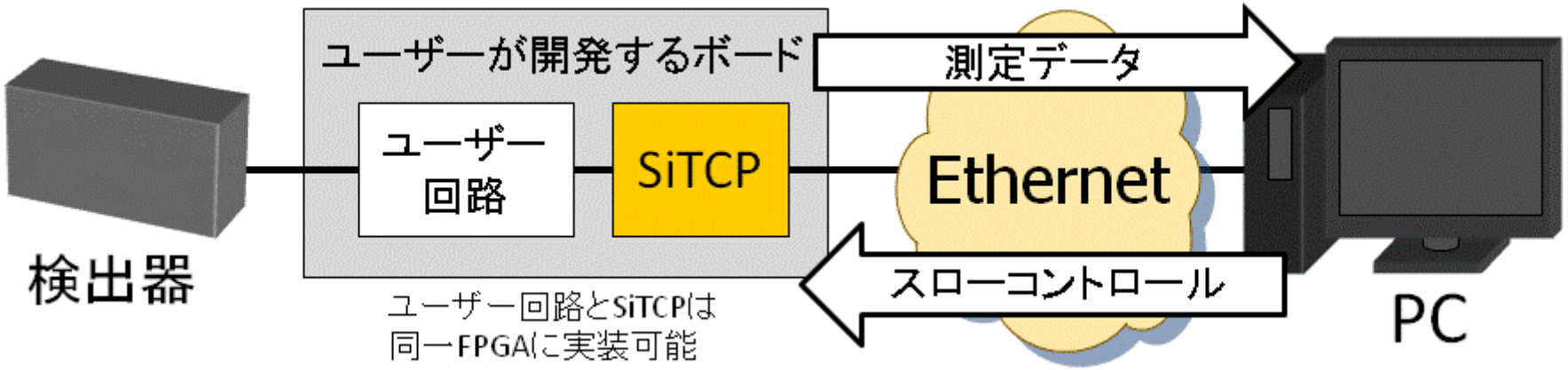
40GbE (10GbE x 4)

千代浩司

KEK/IPNS

SiTCP



Bee Beans Technologies(株)がライセンス
10Mbps, 100Mbps, 1Gbps
10 Gbps (new)

図の出典: ~~http://research.kek.jp/people/uchida/technologies/SiTCP/doc/SiTCP.pdf~~
https://www.sitcp.net/doc/SiTCP.pdf （BBTが運用）

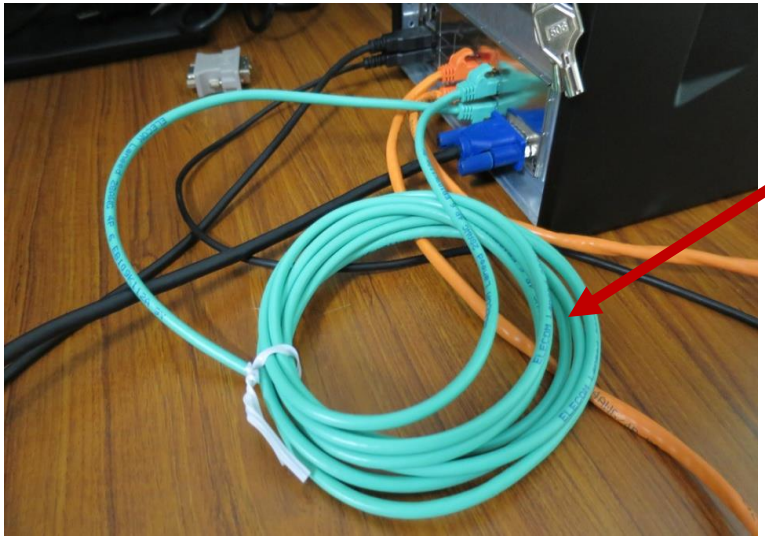
Ethernet

- Ethernet (10Mbps = 10 × 10^6 bps ≠ 10 × 2^20)
- Fast Ethernet (100 Mbps)
- Giga bit Ethernet (1G bps)
- 10 GbE
- 40 GbE
- 100 GbE

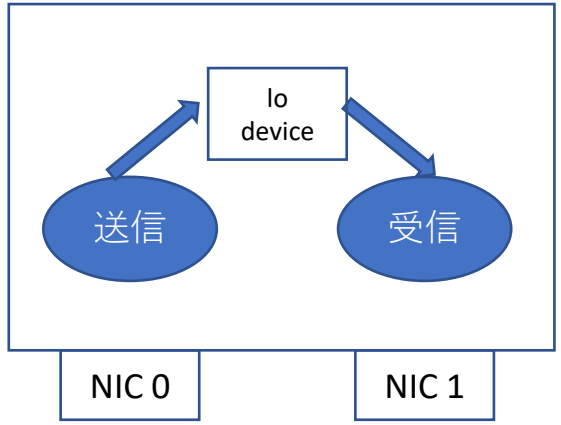
10GbEテスト (2015-12 PCでテスト)

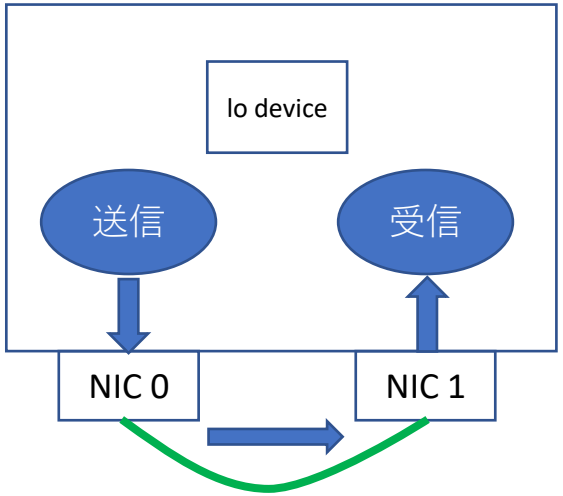
- 10GbE NIC 2個搭載のPCを買った。
- 10GbEで接続できる相手がいない!!!
- 10GbE x 2をケーブルで接続、IPアドレスをつけてもパケットはLANケーブル上を流れないでlo論理デバイスを流れる
- 対応

  その1: Net Namespace
  その2: ルーティングテーブル設定 かつ
  echo 1 >/proc/sys/net/ipv4/conf/net2/accept_local
  echo 1 >/proc/sys/net/ipv4/conf/net3/accept_local



10GbE line





iperf3 Summary

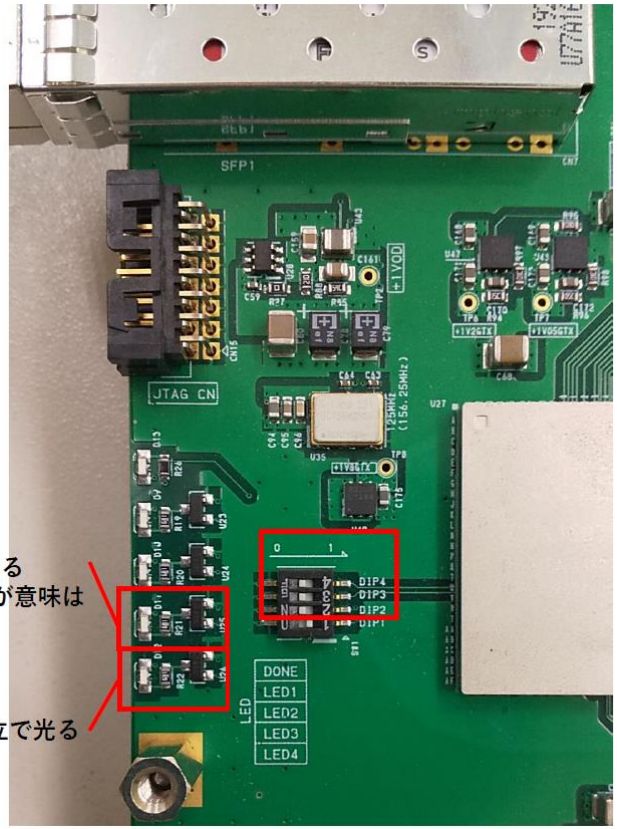
TCP Timestampオプション、TCP Window scaleオプションでの変化

MTU	TCP Timestamp	TCP Window scale	Max. Though Put (Gbps) (計算値)	Through Put (Gbps) (測定値)
1500	ON	ON	9.4148	9.41
1500	OFF	ON	9.4928	9.49
1500	ON	OFF	9.4148	2.09
1500	OFF	OFF	9.4928	2.06
9000	ON	ON	9.9004	9.89
9000	OFF	ON	9.9136	9.91
9000	ON	OFF	9.9004	2.04
9000	OFF	OFF	9.9136	2.13

10GbE ではTCP Window scale optionが必須

J-PARCハドロン実験
連続読み出しDAQ用主回路 AMANEQ

- Esys 本多さん開発
  - 物理学会 実験核物理領域 16pV1-10



DIP
手前側に倒すと0、奥側で1。
DIP2-4で3ビットバイナリを表す
(DIP4, DIP3, DIP2) = 0bXXX
送信スピード

- 0b000 1.25 Gbps
- 0b001 2.50 Gbps
- 0b010 3.75 Gbps
- 0b011 5.00 Gbps
- 0b100 6.25 Gbps
- 0b101 7.50 Gbps
- 0b110 8.75 Gbps
- 0b111 10.0 Gbps

DIP1はSiTCP-XG用。触らない。

本多さんからいただいた資料

10GbE SiTCP Window scale option の確認

- 接続してパケットキャプチャ
- wscale 0を送ってきているのでOK
- PC: 192.168.10.99、 SiTCP: 192.168.10.10

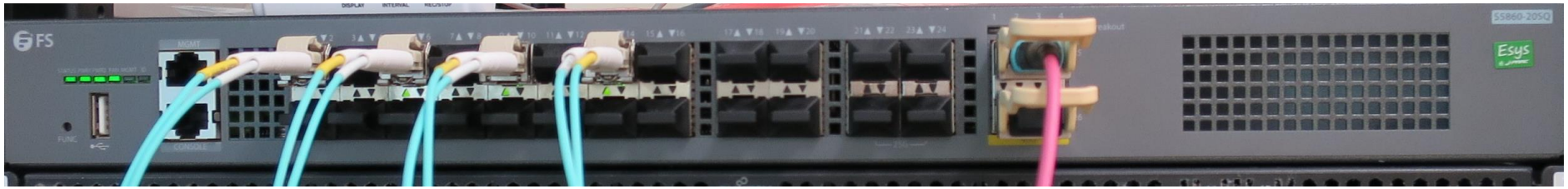
```
192.168.10.99.37910 > 192.168.10.10.24: Flags [S], seq 3626437708,
win 29200, options [mss 1460,sackOK,TS val 3814939925 ecr
0,nop,wscale 9], length 0

192.168.10.10.24 > 192.168.10.99.37910: Flags [S.], seq 0, ack
3626437709, win 65535, options [mss 1460,nop,wscale 0], length 0
```

10GbE x 4 テスト用機材: PC

- マザーボード
  - Supermicro H12SSL-i
- CPU
  - AMD EPYC 7313P 16-Core Processor (基本クロック 3.0 GHz, 最大ブースト 3.7 GHz)
  - 16コア x hyperthreading = 32 コア
- メモリ
  - DIMM DDR4 Synchronous Registered (Buffered) 3200 MHz (0.3 ns) 8GB x 8 = 64GB
- ストレージ
  - NVME 500 GB
- NIC
  - 1GbE x 2 (on board): Broadcom Inc. and subsidiaries NetXtreme BCM5720 Gigabit Ethernet PCIe
  - 40 GbE x 2:Intel Corporation Ethernet Controller XL710 for 40GbE QSFP+ (rev 02)
- IPMIつき (リモートから電源オン、BIOSもさわれるので便利)

10GbE x 4 テスト用機材: ネットワークスイッチ

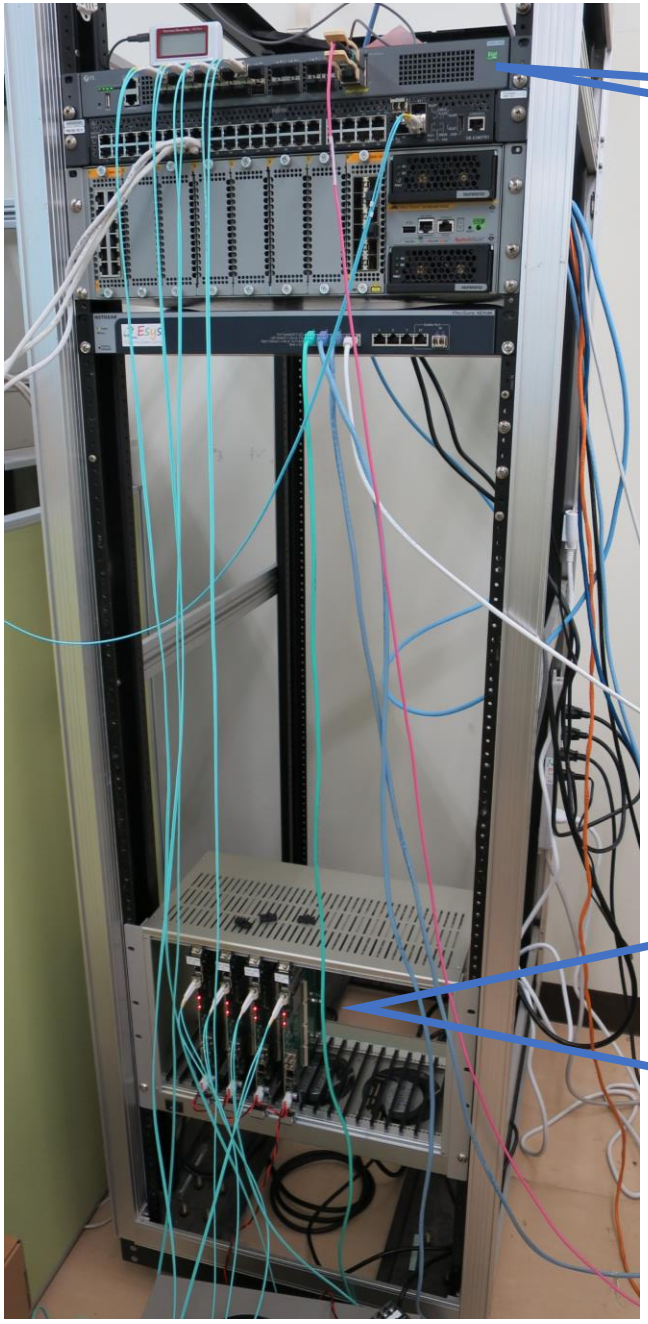
- FS.com S5860-20SQ

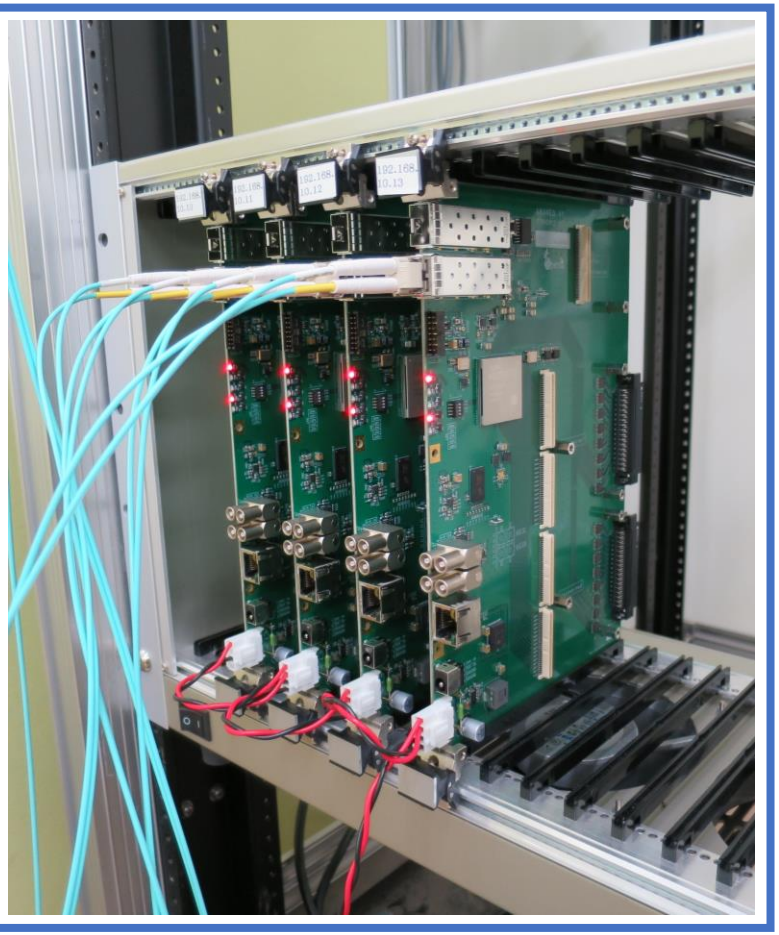


10GbE 10GbE 10GbE 10GbE 40GbE

10GbE SFP (small form-factor pluggable)

40GbE QSFP (Quad small form-factor pluggable)

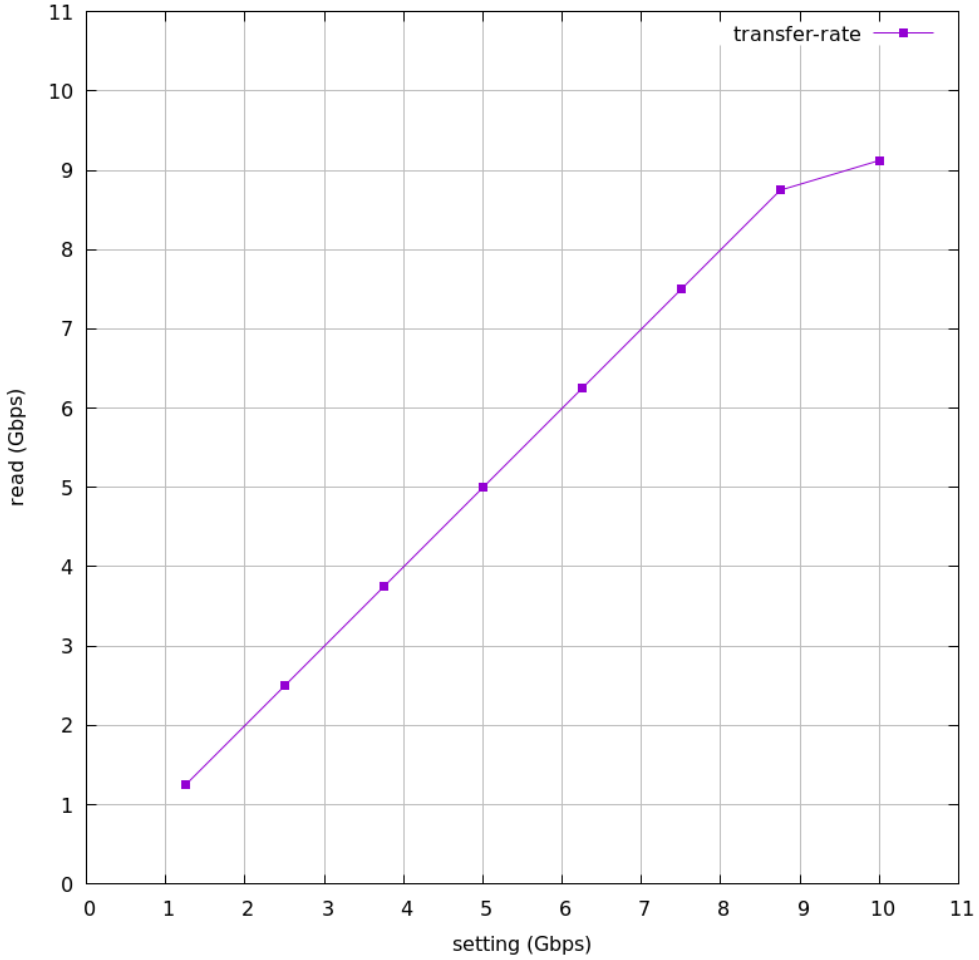




OS セットアップ

- CentOS 8
- JLAN intraという安全地帯に接続しているので
  - firewalld off
  - SELinux off
  - meltdown, spectreのようなCPU脆弱性を緩和する策をoff
    - grub設定ファイルを編集して、カーネルコマンドラインでmitigations=off
    - Intel(R) Core(TM) i7-6800K CPU @ 3.40GHz getppid()について
緩和策有効: 200 ns / 1 getppid()
緩和策無効: 65 ns / 1 getppid()
- ソケットレシーブバッファの最大値 16MB

