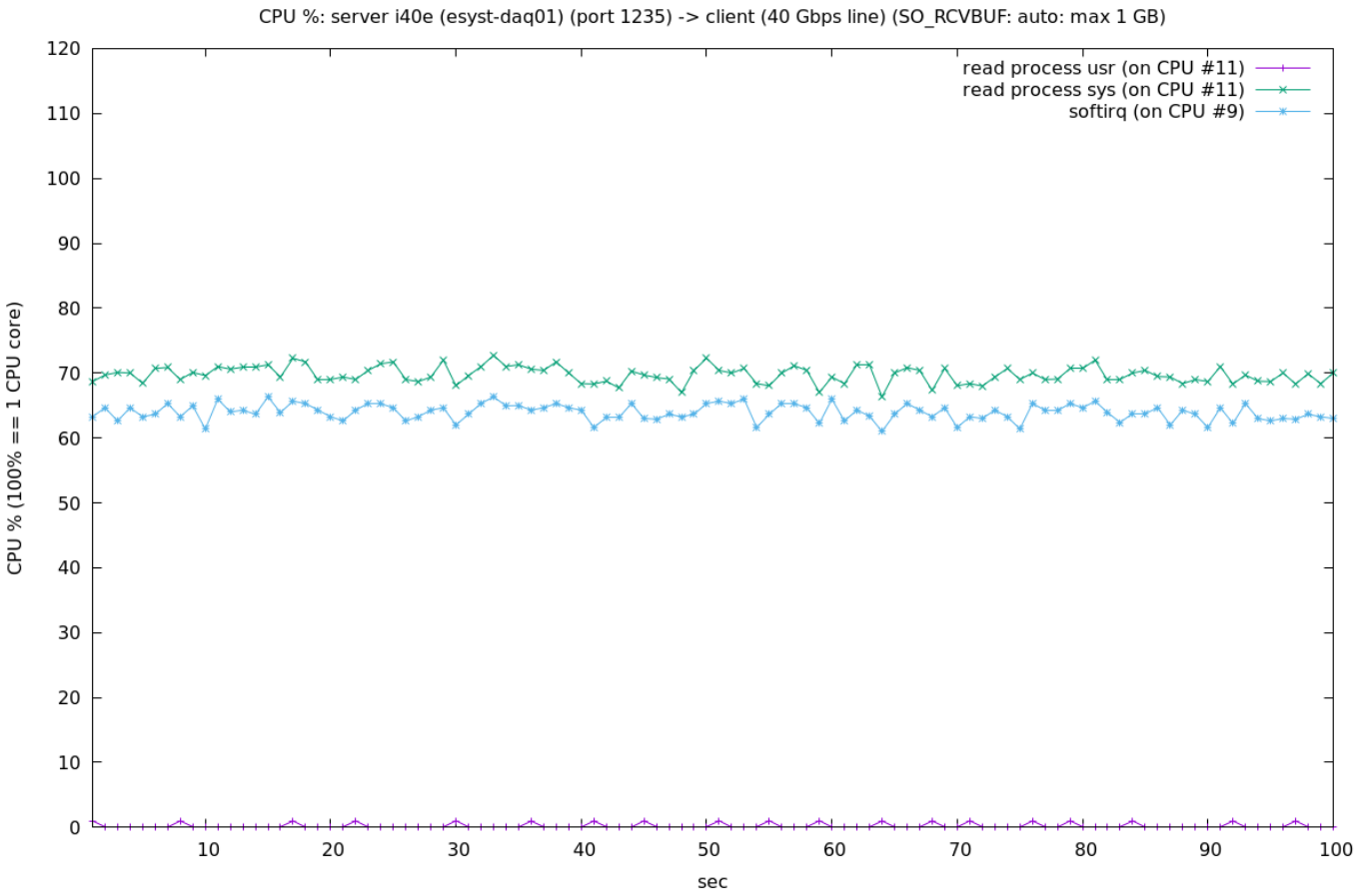
- Intel 40GbE
- Direct Attach Cableで40Gbps ポートと接続
- データサーバープログラムは他とおなじ
- ソケットセンドバッファの設定等も同じ

esyst-daq01 Intel 40GbE NIC (i40e)から読み出し



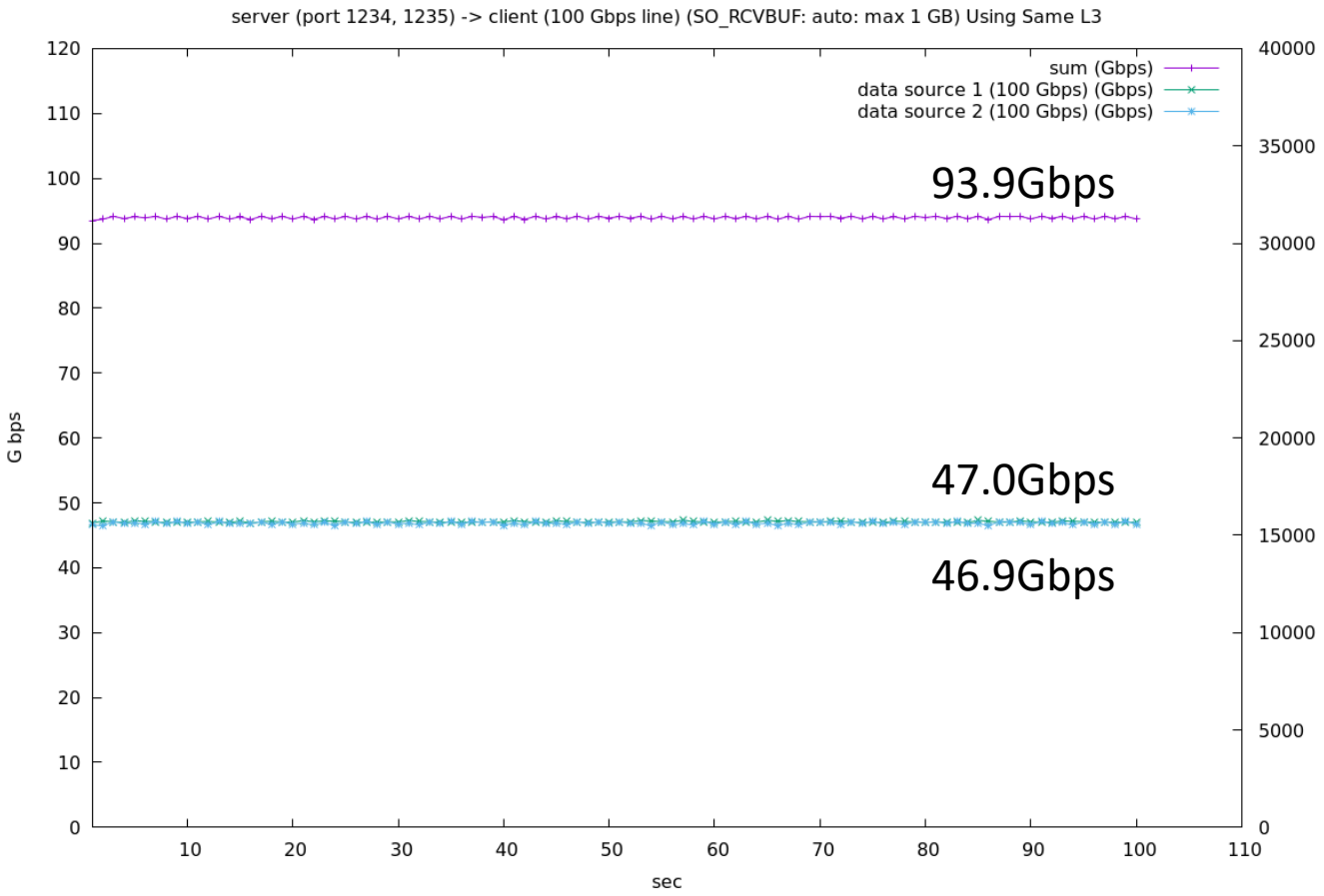
40GbE付近に到達せず。送り側サーバプロセスのsysがcpu 100%になっていた。