10GbE 1台転送レート (スイッチ経由)



10秒間読出し、20回測定
括弧内は標準偏差

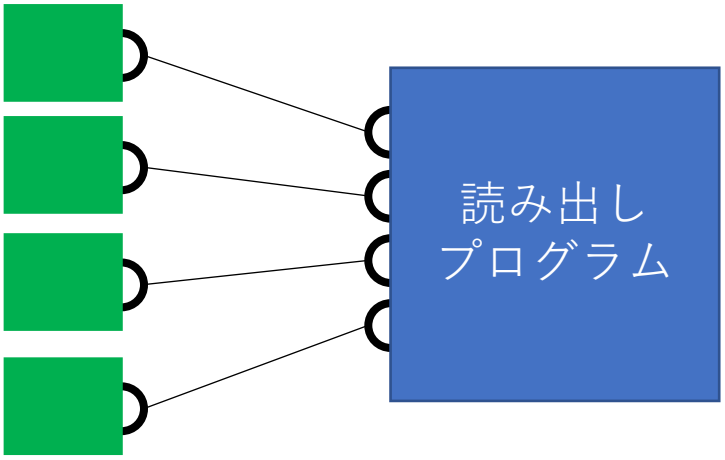
```
 1.25   1.250000 ( 0.000000 )
 2.50   2.500000 ( 0.000000 )
 3.75   3.750000 ( 0.000000 )
 5.00   4.999900 ( 0.000300 )
 6.25   6.249900 ( 0.000436 )
 7.50   7.500000 ( 0.000000 )
 8.75   8.749850 ( 0.000357 )
10.0    9.124950 ( 0.000218 )
```

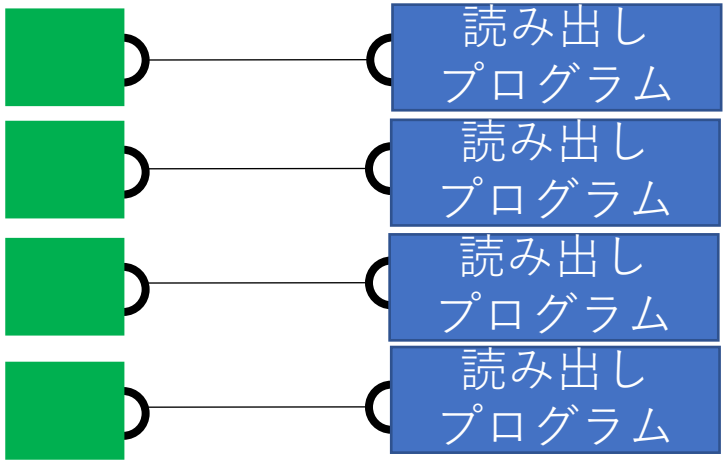
9.12/9.50 = 96% (9.50: TCP最大値)

プログラム:
https://github.com/h-sendai/read-trend

複数ボードの読み出し

- 読み出し方式
  - 1プロセスで多重読出し(select, epoll)
    https://github.com/h-sendai/select-read
    https://github.com/h-sendai/epoll-read
  - マルチプロセス（１プロセス１ボードにはりつけ）
    https://github.com/h-sendai/mp-read

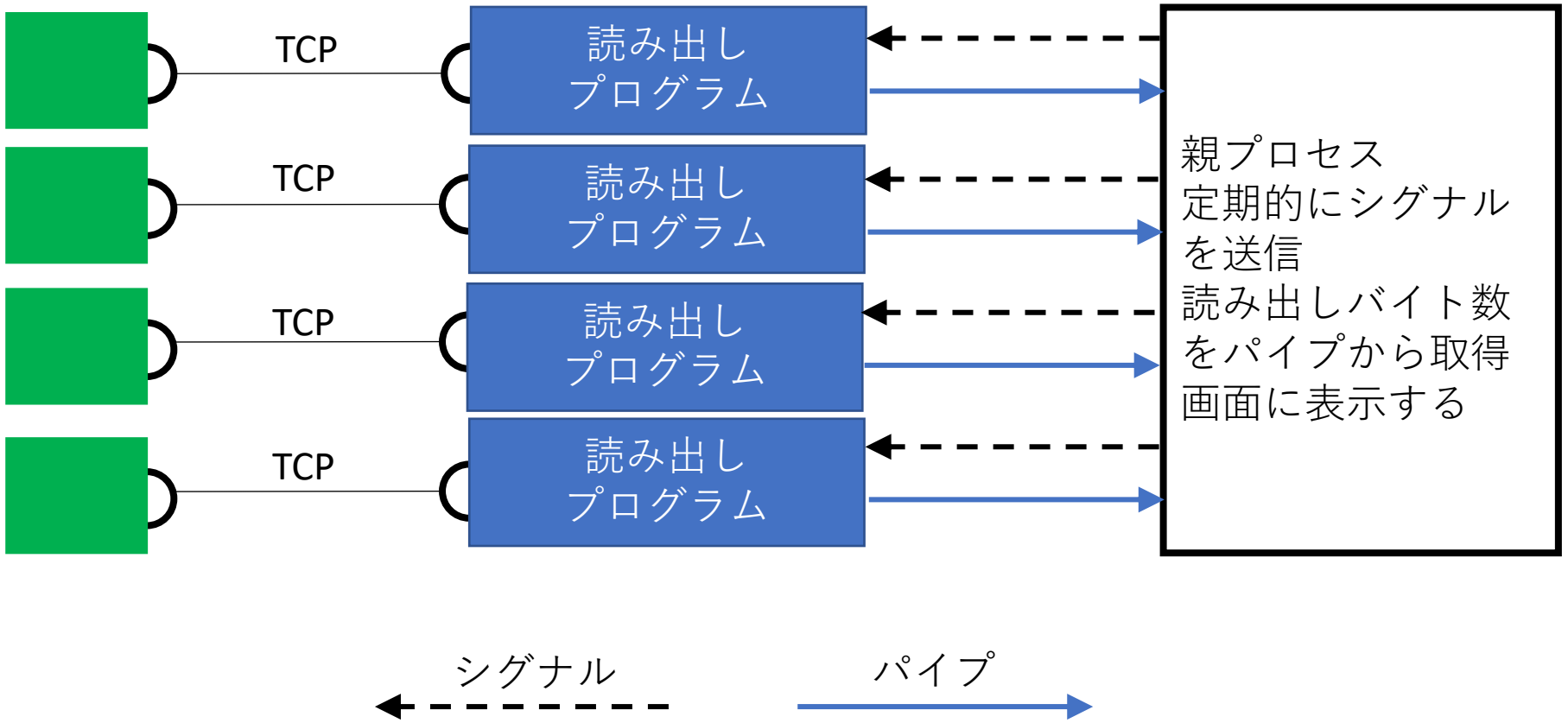




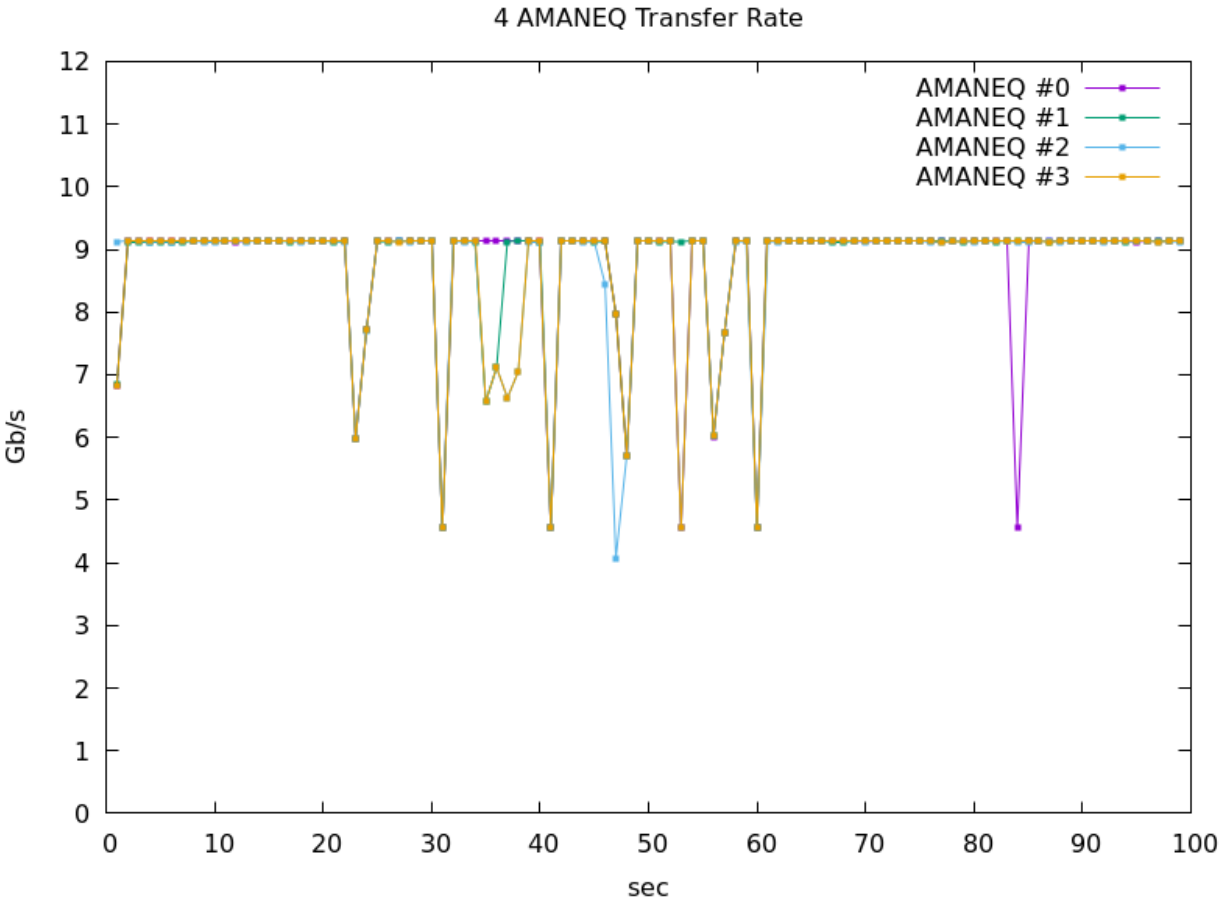
1プロセスで多重読出し

- select, epollとも4台読むにはCPU 1コアではハンドリングできなかった (CPU 1コア全部使い切っている)
- 最大 8 Gbps × 4 = 32 Gbpsくらいで読めてはいた

```
top - 13:14:07 up 20:57,  3 users,  load average: 0.28, 0.07, 0.02
Tasks: 421 total,   2 running, 419 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  3.1 sy,  0.0 ni, 94.1 id,  0.0 wa,  0.0 hi,  2.7 si,  0.0 st
KiB Mem : 65613704 total, 62707968 free,   848776 used,  2056960 buff/cache
KiB Swap: 32972796 total, 32972796 free,        0 used. 64068448 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+  P COMMAND
45076 sendai    20   0   12576   8788   1376 R 100.0   0.0   0:22.93 25 epoll-read
```


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

マルチプロセスの転送レート



各プロセスは1CPUにはりつけ
例: AMANEQ #0を読んでいるプロセスはずっとCPU #1で動作させるようにセットした (tasksetコマンド、sched_setaffinity())

CPU Cn State

- 省電力を目的にCPUにはCstateという状態がありアイドル状態なら電力節約モードに入る。
- 状態はC1, C2などいくつかある（CPUによる）
- Linuxなら/sys/devices/system/cpu/cpuN/cpuidle/stateM/を見る
- Intel Core i 7

```
cpu_num: 0
name     latency (us) disable time (us)      usage          desc
-------- ------------ ------- -------------- -------------- ----------------------
POLL                0       0         1001375         295521 CPUIDLE CORE POLL IDLE
C1                  2       0        37397976        3032498 MWAIT 0x00
C1E                10       0        51381495         936625 MWAIT 0x01
C3                 40       0        27930216         118222 MWAIT 0x10
C6                133       0  1316825716563       24052356 MWAIT 0x20
```

- AMD EPYC 7313P 16-Core Processor

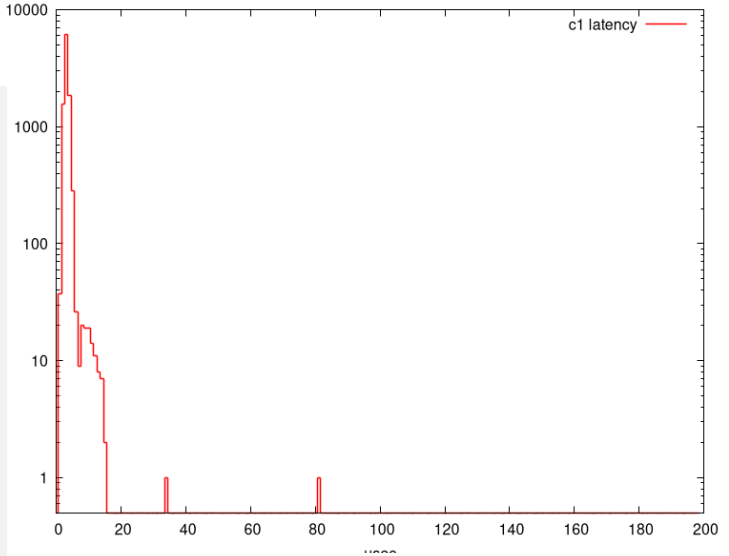
```
cpu_num: 0
name     latency (us) disable time (us)      usage          desc
-------- ------------ ------- -------------- -------------- ----------------------
POLL                0       0           97564            348 CPUIDLE CORE POLL IDLE
C1                  1       0      2253239111         487638 ACPI FFH INTEL MWAIT 0x0
C2                400       0     19991114265         654423 ACPI IOPORT 0x814
```

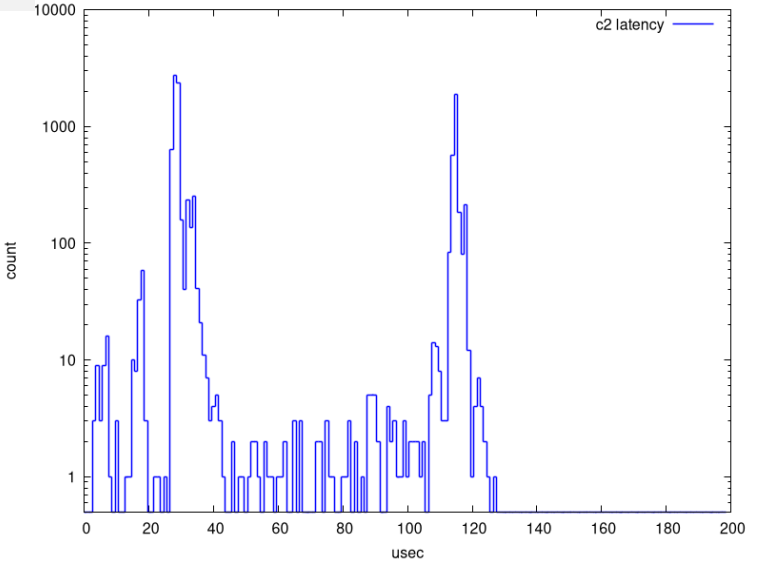
- nが大きいほうが復帰に時間がかかる
- 復帰にかかる時間を測定してみた

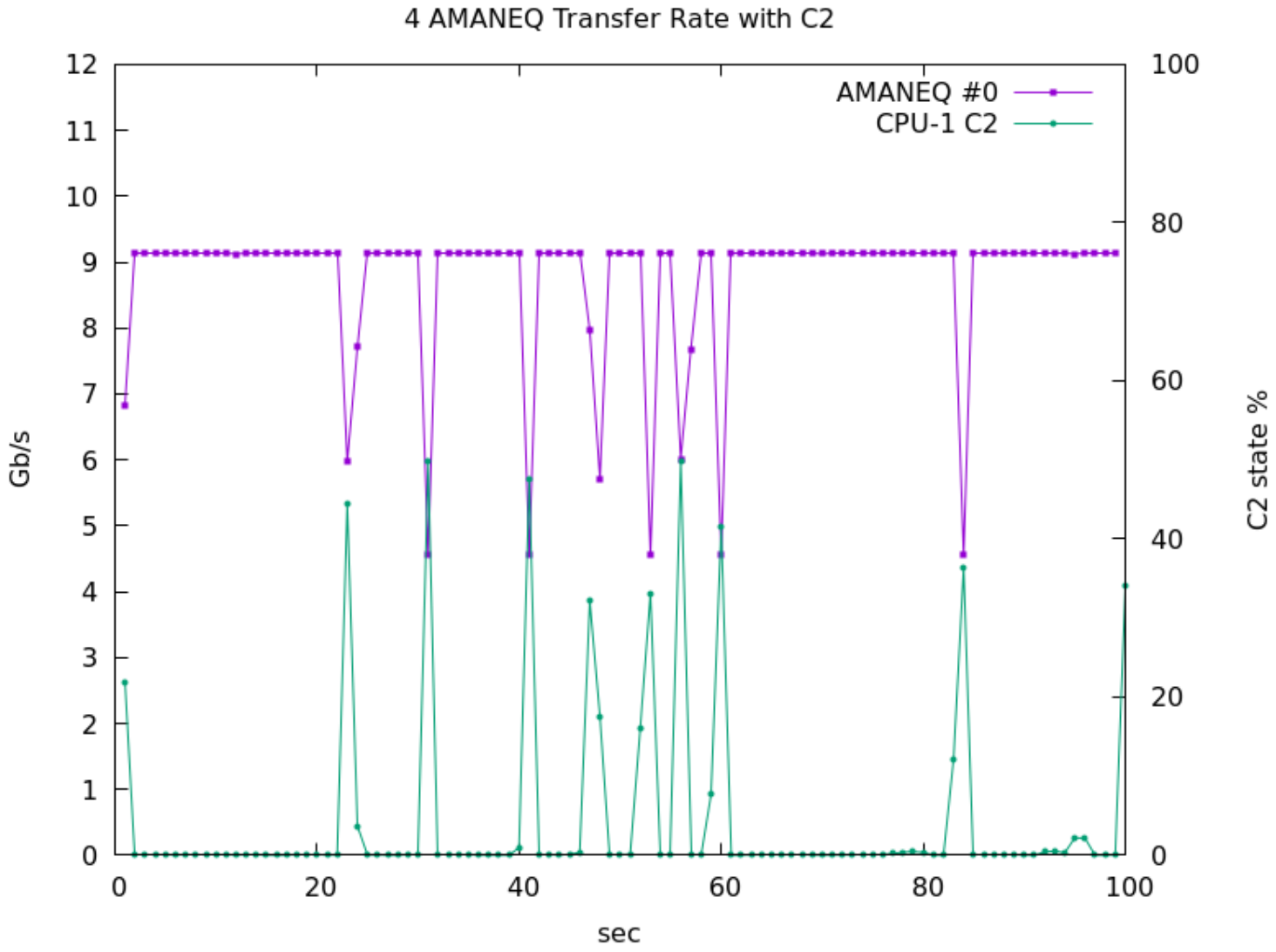
Cnからの復帰にかかる時間の測定

```
// タイマー発火が近いものをまとめることで
// CPU wakeup数を減らし、節電する機能を無効化。
// デフォルトでは50マイクロ秒余計にスリープする
prctl(PR_SET_TIMERSLACK, 1);

clock_gettime(CLOCK_MONOTONIC, &ts0);
usleep(10 000); // 10ミリ秒スリープ
clock_gettime(CLOCK_MONOTONIC, &ts1)
ts1 - ts0の計算
```

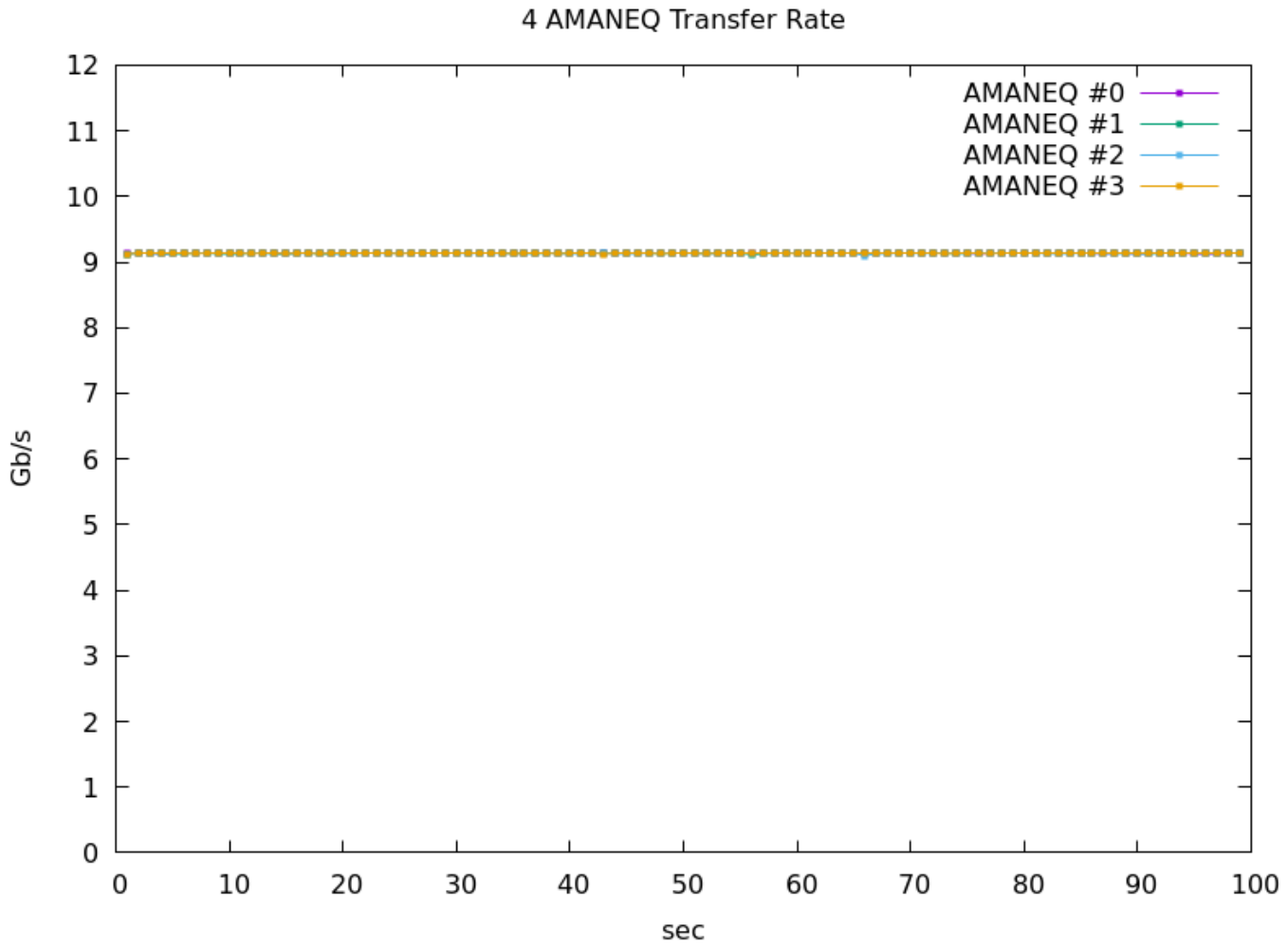
ts1 - ts0はusleep(10 000)の10ミリ秒と
Cn ステートからの復帰にかかる時間の
和になる。
横軸10ミリ秒を引いてプロット






4 AMANEQ Transfer Rate with C2
AMANEQ #0
CPU-1 C2
Gb/s
C2 state %
sec
0
1
2
3
4
5
6
7
8
9
10
11
12
0
20
40
60
80
100
0
20
40
60
80
100

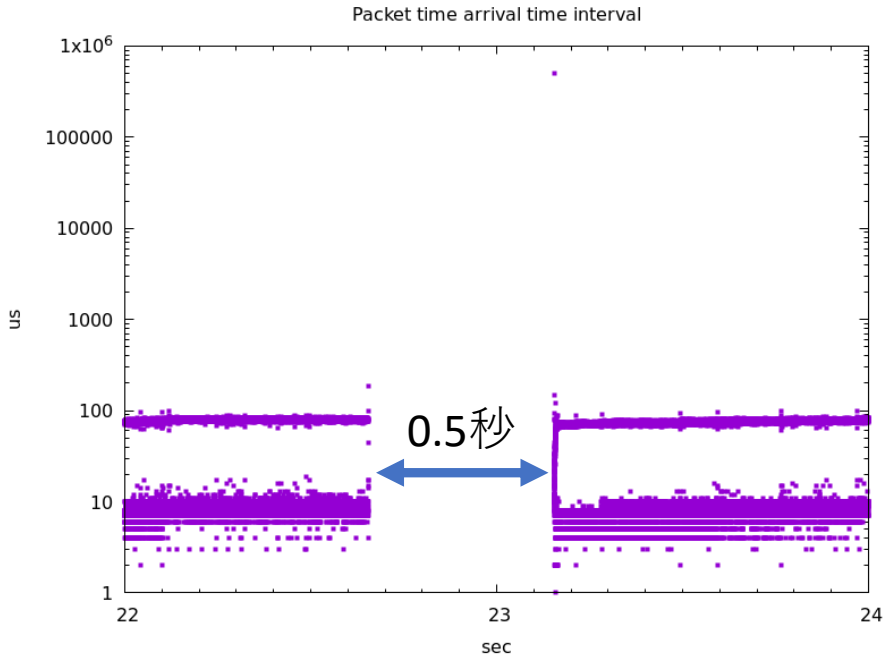
最大CnをC1にセット

- 

が落ちることもある。

なぜC1/C2に落ちるのか

- パケット到着時間間隔を見てみる
- パケットが0.5秒間来ないことがある
- read()でブロック（データがくるまで待っている）



時刻	前のパケットからの経過時間
23.154726	0.500020
30.660708	0.500245
40.652581	0.500748
47.373612	0.500495
52.543604	0.500316
56.159721	0.500304
59.650720	0.500269
83.689565	0.500245

SiTCP文書

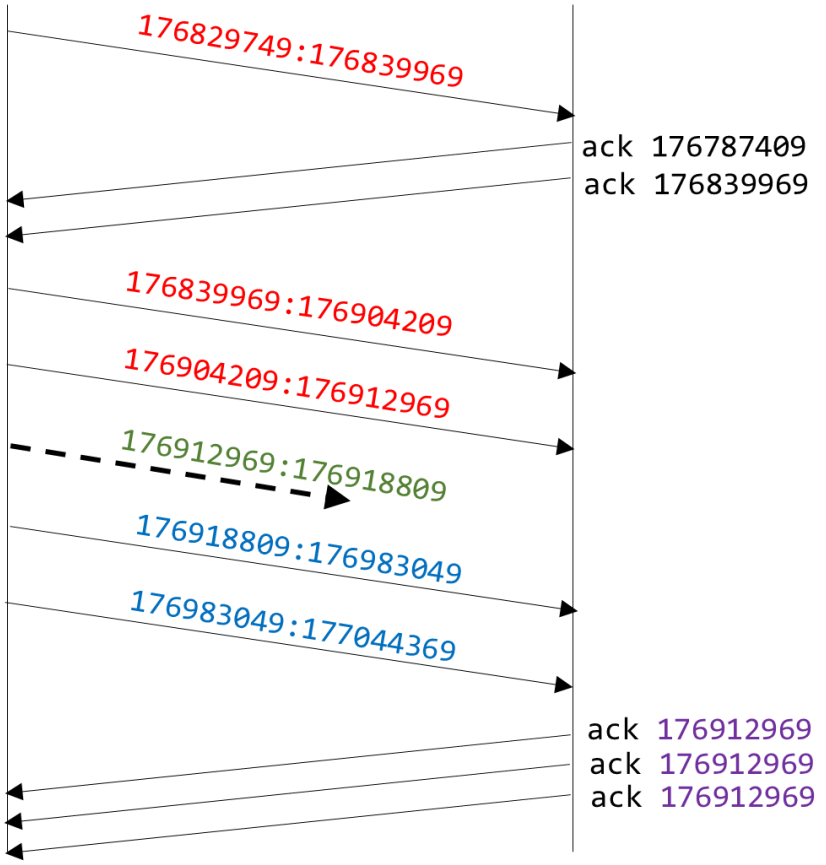
+0x11	R/W	Reserved	
+0x12-17	R/W	MAC address (*1)	
+0x18-1B	R/W	IP address	
+0x1C-1D	R/W	TCP ポート番号（main port）	0x18
+0x1E-1F	R/W	TCP ポート番号（alternative port）	0x17
+0x20-21	R/W	TCP MSS (バイト数)	0x05B4
+0x22-23	R/W	RBCP ポート番号	0x1234
+0x24-25	R/W	TCP キープアライブパケット送信タイマ（送信バッファにデータがある時）（msec 単位）	0x03E8
+0x26-27	R/W	TCP キープアライブパケット送信タイマ（送信バッファが空の時）（msec 単位）	0xEA60
+0x28-29	R/W	TCP オープン時のタイムアウト時間（msec 単位）	0x1388
+0x2A-2B	R/W	TCP クローズ時のタイムアウト時間（256msec 単位）	0x2BF2
+0x2C-2D	R/W	TCP コネクション-コネクション間の待機時間（msec 単位）	0x01F4
+0x2E-2F	R/W	TCP 再送時間タイムアウト時間（msec 単位）	0x01F4
+0x30-3B	R/W	Reserved	0x0
+0x3C-3F	R/W	ユーザー領域 (*2)	0x00000000
+0x40-FF		アクセス禁止領域 (*1)	

0x01F4 = 500

- パケット消失、再送が起きている。
- 読み出しPC側のどこかのキューが足りてない?

```
192.168.10.12.24 > 192.168.10.99.57958: Flags [.], seq 176829749:176839969, ack 1, win 65520, length 10220
192.168.10.99.57958 > 192.168.10.12.24: Flags [.], ack 176787409, win 9266, length 0
192.168.10.99.57958 > 192.168.10.12.24: Flags [.], ack 176839969, win 9468, length 0
192.168.10.12.24 > 192.168.10.99.57958: Flags [.], seq 176839969:176904209, ack 1, win 65520, length 64240
192.168.10.12.24 > 192.168.10.99.57958: Flags [.], seq 176904209:176912969, ack 1, win 65520, length 8760
192.168.10.12.24 > 192.168.10.99.57958: Flags [.], seq 176918809:176983049, ack 1, win 65520, length 64240
192.168.10.12.24 > 192.168.10.99.57958: Flags [.], seq 176983049:177044369, ack 1, win 65520, length 61320
192.168.10.99.57958 > 192.168.10.12.24: Flags [.], ack 176912969, win 8898, length 0
192.168.10.99.57958 > 192.168.10.12.24: Flags [.], ack 176912969, win 8898, length 0
192.168.10.99.57958 > 192.168.10.12.24: Flags [.], ack 176912969, win 8898, length 0
```

176918809-176912969 = 5840バイト分データが飛んでいる
0.5秒後に再送が起こる。0.5秒: SiTCP-XG デフォルトの設定が0.5秒。



ethtool -S enp193s0f0 でPC NICの統計情報が取得できる。
そのなかにドロップ数がある。
ラン前: port.rx_dropped: 4472
ラン後: port.rx_dropped: 4495

ネットワークスイッチの統計情報 (port #1: amaneq0接続)

FS# **show interface tenGigabitEthernet 0/1 counters**

Interface : TenGigabitEthernet 0/1
10 seconds input rate :0 bits/sec, 0 packets/sec
10 seconds output rate :144 bits/sec, 0 packets/sec
Rxload : 0%
InOctets : 89477425895444
InPkts : 58950606159 (Unicast: 100%, Multicast: 0%, Broadcast: 0%)
InUcastPkts : 58950606159
InMulticastPkts : 0
InBroadcastPkts : 0
Txload : 0%
OutOctets : 106665493252
OutPkts : 1664805665 (Unicast: 99%, Multicast: 0%, Broadcast: 0%)
OutUcastPkts : 1664079673
OutMulticastPkts : 483918
OutBroadcastPkts : 12893
Undersize packets : 2
Oversize packets : 0
collisions : 0
Fragments : 0
Jabbers : 0
CRC alignment errors : 0
AlignmentErrors : 0
FCSErrors : 0
dropped packet events (due to lack of resources): 0

packets received of length (in octets):
64 : 2864400
65-127 : 848445
128-255 : 448864
256-511 : 289405
512-1023 : 3898647
1024-1518 : 58942256398
Interface : TenGigabitEthernet 0/1
Packet increment in last sampling interval(6061 milliseconds):
InOctets : 0
InPkts : 0 (Unicast: 0%, Multicast: 0%, Broadcast: 0%)
InUcastPkts : 0
InMulticastPkts : 0
InBroadcastPkts : 0
OutOctets : 273
OutPkts : 1 (Unicast: 0%, Multicast: 100%, Broadcast: 0%)
OutUcastPkts : 0
OutMulticastPkts : 1
OutBroadcastPkts : 0

amaneq1 - 3のポートも
同様にドロップなしとなっている

ネットワークスイッチ上の統計情報 (PC: 40GbE接続)

FS# **show interface FortyGigabitEthernet 0/25 counters**

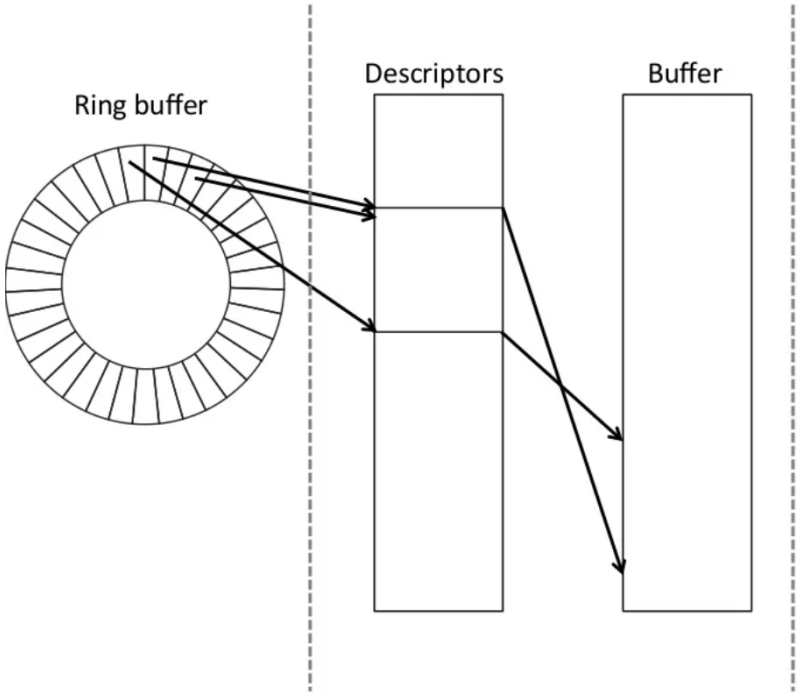
Interface : FortyGigabitEthernet 0/25
10 seconds input rate :0 bits/sec, 0 packets/sec
10 seconds output rate :0 bits/sec, 0 packets/sec
Rxload : 0%
InOctets : 795107287634
InPkts : 12422210002 (Unicast: 99%, Multicast: 0%, Broadcast: 0%)
InUcastPkts : 12421710583
InMulticastPkts : 453450
InBroadcastPkts : 353
Txload : 0%
OutOctets : 626198344423168
OutPkts : 412525715717 (Unicast: 100%, Multicast: 0%, Broadcast: 0%)
OutUcastPkts : 412525255258
OutMulticastPkts : 447878
OutBroadcastPkts : 12581
Undersize packets : 0
Oversize packets : 0
collisions : 0
Fragments : 0
Jabbers : 0
CRC alignment errors : 0
AlignmentErrors : 0
FCSErrors : 0
dropped packet events (due to lack of resources): 0

packets received of length (in octets):
64 : 12409426298
65-127 : 12783176
128-255 : 313
256-511 : 97
512-1023 : 118
1024-1518 : 0
Interface : FortyGigabitEthernet 0/25
Packet increment in last sampling interval(6075 milliseconds):
InOctets : 0
InPkts : 0 (Unicast: 0%, Multicast: 0%, Broadcast: 0%)
InUcastPkts : 0
InMulticastPkts : 0
InBroadcastPkts : 0
OutOctets : 0
OutPkts : 0 (Unicast: 0%, Multicast: 0%, Broadcast: 0%)
OutUcastPkts : 0
OutMulticastPkts : 0
OutBroadcastPkts : 0

https://www.kernel.org/doc/Documentation/networking/i40e.rst

Flow Control
Ethernet Flow Control (IEEE 802.3x) can be configured with ethtool to enable receiving and transmitting pause frames for i40e. When transmit is enabled, pause frames are generated when the receive packet buffer crosses a predefined threshold. When receive is enabled, the transmit unit will halt for the time delay specified when a pause frame is received.
NOTE: You must have a flow control capable link partner.
Flow Control is on by default.
Use ethtool to change the flow control settings.
To enable or disable Rx or Tx Flow Control::
ethtool -A eth? rx <on|off> tx <on|off>

```
esyst-daq01% ethtool -g enp193s0f0
Ring parameters for enp193s0f0:
Pre-set maximums:
RX:             4096
RX Mini:        n/a
RX Jumbo:       n/a
TX:             4096
Current hardware settings:
RX:             4096
RX Mini:        n/a
RX Jumbo:       n/a
TX:             4096
```