esyst-daq01 Intel 40GbE NIC (i40e)から読み出し CPU消費量

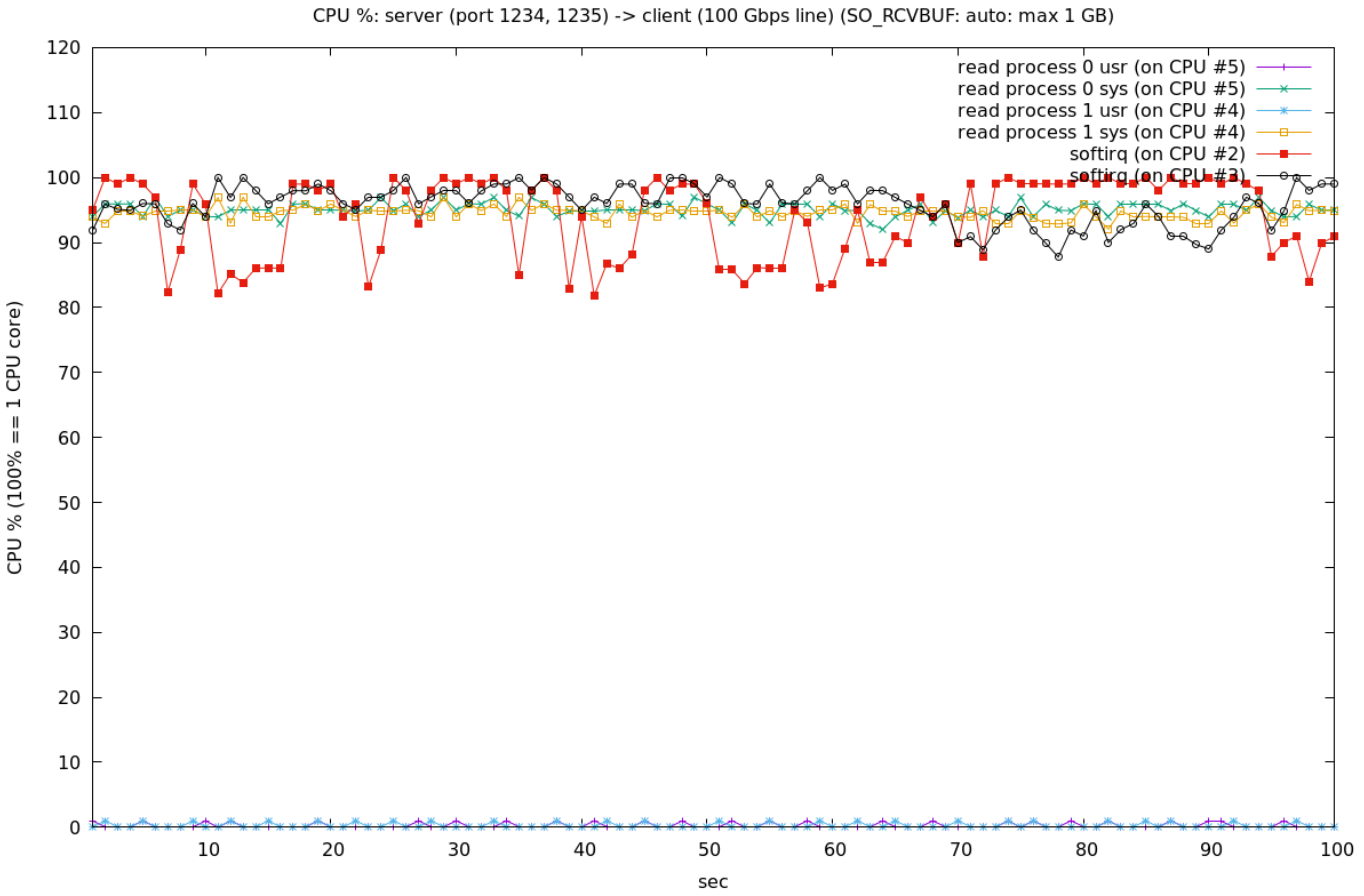


- e16からお借りしたMellanox 40GbEを手持ちPCにさしたが２台でためすも両方とも認識せず(lspciでもでてこない)