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

スイッチ設定　pauseの有効化

- flow control (pause)が無効になっている（デフォルトで無効）
- コントロールポート（デフォルト192.168.1.1）にsshログインして設定

```
FS# config t
port 1の設定
FS(config)# interface tenGigabitEthernet 0/1   (0/1: 1: ポート番号。40GbEのポートは25, 26)
FS(config-if-TenGigabitEthernet 0/1)# flowcontrol on

以下3個分同様に設定

ケーブル抜き差し
40GbEポートも設定
pauseが無効になっていたら手動で設定
esyst-daq01# ethtool -A enp193f0 rx on tx on
```

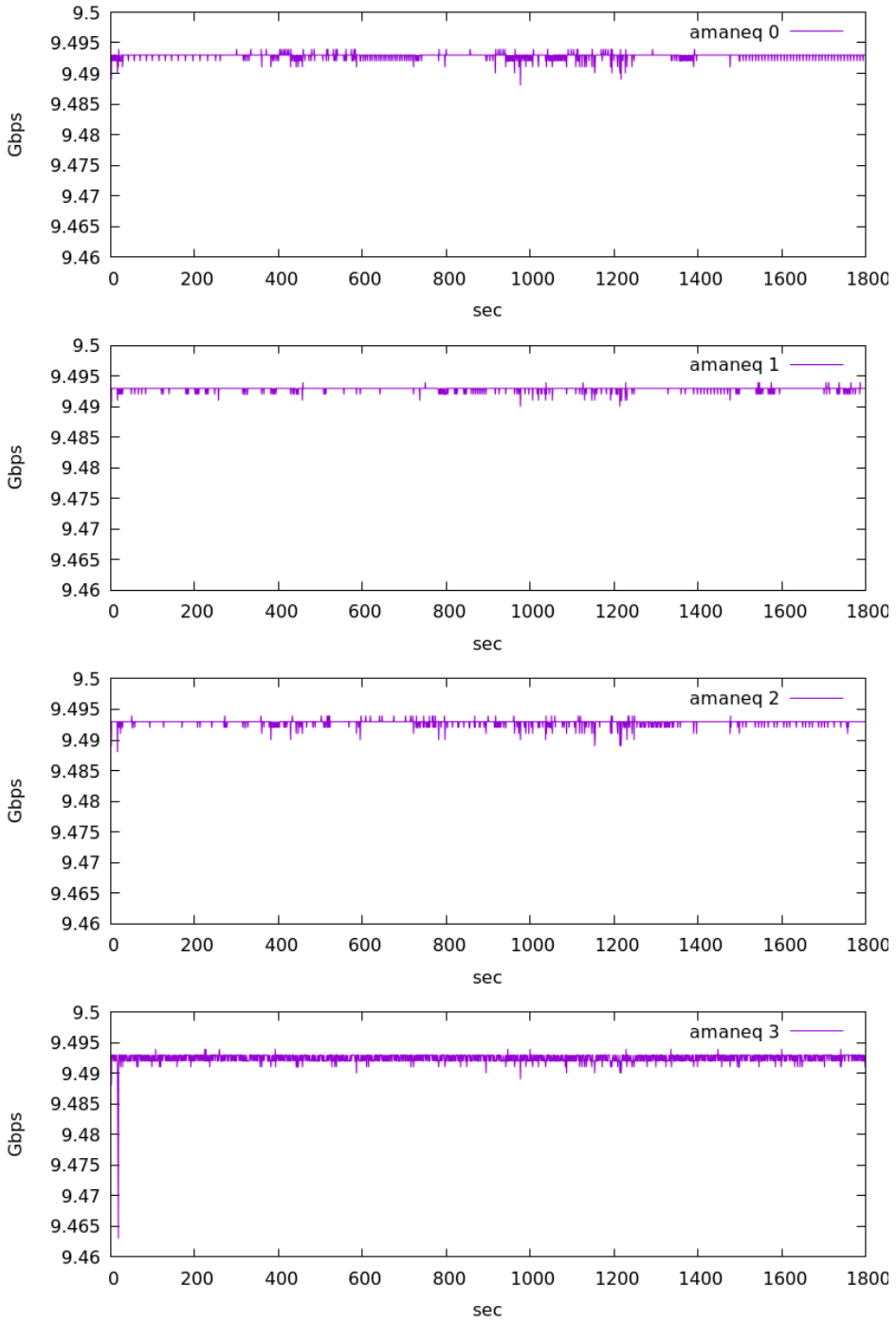
Ethernet Pauseを送った回数

- ethtool -S enp193s0f0に port.link_xon_tx, xon_tx がある。これがpauseを送った回数か？
  スイッチ設定前はいつも０だった。

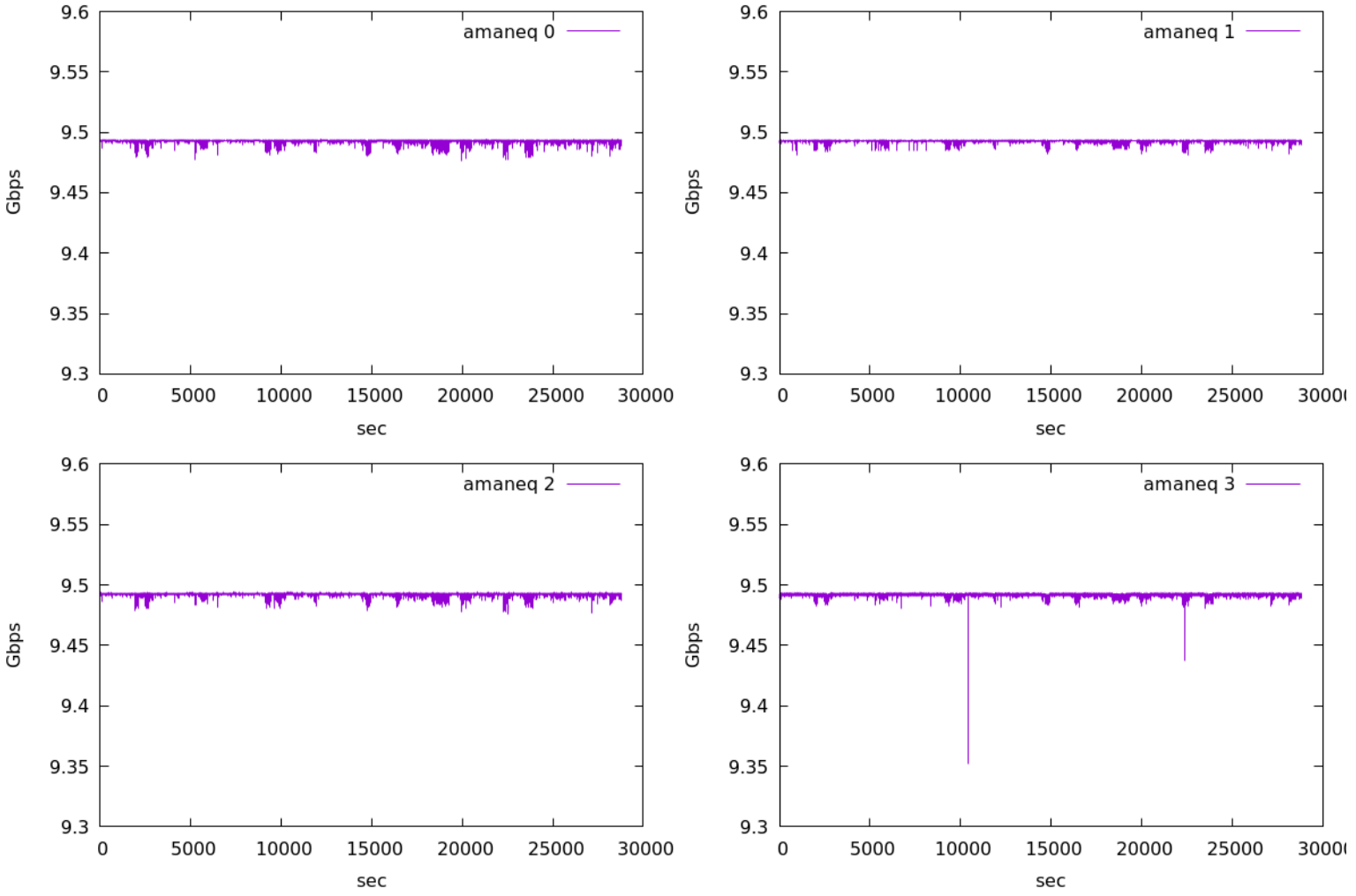
  ラン前: port.link_xon_tx: 280
  port.link_xoff_tx: 280
  ラン後: port.link_xon_tx: 397
  port.link_xoff_tx: 397

port.rx_droppedは増加なし

30分（1800秒）の例

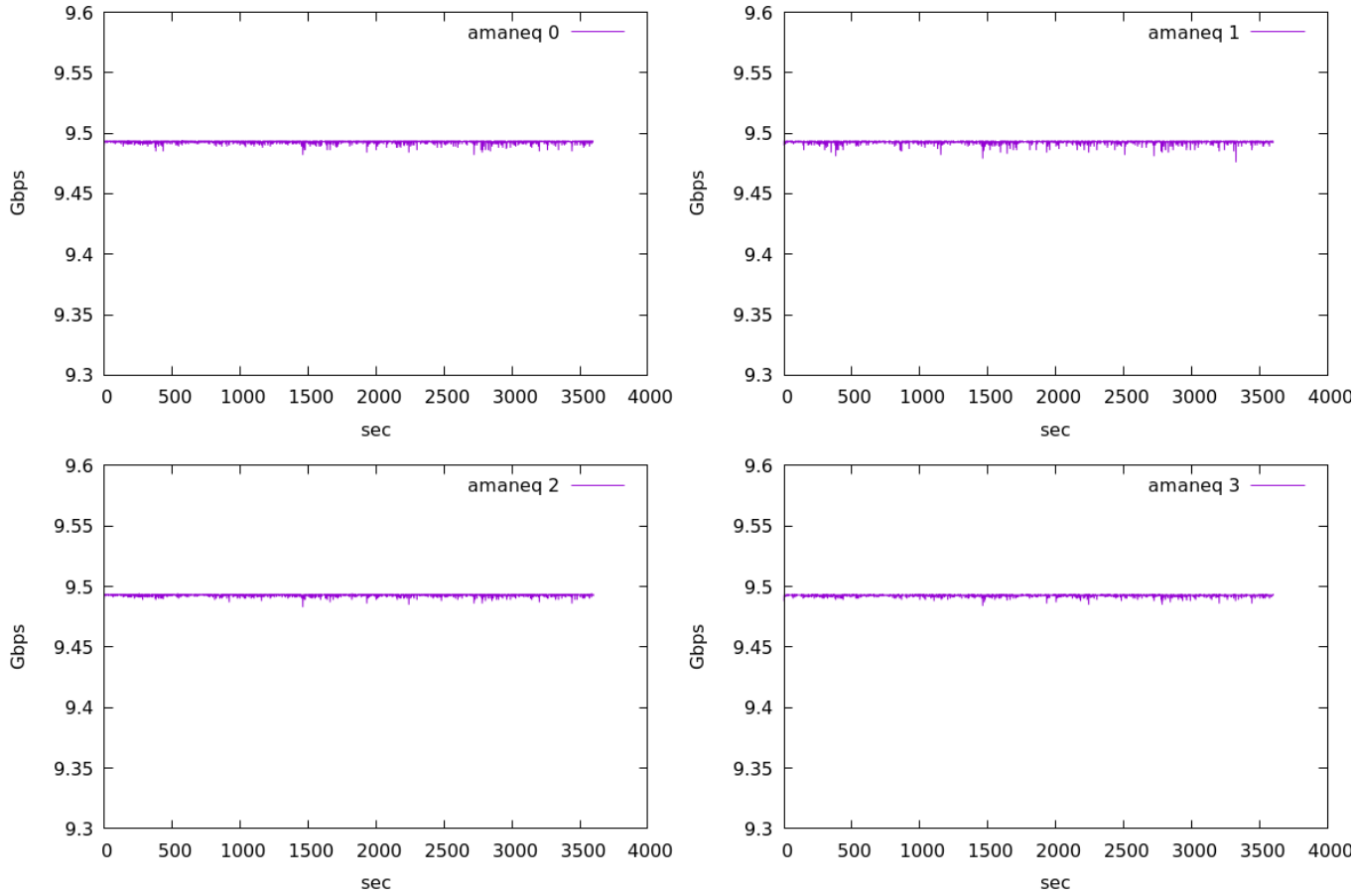


read(sockfd, buf, bufsize) (bufsize: 2MBで読める分読む方式)



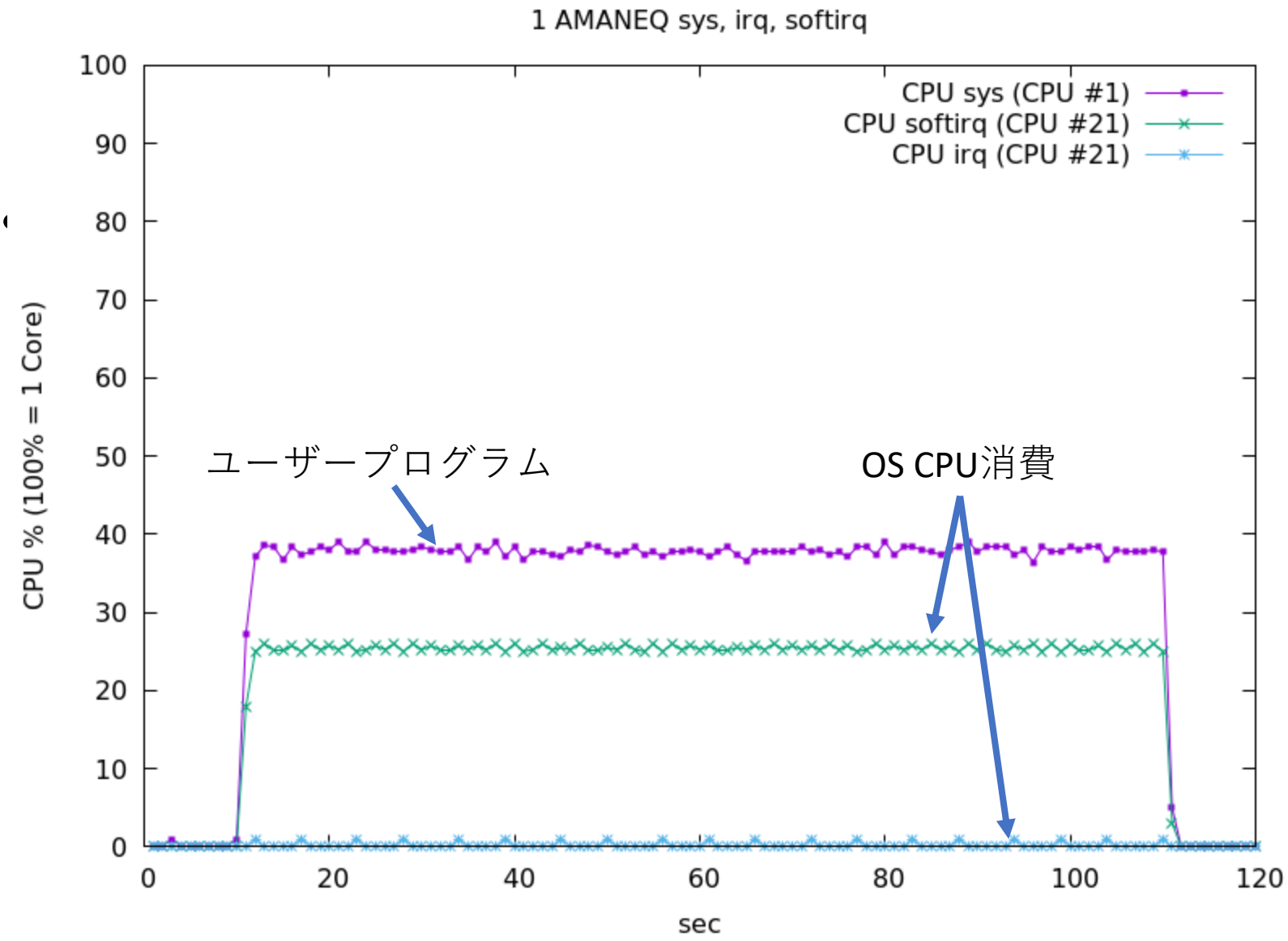
port.rx_dropped: 0
port.link_xoff_tx: 16539 (0.57回/sec)

データ検証ありで読む



- データ欠落なし
- パケットドロップなし
- xoff_tx 1401回/hr (0.39回/sec)

24時間読出しもOKだった


1 AMANEQ sys, irq, softirq
CPU sys (CPU #1)
CPU softirq (CPU #21)
CPU irq (CPU #21)
CPU % (100% = 1 Core)
sec
ユーザープログラム
OS CPU消費
0
10
20
30
40
50
60
70
80
90
100
0
20
40
60
80
100
120

まとめ

- 10GbE を読むのはそんなにむずかしくなさそう
- 4 x 10GbEの読み方
  - 多重読出し（select, epoll）は無理
  - 10GbEボード1台に1プロセス張り付ける方法ならOK
  - スイッチのフローコントロールを有効にしないとパケット消失が発生する

スイッチ設定　pauseの有効化

- flow control (pause)が無効になっている（デフォルトで無効）
- コントロールポート（デフォルト192.168.1.1）にssh (admin@192.168.1.1、パスワードadmin)

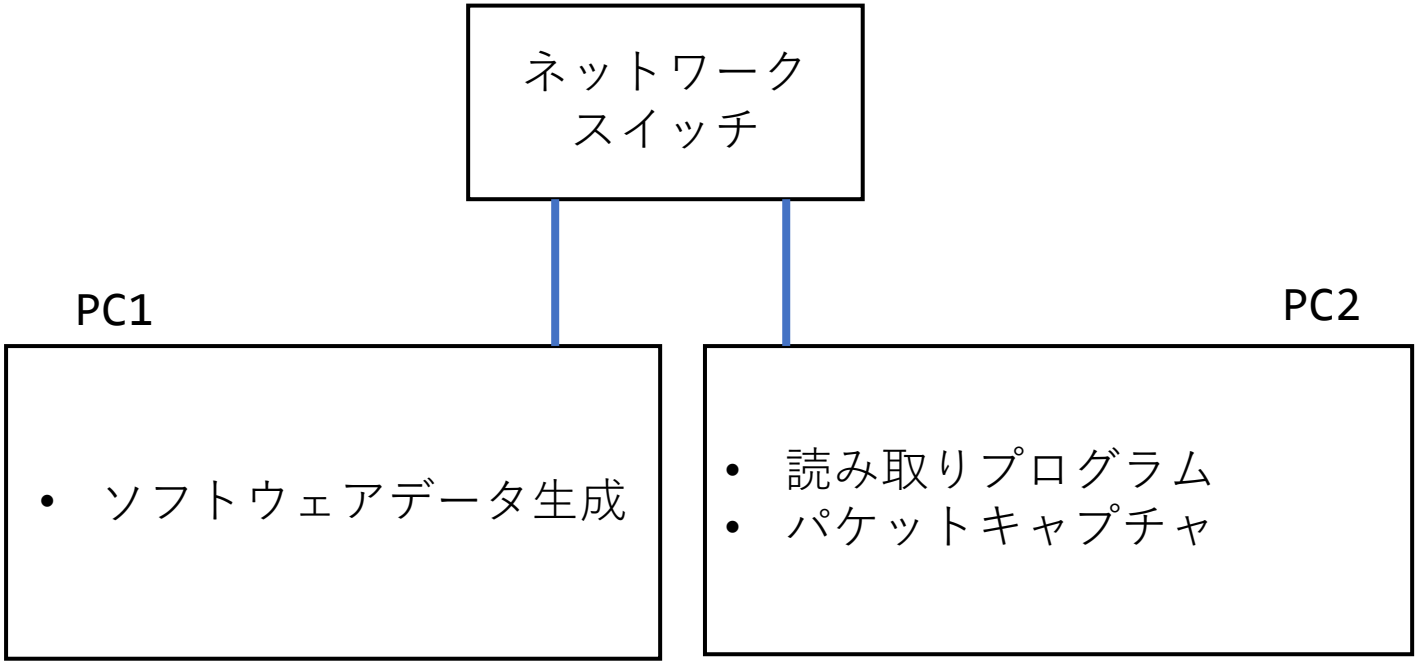
```
FS# config t
port 1の設定
FS(config)# interface tenGigabitEthernet 0/1   (0/1: 1: ポート番号。40GbEのポートは25, 26)
FS(config-if-TenGigabitEthernet 0/1)# flowcontrol on

以下3個分同様に設定

ケーブル抜き差し
40GbEポートも設定
pauseが無効になっていたら手動で設定
esyst-daq01# ethtool -A enp193f0 rx on tx on
```

スイッチの設定を行わなくても
flow controlを有効にできないか？

調査方法

- データサーバー（ソフトウェア、あるいはamaneq）からデータを読んでいる途中に1秒間隔でpauseパケットを送ってみる。
- 読出し中にtcpdumpでキャプチャしておいてパケットがやってくる時間間隔を調べる。
- pauseが有効に働いていれば時間間隔が大きいところがあるはず。
- スイッチでflow control on/offにして違いを調査
  - offの場合はethtool -A eth0 rx on tx onでNIC側flow controlを有効化
- 1GbE (銅線), 10GbE (銅線、光), 40GbE (光)で調査

テスト構成



Ethernet Pause frame

LinuxでEthernet frameを送信する: man 7 packet

NAME

packet - packet interface on device level

SYNOPSIS

```
#include <sys/socket.h>
#include <linux/if_packet.h>
#include <net/ethernet.h> /* the L2 protocols */

packet_socket = socket(AF_PACKET, int socket_type, int protocol);
```

DESCRIPTION

Packet sockets are used to receive or send raw packets at the device driver (OSI Layer 2) level. They allow the user to implement protocol modules in user space on top of the physical layer.

The socket_type is either **SOCK_RAW** for raw packets including the link-level header or **SOCK_DGRAM** for cooked packets with the link-level header removed. The link-level header information is available in a common format in a sockaddr_ll structure. protocol is the IEEE 802.3 protocol number in network byte order. See the <linux/if_ether.h> include file for a list of allowed protocols. When protocol is set to **htons(ETH_P_ALL)**, then all protocols are received. All incoming packets of that protocol type will be passed to the packet socket before they are passed to the protocols implemented in the kernel.

プログラムからpause frameを投げる

```
int sockfd = socket(AF_PACKET, SOCK_DGRAM, htons(ETH_P_PAUSE));

/* Determine the index number of the Ethernet interface to be used. */
unsigned int if_index = if_nametoindex(if_name); /* if_name: eth0 etc */

/* Construct the destination address */
struct sockaddr_ll addr;
memset(&addr, 0, sizeof(addr));

addr.sll_family   = AF_PACKET;
addr.sll_ifindex  = if_index;
addr.sll_halen    = ETHER_ADDR_LEN;
addr.sll_protocol = htons(ETH_P_PAUSE);
addr.sll_addr[0]  = 0x01;
addr.sll_addr[1]  = 0x80;
addr.sll_addr[2]  = 0xc2;
addr.sll_addr[3]  = 0x00;
addr.sll_addr[4]  = 0x00;
addr.sll_addr[5]  = 0x01;

unsigned char en_payload[4];
en_payload[0] = 0x00;
en_payload[1] = 0x01;
en_payload[2] = pause_time >> 8;
en_payload[3] = pause_time;

/* Ethernet 最小ペイロードサイズ（46バイト）になる処理はOS側がやってくれるようで
自前でパディングする必要はない */

/* Send the Ethernet frame. */
sendto(sockfd, en_payload, sizeof(en_payload), 0, (struct sockaddr *)&addr, sizeof(addr));
```

```
struct sockaddr_ll {
    unsigned short sll_family;   /* Always AF_PACKET */
    unsigned short sll_protocol; /* Physical-layer protocol */
    int            sll_ifindex;  /* Interface number */
    unsigned short sll_hatype;   /* ARP hardware type */
    unsigned char  sll_pkttype;  /* Packet type */
    unsigned char  sll_halen;    /* Length of address */
    unsigned char  sll_addr[8];  /* Physical-layer address */
};
```

```
Pause frame format
+---------+--------+---------+-------+--------------+---------+--------+
| dest(6) | src(6) | type(2) | op(2) | pausetime(2) | pad(42) | FCS(4) |
+---------+--------+---------+-------+--------------+---------+--------+
                              |<----- Ethernet Payload ------> |
dest:      01:80:C2:00:00:01
type:      0x8808
op code:   0x0001
pausetime: 0 - 65535
```

- 前ページのプログラムでpause frameを送ってもethtool -S eth0 の tx_flow_control_xoffなどにはカウントされない
- ドライバが投げるpause frameは送受信ともtcpdumpでキャプチャできないが、前ページのプログラムでなげると送ったPCではキャプチャ可能
- padding部分はpause frameを送ったPCのtcpdumpには出ない

```
09:42:52.494362 00:15:17:1c:ef:9d > 01:80:c2:00:00:01, ethertype MPCP (0x8808),
length 18:
    [|MPCP]
    0x0000:  0180 c200 0001 0015 171c ef9d 8808 0001  ................
    0x0010:  ffff                                     ..
```

```
Pause frame format
+---------+--------+---------+-------+--------------+---------+--------+
| dest(6) | src(6) | type(2) | op(2) | pausetime(2) | pad(42) | FCS(4) |
+---------+--------+---------+-------+--------------+---------+--------+
                              |<----- Ethernet Payload ------> |
dest:       01:80:C2:00:00:01
type:       0x8808
op code:    0x0001
pausetime: 0 - 65535
```

close()にかかる時間

socket	平均 (ミリ秒)
socket(AF_INET, SOCK_DGRAM, 0)	0.49
socket(AF_PACKET, SOCK_DGRAM, htons(ETH_P_PAUSE));	8.24

pauseを送るのに使うソケットを毎度毎度close()するとシングルスレッドではそこで動作が8ms程度止まるので注意。

```
/* 計測プログラム */
int do_socket()
{
    int sockfd = socket(AF_PACKET, SOCK_DGRAM, htons(ETH_P_PAUSE));
    if (sockfd < 0) {
        err(1, "socket");
    }

    struct timeval tv0, tv1, diff;
    gettimeofday(&tv0, NULL);
    int n = close(sockfd);
    gettimeofday(&tv1, NULL);
    if (n < 0) {
        err(1, "close");
    }
    timersub(&tv1, &tv0, &diff);
    printf("close: %ld usec\n", 1000000*diff.tv_sec + diff.tv_usec);

    return 0;
}
```

pause_time	65535	32768
1 GbEでの実時間	33.5ms	16.8ms
10GbEでの実時間	3.35ms	1.68ms
40GbEでの実時間	838us	419us

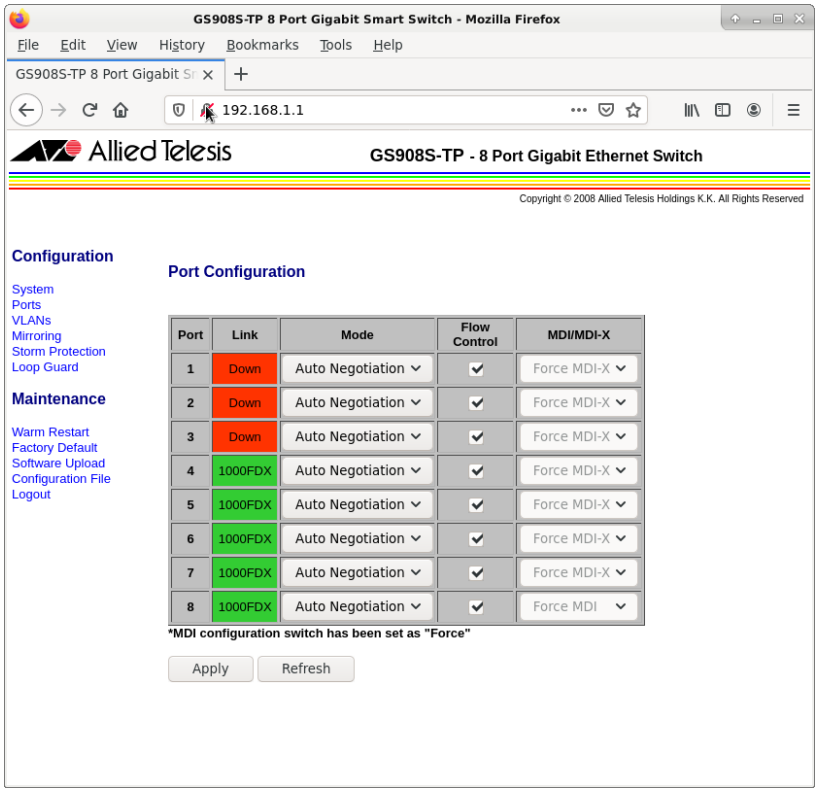
pause_time: 1 == 512 bit時間
1GbEなら1bit時間は1ns
pause_time 65535(最大値)が指定された場合は
512 * 65535 ns = 33_553_920 ns = 33.5ms

```
Pause frame
+---------+--------+---------+-------+--------------+---------+--------+
| dest(6) | src(6) | type(2) | op(2) | pausetime(2) | pad(42) | FCS(4) |
+---------+--------+---------+-------+--------------+---------+--------+
                              |<----- Ethernet Payload ------> |
dest:      01:80:C2:00:00:01
type:      0x8808
op code:   0x0001
pausetime: 0 - 65535
```

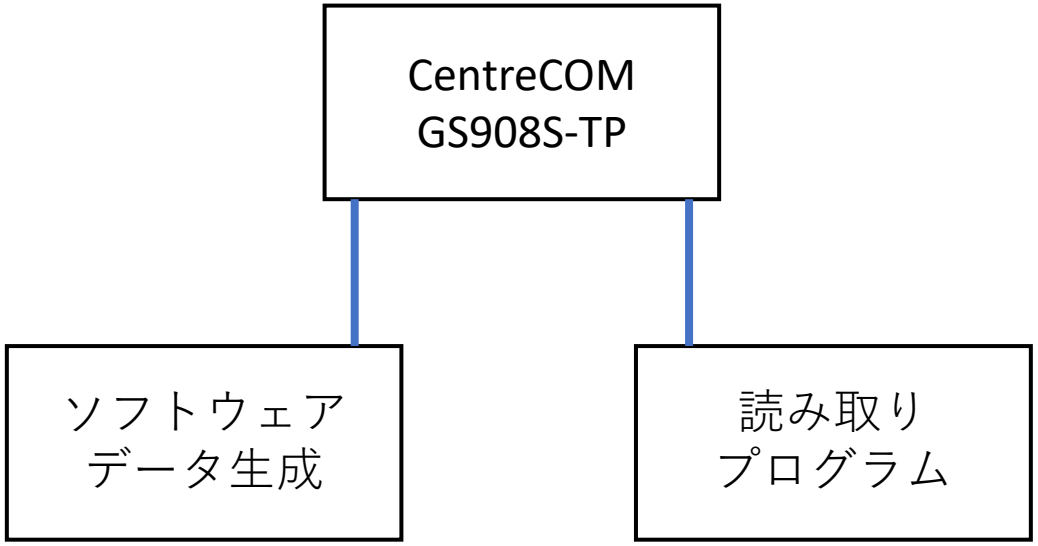
1GbE

- データサーバーはソフトウェア
- スイッチ: CentreCOM GS908S-TP
  https://www.allied-telesis.co.jp/products/list/switch/gs908stpv2/catalog.html
- flowcontrol on/off可能

パフォーマンス	スイッチング方式	ストア&フォワード方式	
	最大パケット転送能力（装置全体/64Byte）		11.9Mpps
	スイッチング遅延	1000M ⇔ 1000M	1.8 μ sec（64Byte）
		100M ⇔ 100M	2.6 μ sec（64Byte）
		10M ⇔ 10M	12.7 μ sec（64Byte）
	スイッチング・ファブリック		16Gbps
	メモリー容量	パケットバッファー	176KByte
		フラッシュメモリー	1024KByte
		メインメモリー	128KByte
	MACアドレス登録数	8K（最大）	
	MACアドレス保持時間	300～600秒	
	VLAN登録数	32個（VID=1～4,094）	



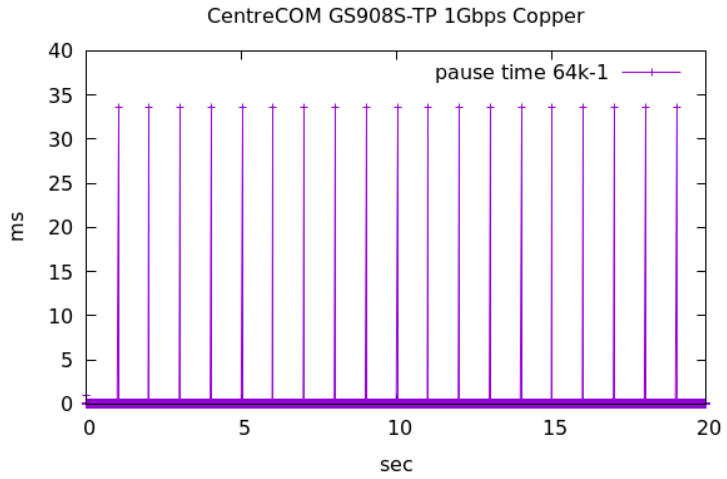
1GbE テスト構成

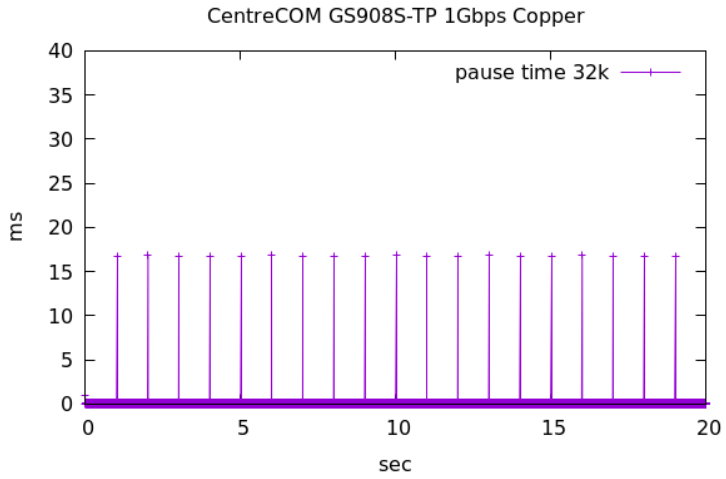


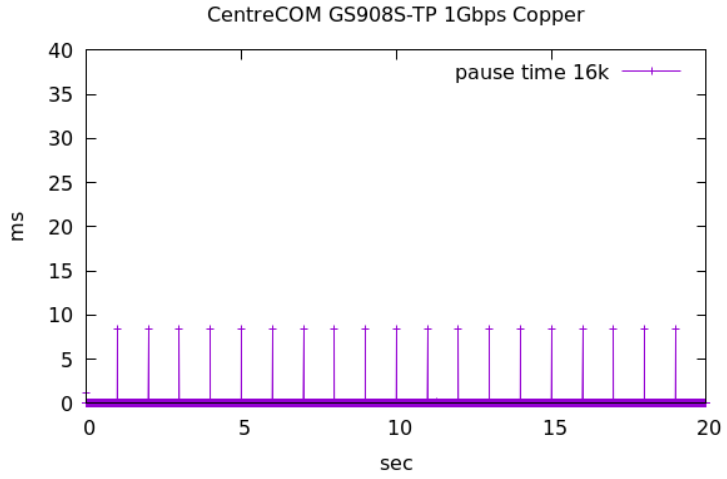
CentreCOM GS908S-TP
flow control 有効
flow control 無効
のそれぞれでテスト