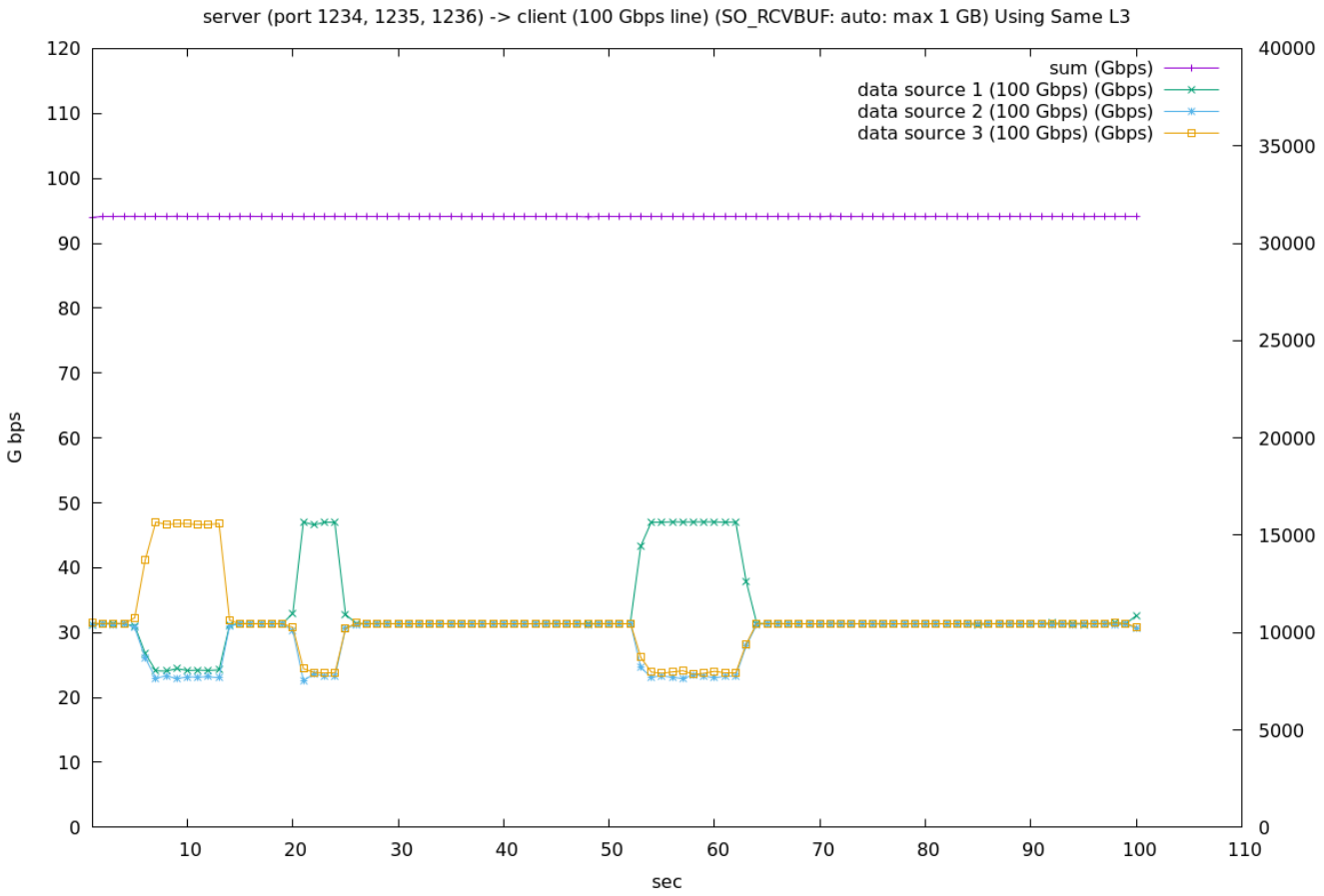
同一L3にTCP 2本（プロセス+キュー）x 2



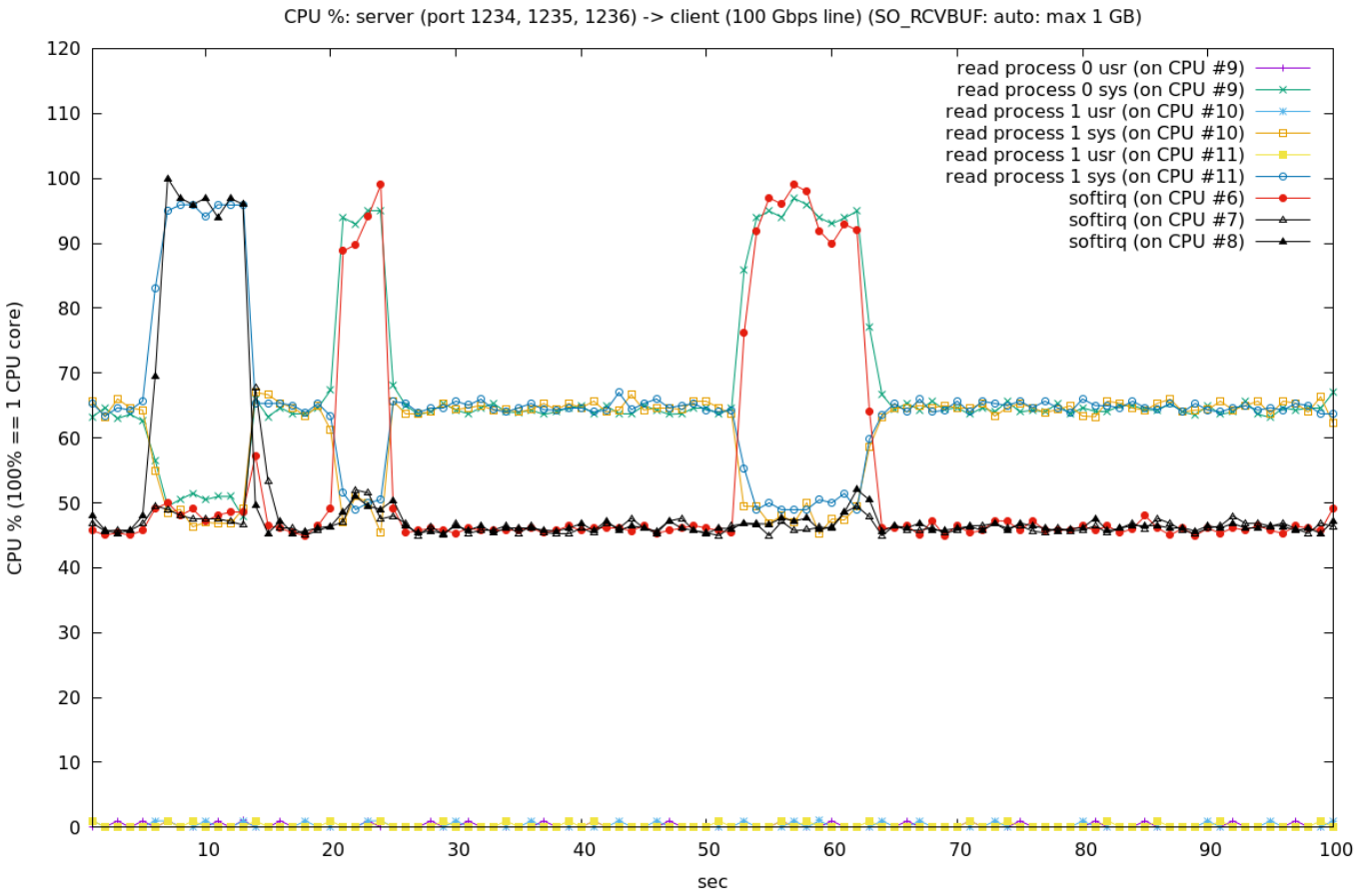
同一L3にTCP 2本（プロセス+キュー）x 2



同一L3にTCP 3本（プロセス+キュー）x 3