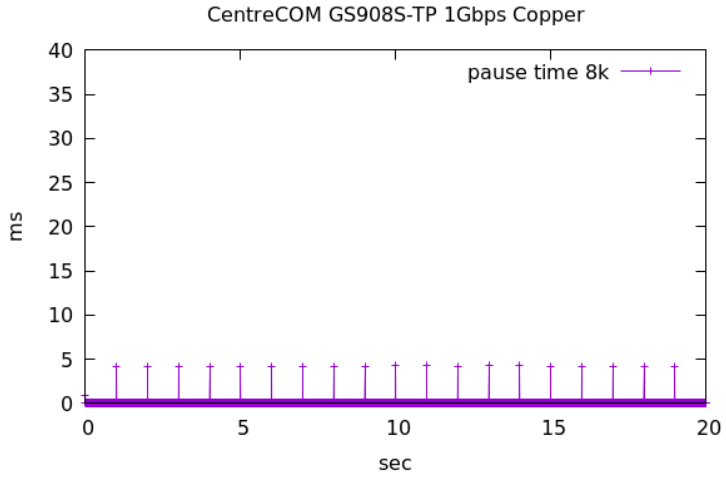
1GbEスイッチ フローコントールON

PC: CPU: Core2 Quad
NIC: Intel 82572EI Gigabit Ethernet Controller (Copper) (rev 06)



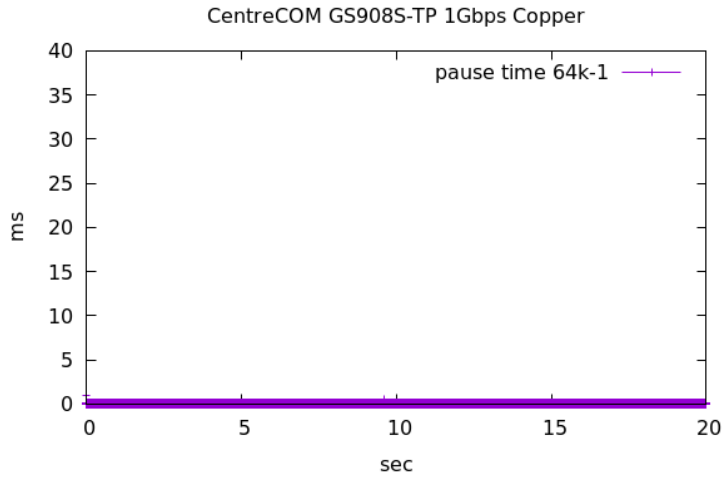


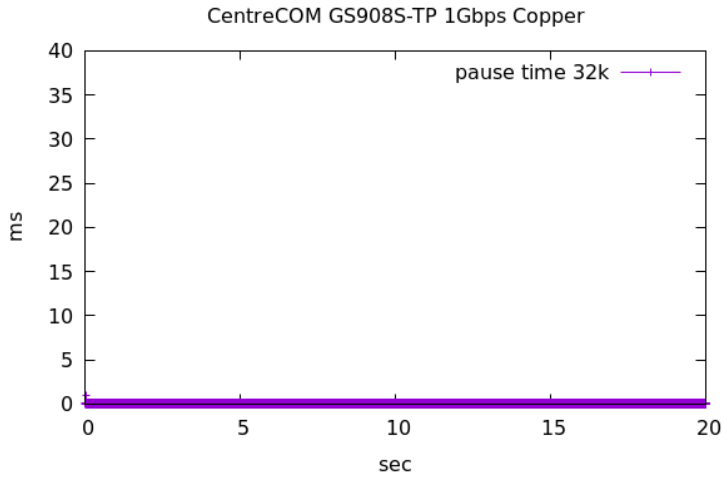


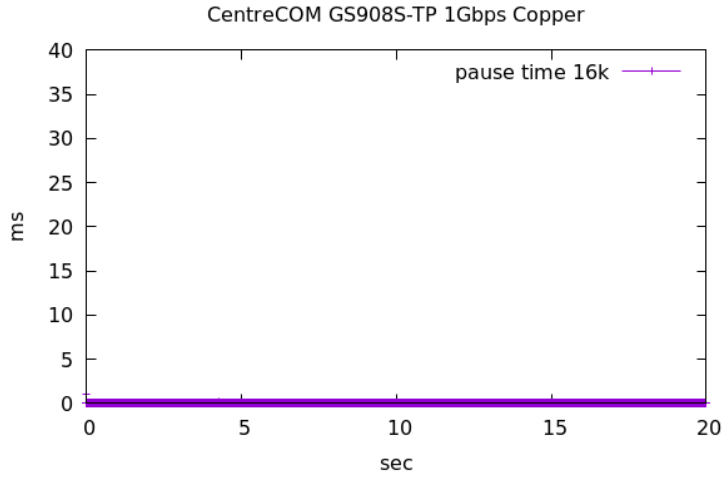


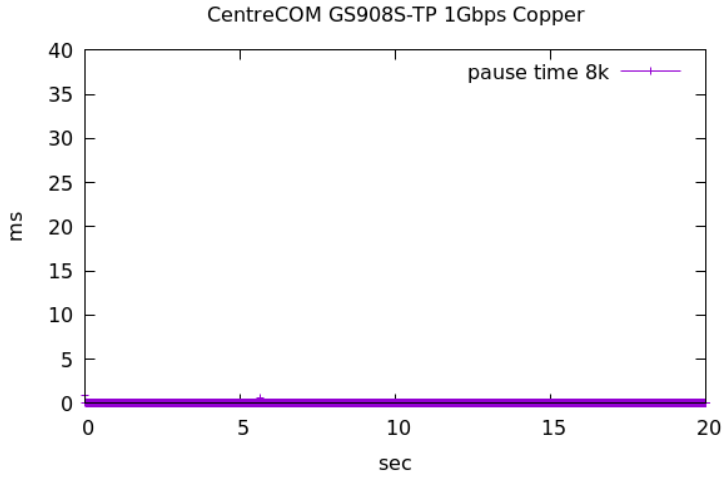
1GbEスイッチ フローコントールOFF

PC: CPU: Core2 Quad
NIC: Intel 82572EI Gigabit Ethernet Controller (Copper) (rev 06)





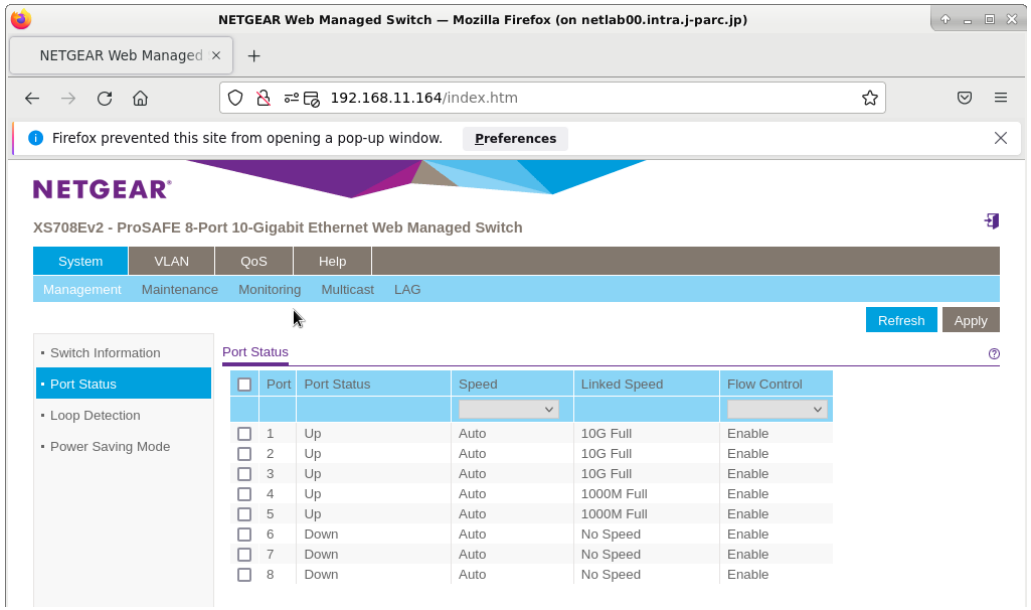




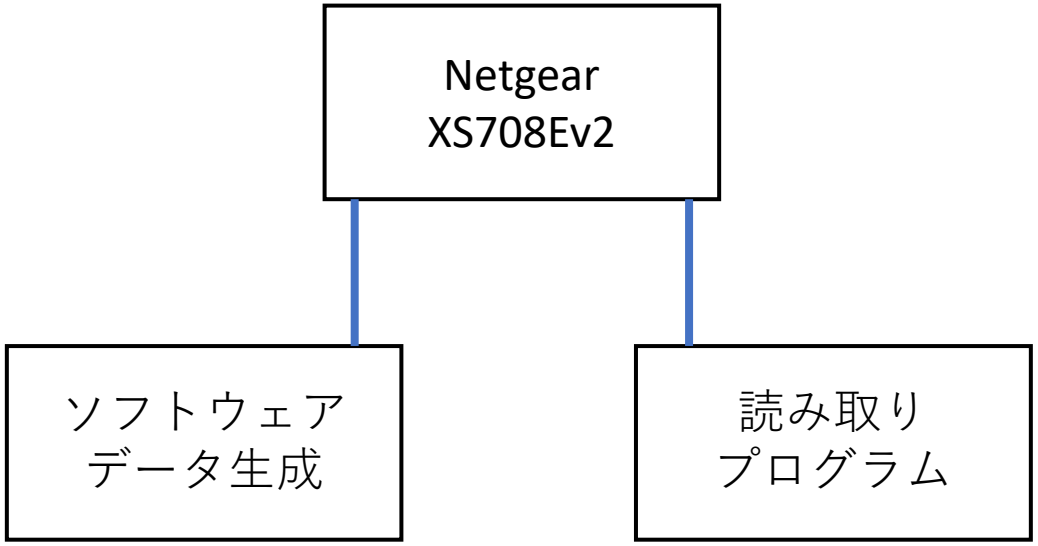
ethtool -A eth0 autoneg off rx on rx onでNIC側flow control有効化

10GbE 銅線

- データサーバーはソフトウェア
- スイッチ Netgear XS708Ev2 (低価格10GbEスイッチのはしり)
- RJ45 8ポート、SFPポート 1 (SFPはRJ45と排他)
- 管理用専用ポートはないが接続したPCからスイッチIPアドレスに接続して設定できる
  - スイッチIPアドレスはDHCP。DHCPがないときは192.168.0.239



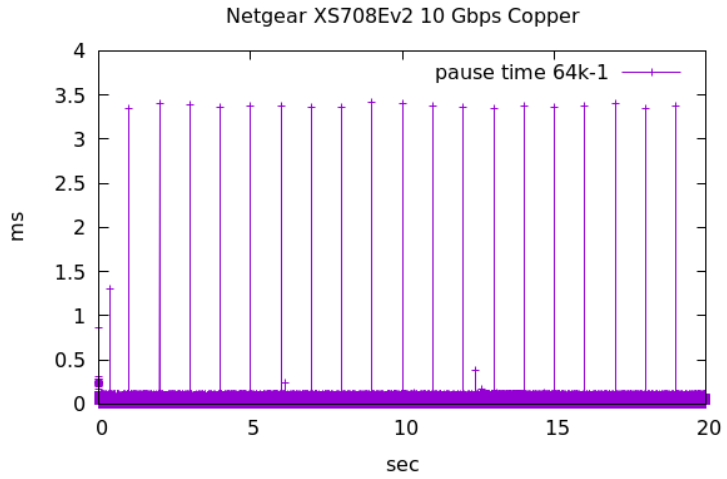
10 GbE copper テスト構成

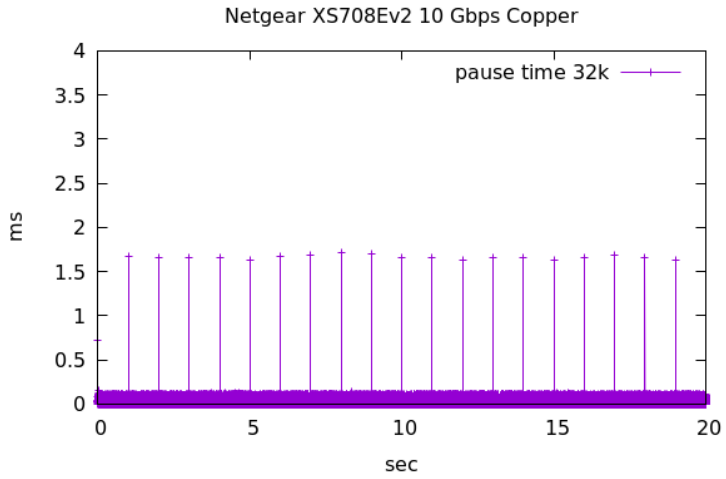


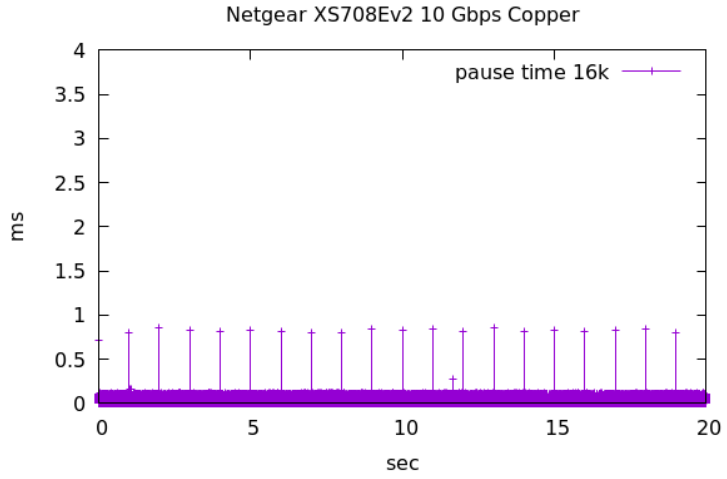
Netgear XS708Ev2
flow control 有効
flow control 無効
のそれぞれでテスト

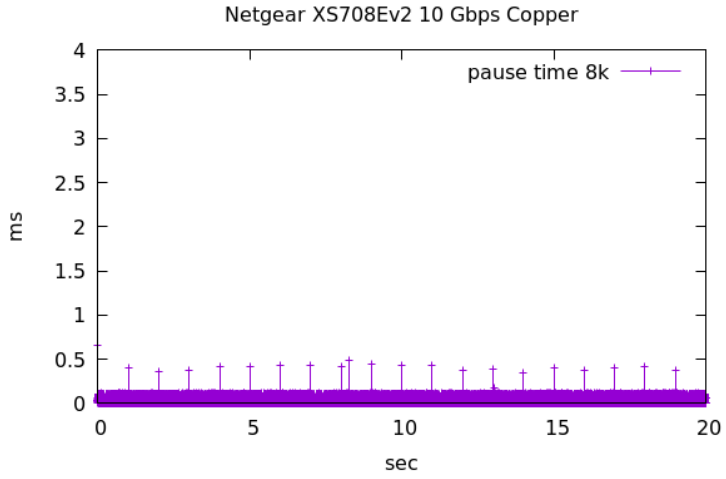
10 GbE copper スイッチ フローコントロールON

```
CPU: i7-6800K
NIC: Intel 10G X550T (rev 01)
```



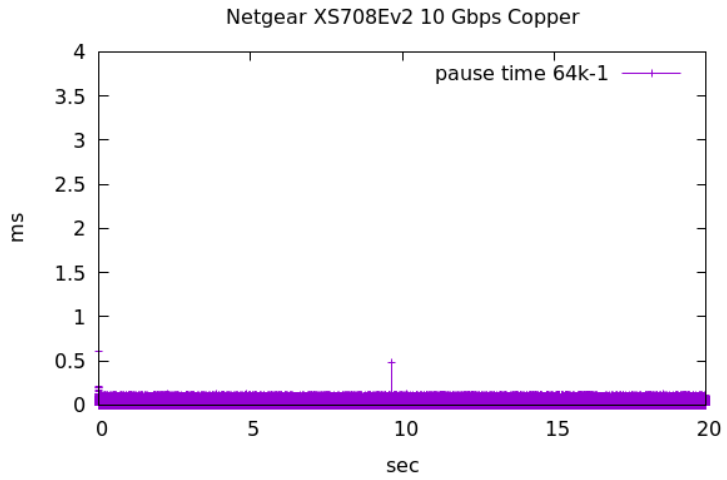


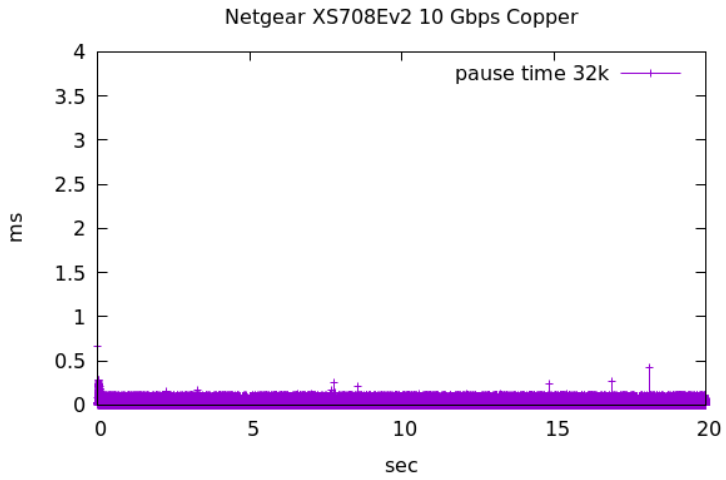


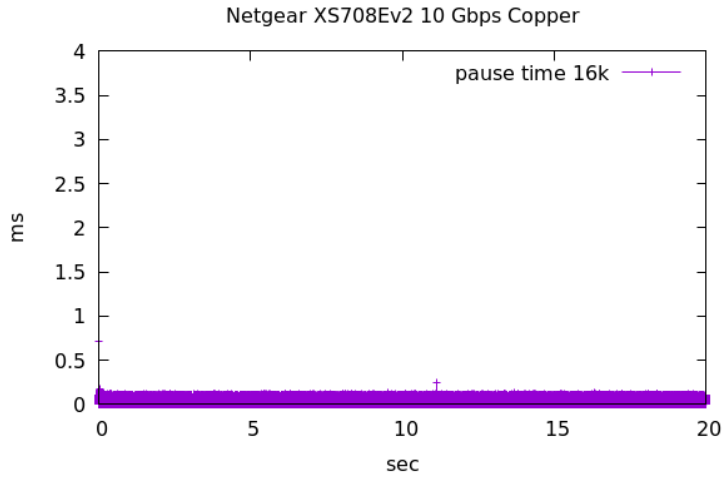


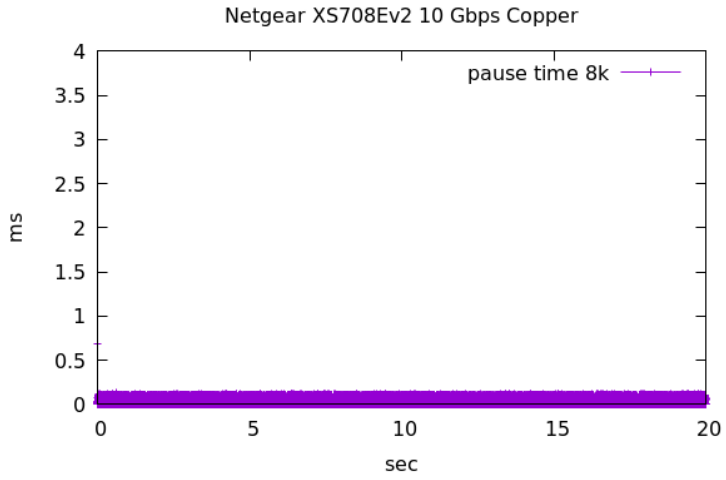
10 GbE copper スイッチ フローコントールOFF

```
CPU: i7-6800K
NIC: Intel 10G X550T (rev 01)
```







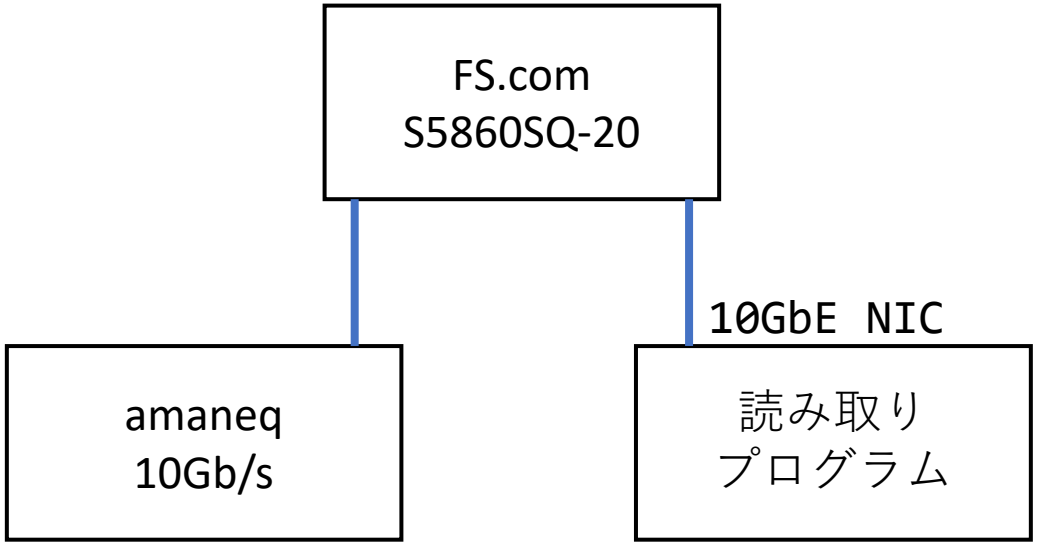


ethtool -A eth0 autoneg off rx on rx onでNIC側flow control有効化

10GbE optical

- データサーバーはamaneq 1台 (10Gb/sにセット)
- スイッチ FS.com S5860SQ-20
- コマンドラインでスイッチ側ポートflow control ON/OFF
- 読み出しPCはSFP 10GbE を利用
  - 40GbEを10GbEにしたわけではない
  - ドライバはixgbe

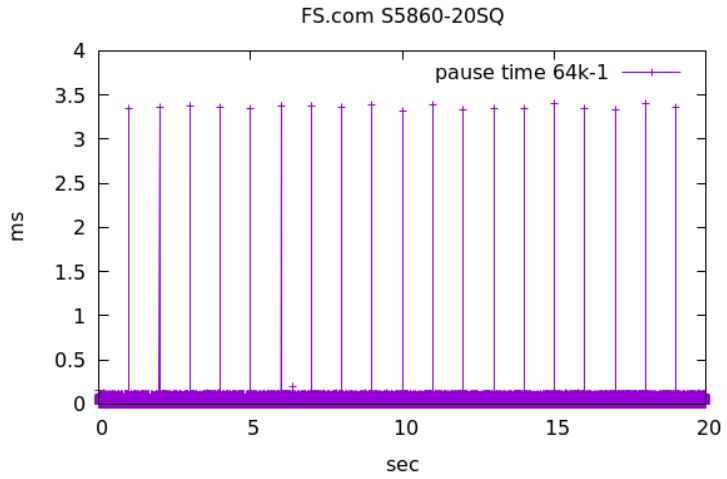
10 GbE optical テスト構成

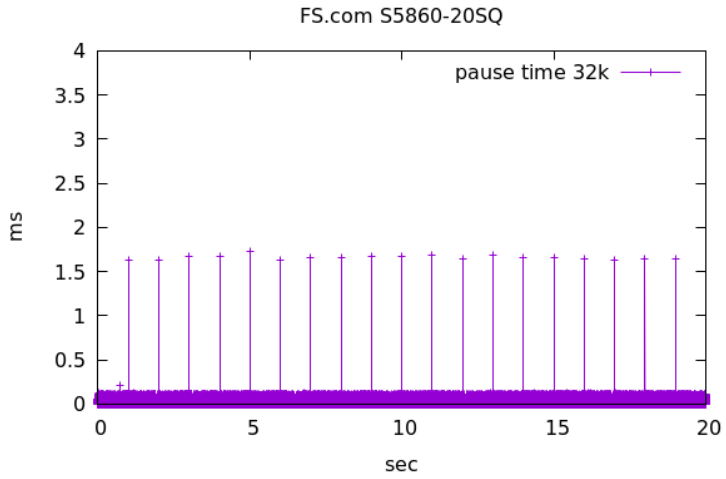


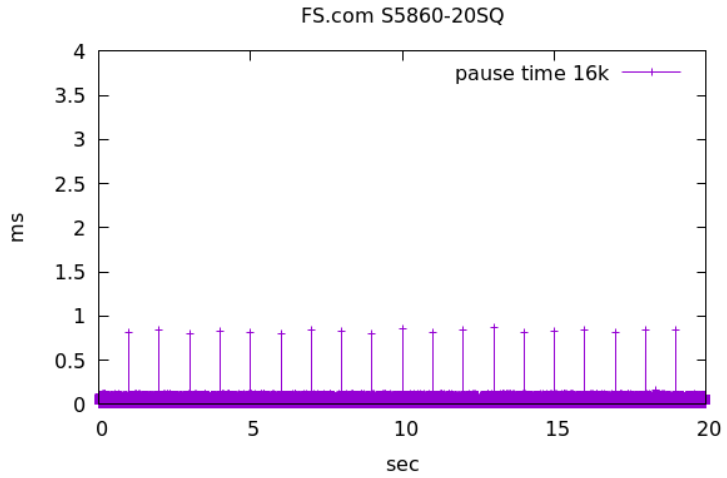
FS.com S5860SQ-20
flow control 有効
flow control 無効
のそれぞれでテスト

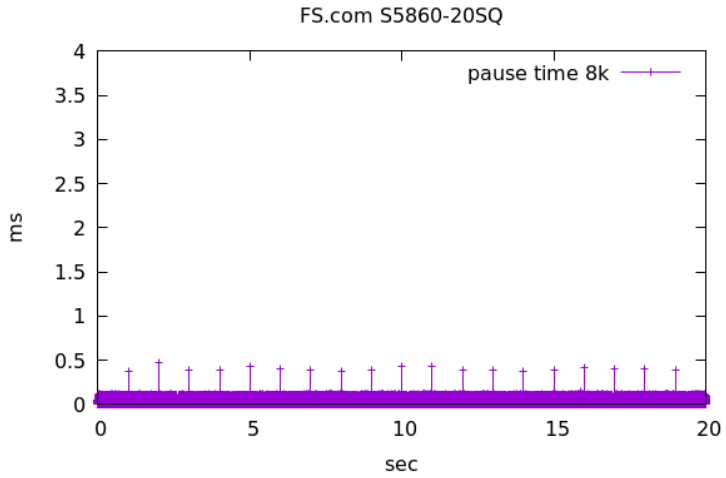
10 GbE optical スイッチ フローコントールON

```
CPU: i7-6800K
NIC: Intel 10G X550T (rev 01)
```



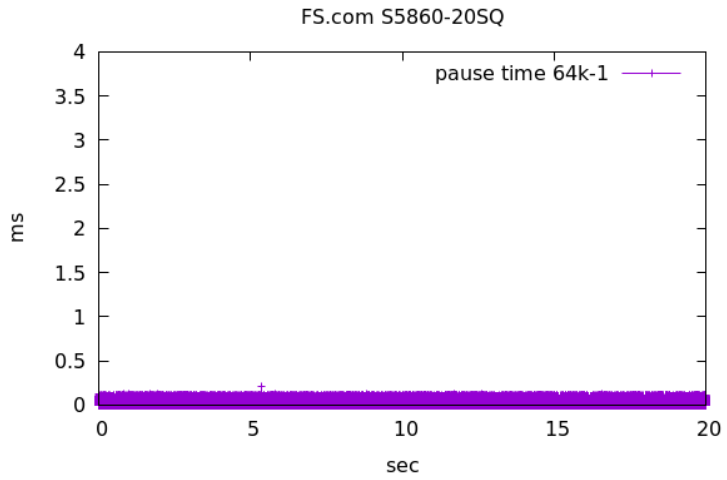


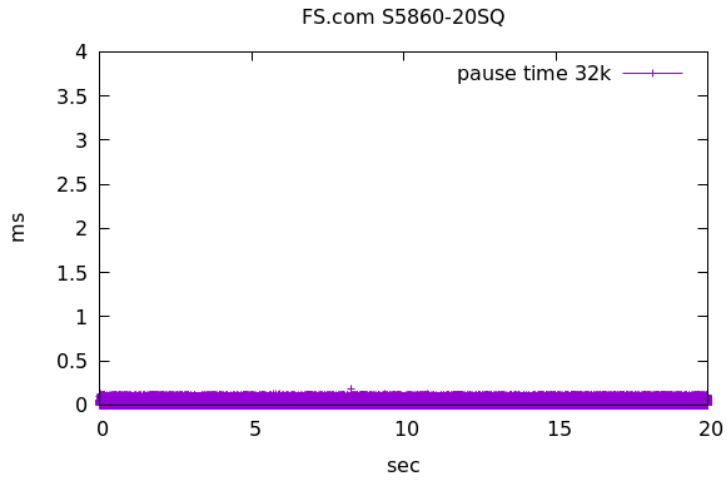


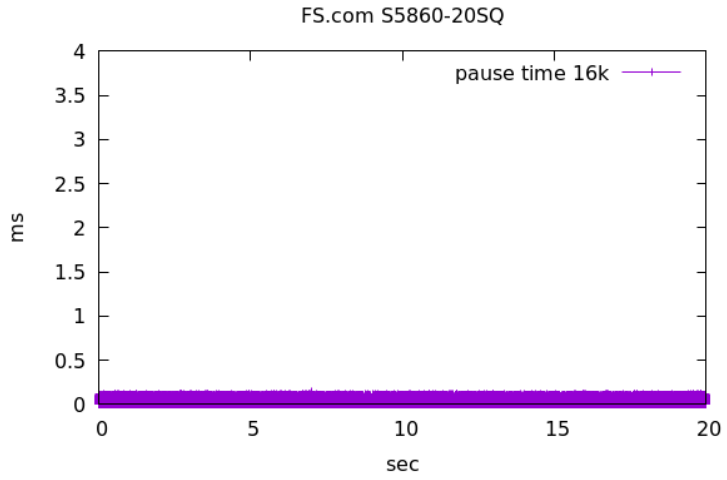


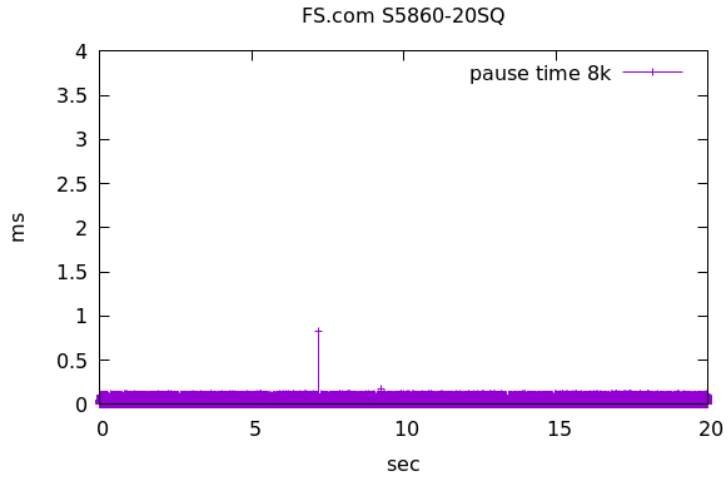
10 GbE optical スイッチ フローコントロールOFF

```
CPU: i7-6800K
NIC: Intel 10G X550T (rev 01)
```



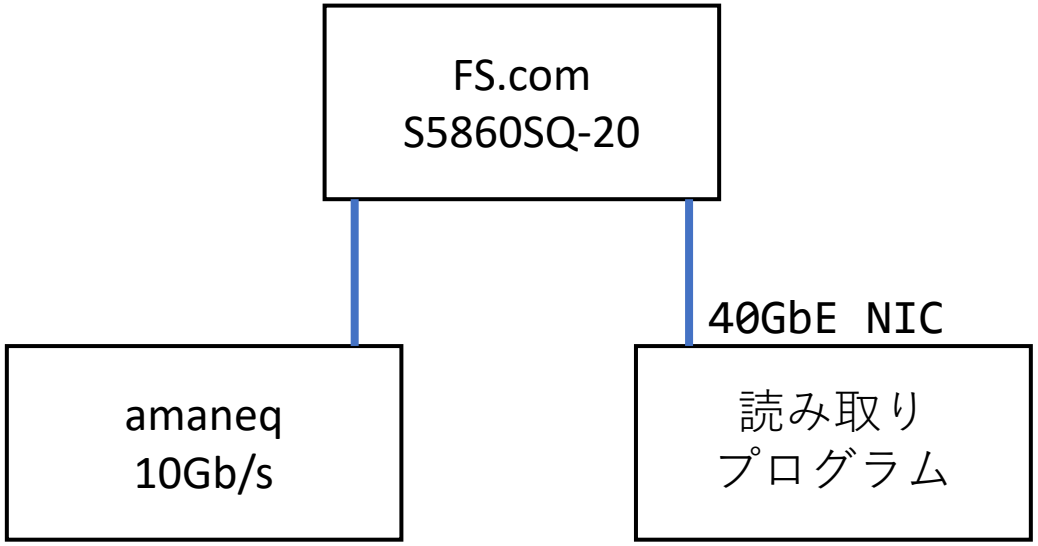






ethtool -A eth0 autoneg off rx on rx onでNIC側flow control有効化

40 GbE optical テスト構成

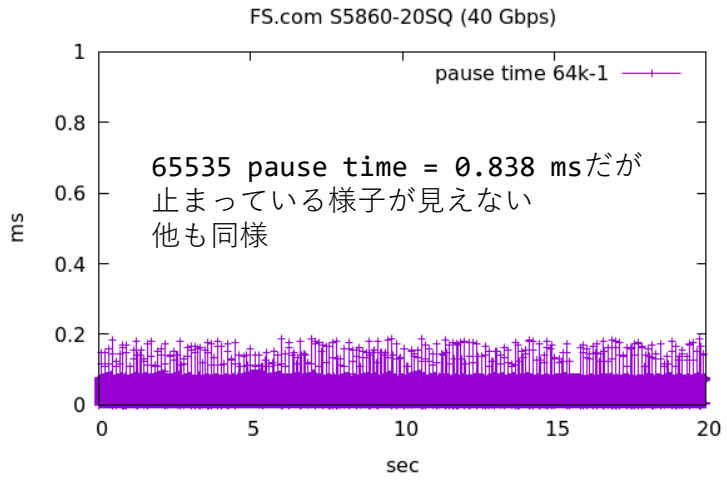


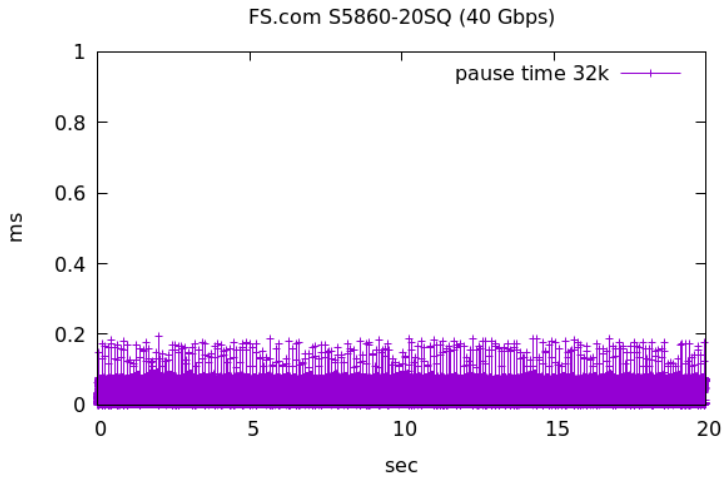
FS.com S5860SQ-20
flow control 有効
flow control 無効
のそれぞれでテスト

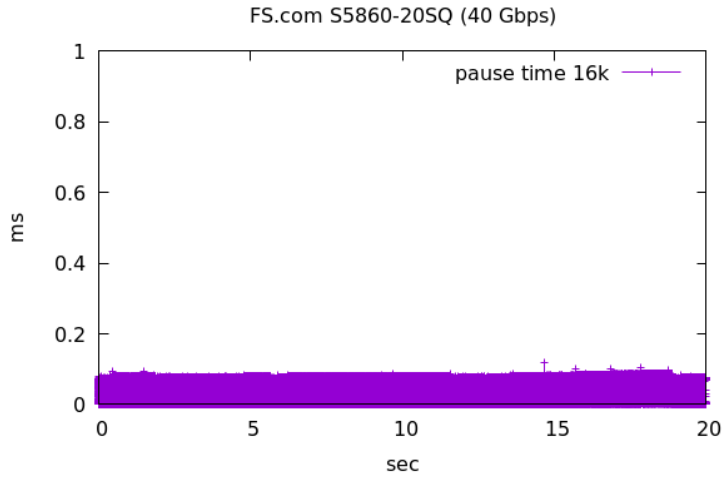
40GbE ドライバはi40e

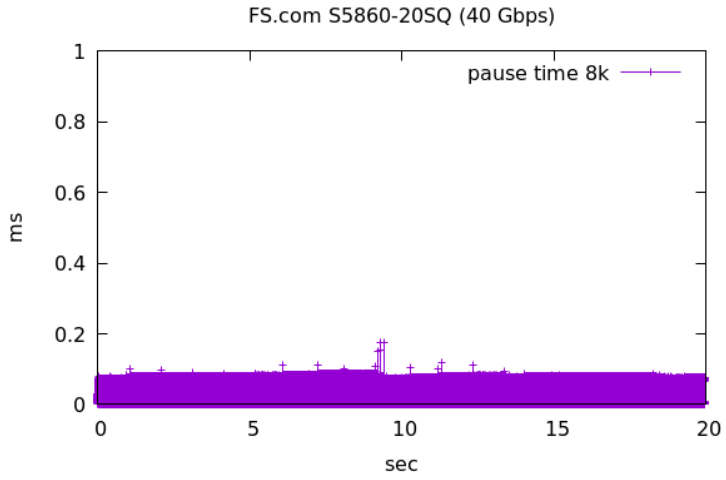
40 GbE optical スイッチ フローコントールON

```
CPU:  AMD EPYC 7313P
NIC: Intel XL710 for 40GbE QSFP+ (rev 02)
```