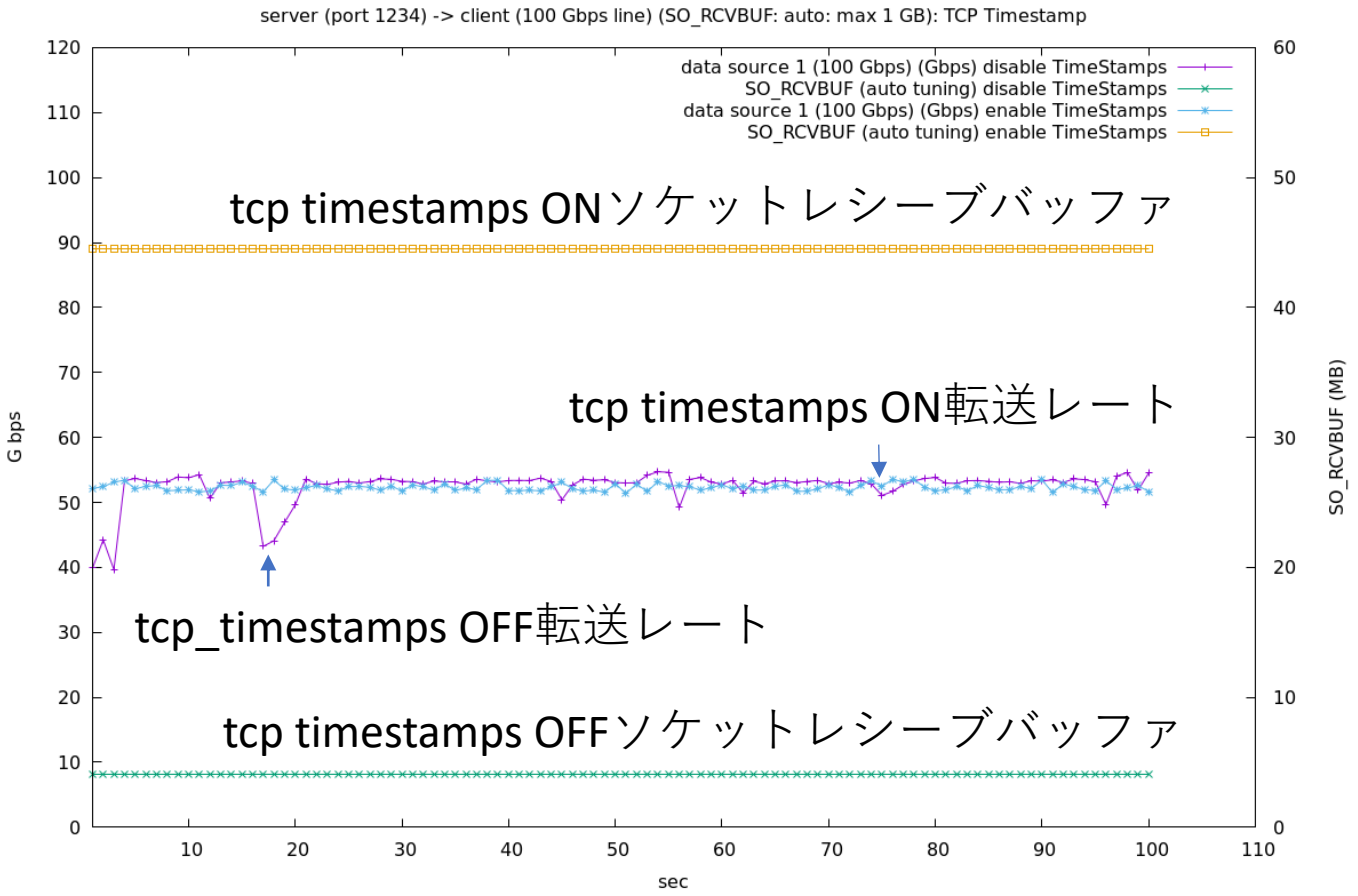
同一L3にTCP 2本（プロセス+キュー）x 3



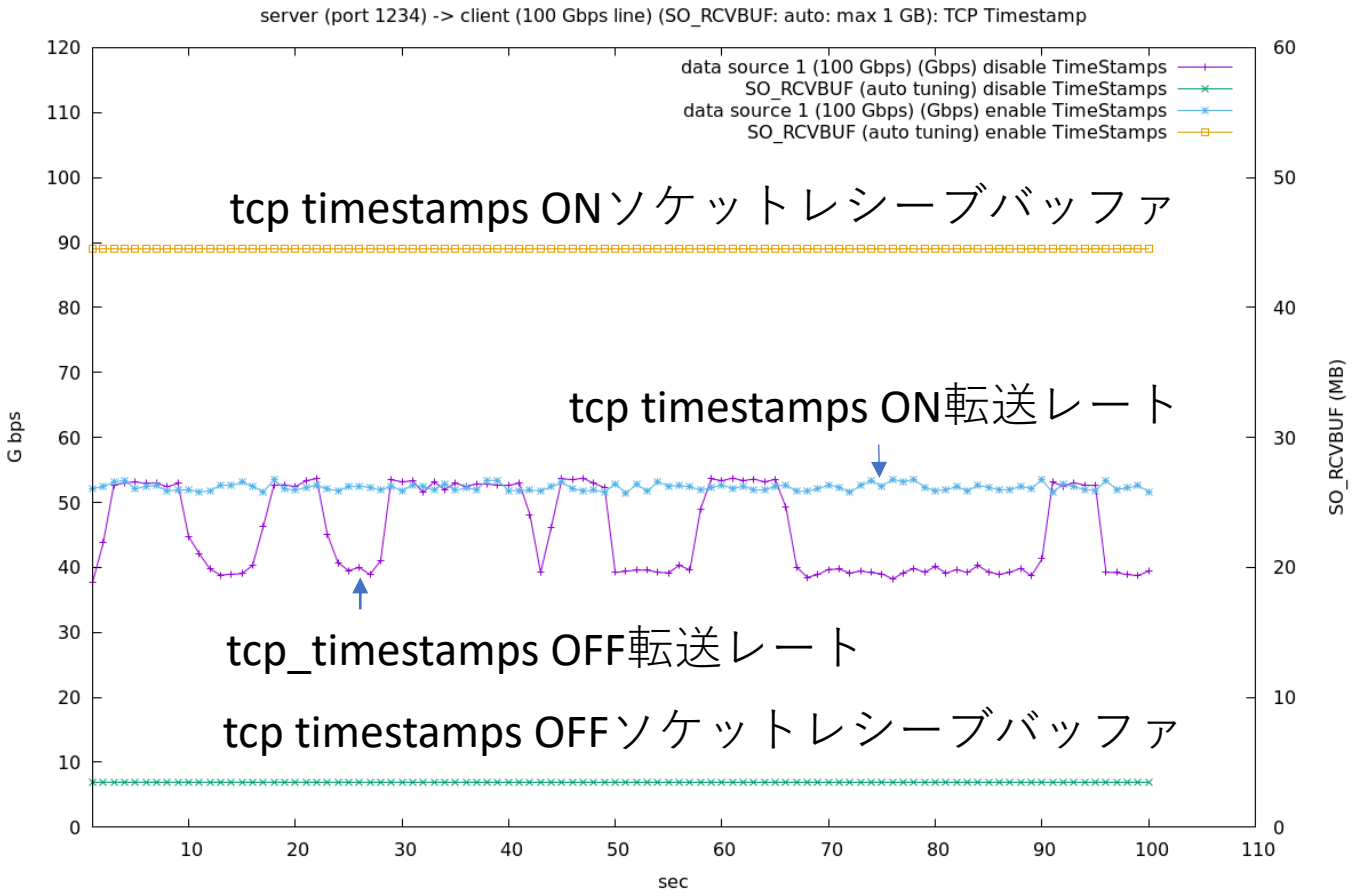
SiTCP機器と TCP Timestampsオプション

- PC間通信では通常TCP Timestampsオプションが有効化されて通信がなされている。
  - Ethernet最大ペイロード1500 - (IPヘッダ) - (TCPヘッダ) – (TCP Timestampsデータ) = 1448バイト
- SiTCP機器からのデータはTCP Timestampsオプションが無効化されて通信がなされる。
  - Ethernet最大ペイロード 1460バイト
- これまでのテストはTCP Timestampsオプション付きで通信していたのでなしにしたときの影響を調べる。
- Linuxではecho 0 > /proc/sys/net/ipv4/tcp_timestamps

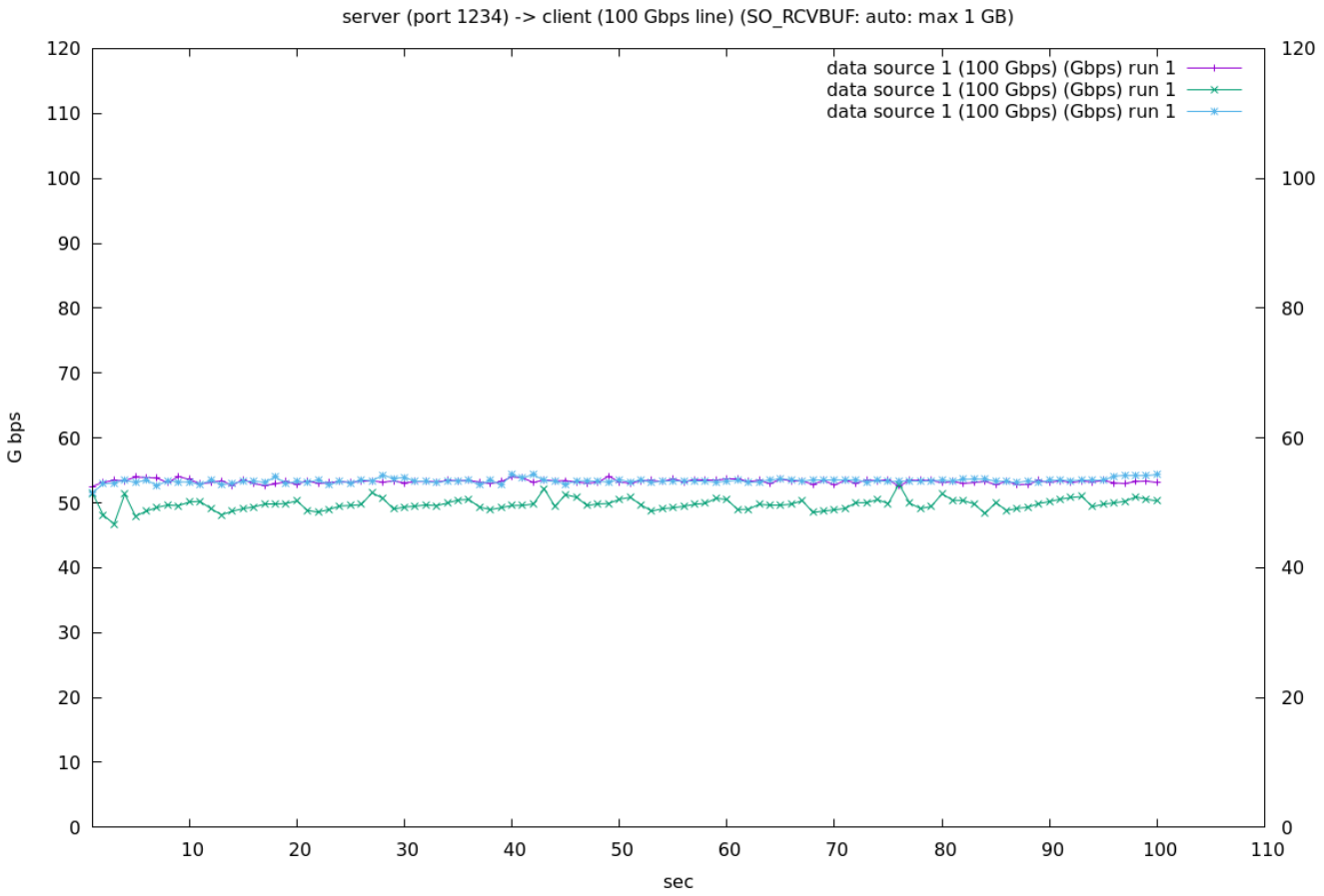
tcp timestamps 無効化とソケットレシーブバッファサイズ（自動調整）run #1



tcp timestamps 無効化とソケットレシーブバッファサイズ（自動調整）run # 2



tcp timestamps無効化
ソケットレシーブバッファ
手動設定
setsockopt()で64MBにセット
実際には２倍の128MBがセットされる。３回やってプロット。



- 高速通信ではTCP Timestampsオプション有効が有利(?)
- SiTCPでは10GbE版もTCP Timestampsが使えないので読み側でソケットレシーブバッファを手動設定である程度回復できそう
- このへんで時間切れ