スイッチ、NICのLED

- FS.comスイッチLEDは10GbE, 40GbEともに点滅せず
- NIC: 10GbEはpause frameを送ったタイミングで点滅している
- NIC: 40GbEは点滅せず

ドライバのソースコード drivers/net/ethernet/intel/i40e/i40e_main.c

```
/**
 * i40e_rebuild - rebuild using a saved config
 * @pf: board private structure
 * @reinit: if the Main VSI needs to re-initialized.
 * @lock_acquired: indicates whether or not the lock has been acquired
 * before this function was called.
 **/
static void i40e_rebuild(struct i40e_pf *pf, bool reinit, bool lock_acquired)

:
:

/* Add a filter to drop all Flow control frames from any VSI from being
 * transmitted. By doing so we stop a malicious VF from sending out
 * PAUSE or PFC frames and potentially controlling traffic for other
 * PF/VF VSIs.
 * The FW can still send Flow control frames if enabled.
 */

i40e_add_filter_to_drop_tx_flow_control_frames(&pf->hw, pf->main_vsi_seid);
```

```
commit e7358f54a3954df16d4f87e3cad35063f1c17de5
Author: Anjali Singhai Jain <anjali.singhai@intel.com>
Date:   Thu Oct 1 14:37:34 2015 -0400

    i40e/i40evf: Add a workaround to drop all flow control frames

    This patch adds a workaround to drop any flow control frames from being
    transmitted from any VSI. FW can still send flow control frames if flow
    control is enabled.

    With this patch in place a malicious VF cannot send flow control or PFC
    packets out on the wire.

    Change-ID: I4303b24e98b93066d2767fec24dfe78be591c277
    Signed-off-by: Anjali Singhai Jain <anjali.singhai@intel.com>
    Tested-by: Andrew Bowers <andrewx.bowers@intel.com>
    Signed-off-by: Jeff Kirsher <jeffrey.t.kirsher@intel.com>
```

https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=e7358f54a3954df16d4f87e3cad35063f1c17de5
なにに対するworkaroundなのかコミットログからは不明。
commit SHA1 hash e7358f54a3954df16d4f87e3cad35063f1c17de5 で検索する。

CVE List ▾ CNAs ▾ WGs ▾ Board ▾
About ▾ News & Blog ▾

NVD
Go to for:
CVSS Scores
CPE Info

Search CVE List | **Downloads** | **Data Feeds** | **Update a CVE Record** | **Request CVE IDs**

TOTAL CVE Records: 171679

NOTICE: Transition to the all-new CVE website at WWW.CVE.ORG is underway and will last up to one year. (details)

NOTICE: Changes coming to CVE Record Format JSON and CVE List Content Downloads in 2022.

HOME > CVE > CVE-2015-1142857

Printer-Friendly View

CVE-ID

CVE-2015-1142857

Learn more at National Vulnerability Database (NVD)
• CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information

Description

On multiple SR-IOV cars it is possible for VF's assigned to guests to send ethernet flow control pause frames via the PF. This includes Linux kernel ixgbe driver before commit f079fa005aae08ee0e1bc32699874ff4f02e11c1, the Linux Kernel i40e/i40evf driver before e7358f54a3954df16d4f87e3cad35063f1c17de5 and the DPDK before commit 3f12b9f23b6499ff66ec8b0de941fb469297e5d0, additionally Multiple vendor NIC firmware is affected.

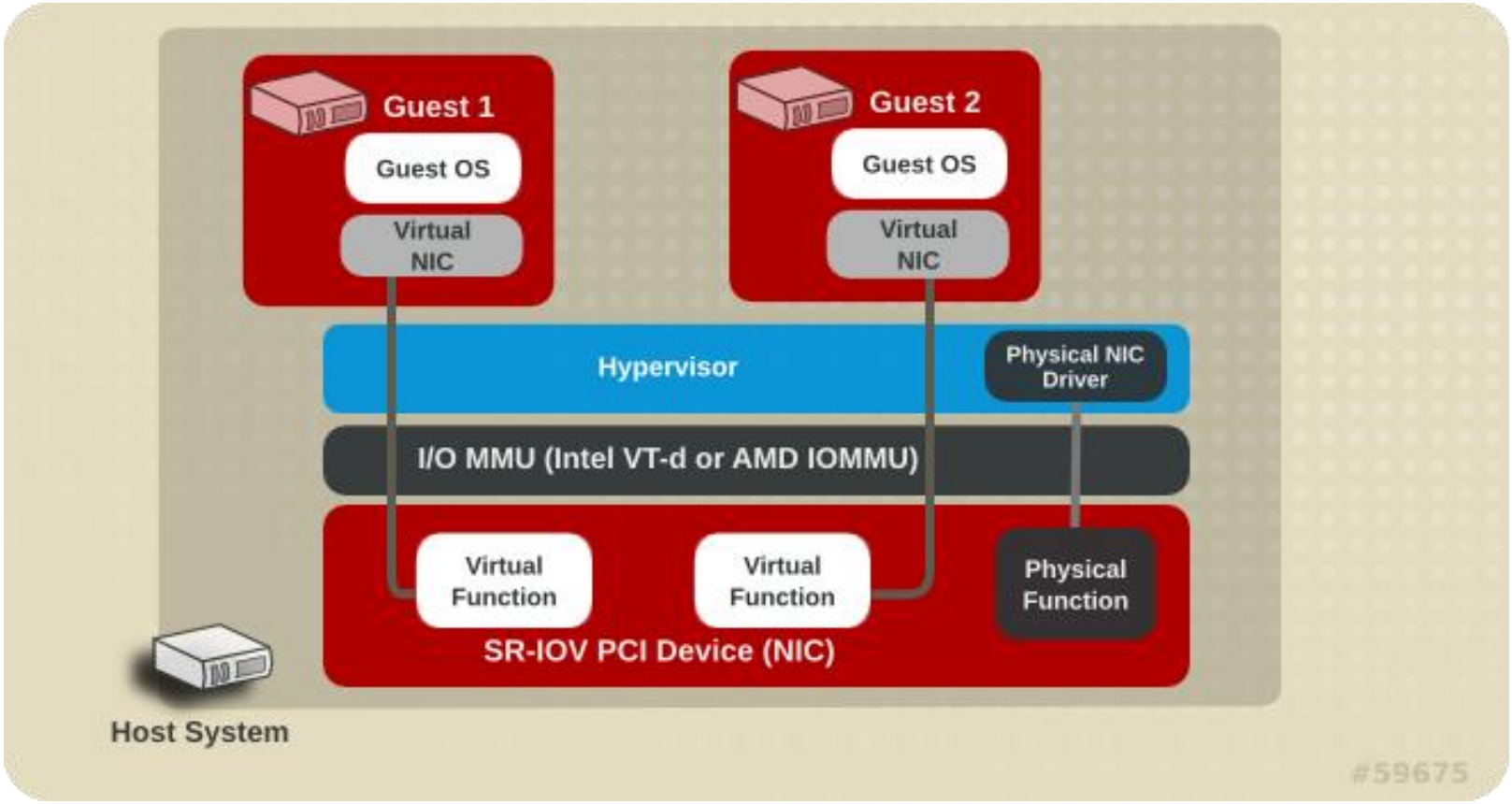
References

Note: References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.

- CONFIRM:https://security-center.intel.com/advisory.aspx?intelid=INTEL-SA-00046&languageid=en-fr
- MISC:https://www.usenix.org/system/files/conference/usenixsecurity15/sec15-paper-smolyar.pdf
- MLIST:[oss-security] 20151203 CVE request -- Ethernet flow control vulnerability in SRIOV devices
- URL:http://seclists.org/oss-sec/2015/q4/425

https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-1142857

SR-IOV
(Single ROOT IO Virtualization)



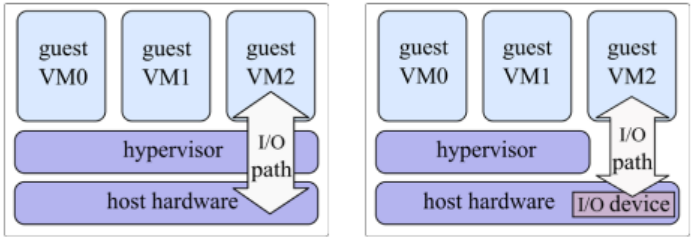
https://access.redhat.com/documentation/ja-jp/red_hat_enterprise_linux/6/html/virtualization_host_configuration_and_guest_installation_guide/chap-virtualization_host_configuration_and_guest_installation_guide-sr_iov

Securing Self-Virtualizing Ethernet Devices

Igor Smolyar Muli Ben-Yehuda Dan Tsafrir
Technion – Israel Institute of Technology
{igors,muli,dan}@cs.technion.ac.il

Abstract

Single root I/O virtualization (SRIOV) is a hardware/software interface that allows devices to "self virtualize" and thereby remove the host from the critical I/O path. SRIOV thus brings near bare-metal performance to untrusted guest virtual machines (VMs) in public clouds, enterprise data centers, and high-performance computing setups. We identify a design flaw in current Ethernet SRIOV NIC deployments that enables untrusted VMs to completely control the throughput and latency of other, unrelated VMs. The attack exploits Ethernet "pause" frames, which enable network flow control functionality. We experimentally launch the attack across several NIC models and find that it is effective and highly accurate, with substantial consequences if left unmitigated: (1) to be safe, NIC vendors will have to modify their NICs so as to filter pause frames originating from SRIOV instances; (2) in the meantime, administra-



(a) Traditional Virtualization (b) Direct I/O Device Assignment

Figure 1: Types of I/O Virtualization

driver is installed in the guest [20, 69]; (3) the host assigns a real device to the guest, which then controls the device directly [22, 52, 64, 74, 76]. When emulating a device or using a paravirtual driver, the hypervisor intercepts all interactions between the guest and the I/O device, as shown in Figure 1a, leading to increased overhead and significant performance penalty.

https://www.usenix.org/system/files/conference/usenixsecurity15/sec15-paper-smolyar.pdf

クラウド環境で隣の無関係なVMにpauseパケットを使って通信不可能攻撃をすることが可能である。

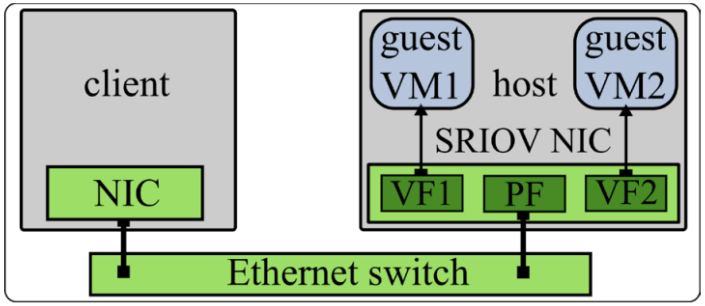


Figure 3: Setup scheme

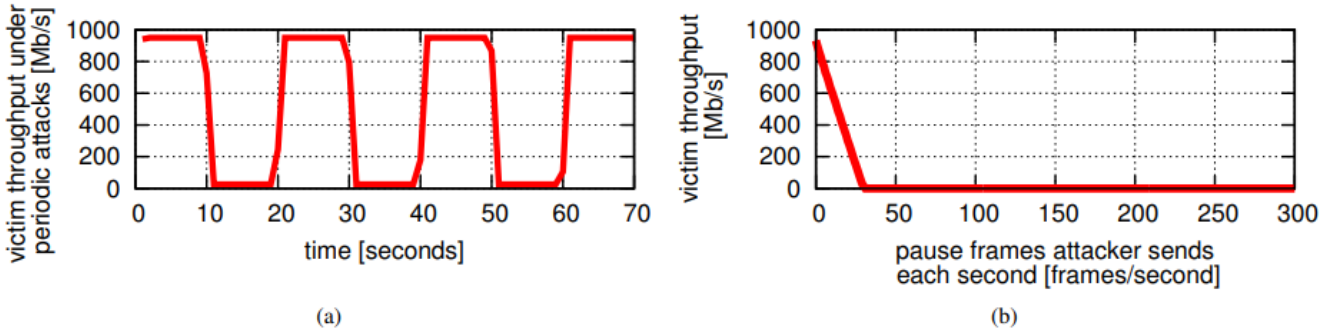


Figure 4: Pause frame attack: victim throughput in 1GbE environment

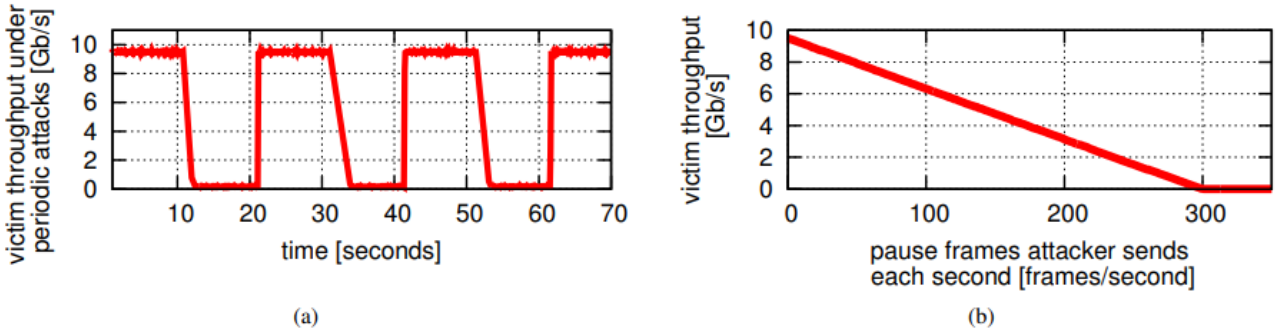


Figure 5: Pause frame attack: victim throughput in 10GbE environment

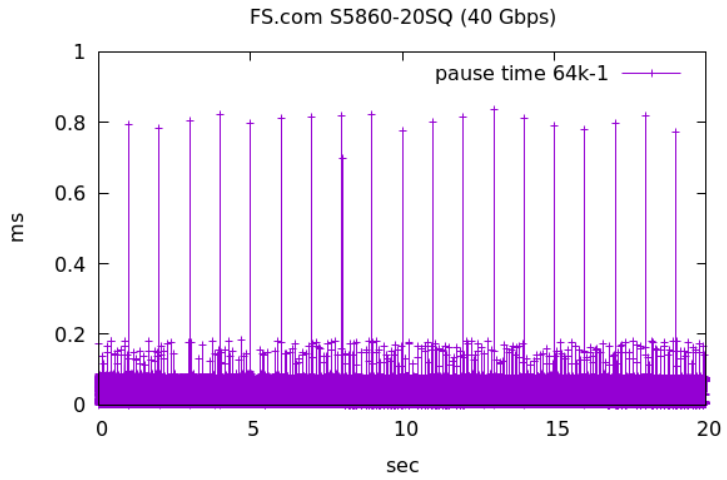
https://www.usenix.org/system/files/conference/usenixsecurity15/sec15-paper-smolyar.pdf

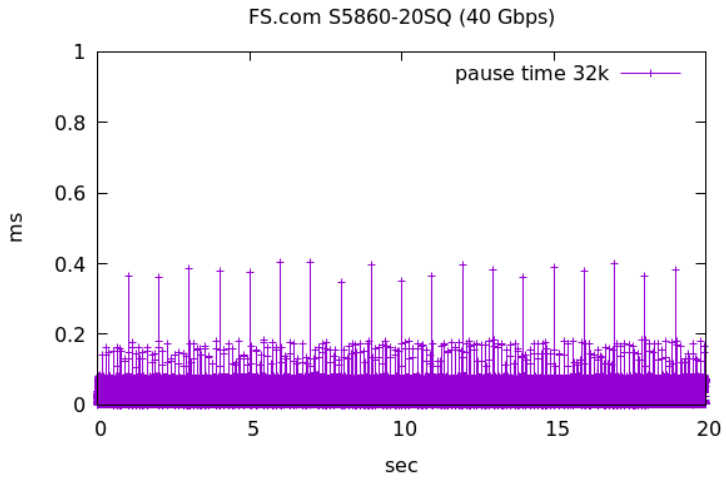
workaroundを取り除いたドライバの作成

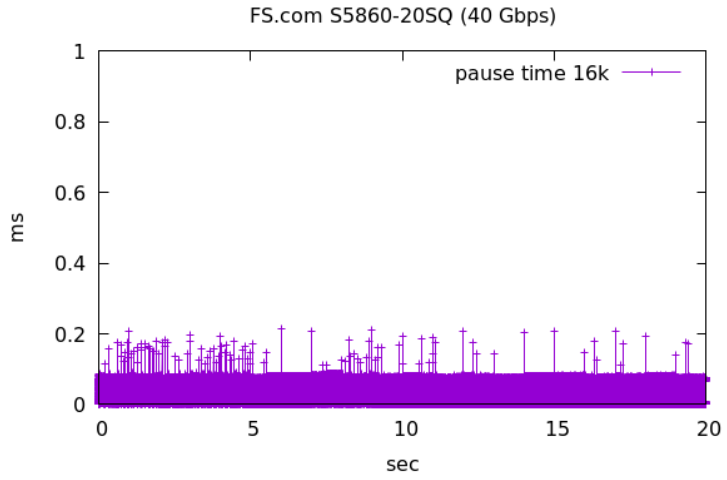
- rpmbuildディレクトリを作成
  - $HOME/.rpmmacrosに次のように書いて
    rpmdev-setuptreeコマンドを実行
    %_topdir /home/sendai/rpm
  - $HOME/rpm/{BUILD,BUILDROOT,RPMS,SOURCES, SPECS,SRPMS}ができる
- カーネルソースをダウンロード
  - cd ~/rpm/SRPMS; dnf download --source kernel
  - kernel-4.18.0-365.el8.src.rpmが取得できる
- rpm -ihv kernel-4.18.0-365.el8.src.rpm
- cd ../SPECS; rpmbuild -bp kernel.spec
- cd ~/rpm/BUILD/kernel-4.18.0-365.el8/linux-4.18.0-365.el8.x86_64
- make oldconfig; make prepare; make modules_prepare
- drivers/net/ethernet/intel/i40e/i40e_main.cを変更して先ほどのworkaroundを無効化
- make M=drivers/net/ethernet/intel/i40e
- mkdir -p /lib/modules/4.18.0-365.el8.x86_64/updates/drivers/net/ethernet/intel/i40e
- cp drivers/net/ethernet/intel/i40e/i40e.ko /lib/modules/4.18.0-365.el8.x86_64/updates/drivers/net/ethernet/intel/i40e
- depmod -a; dracut --force
- reboot

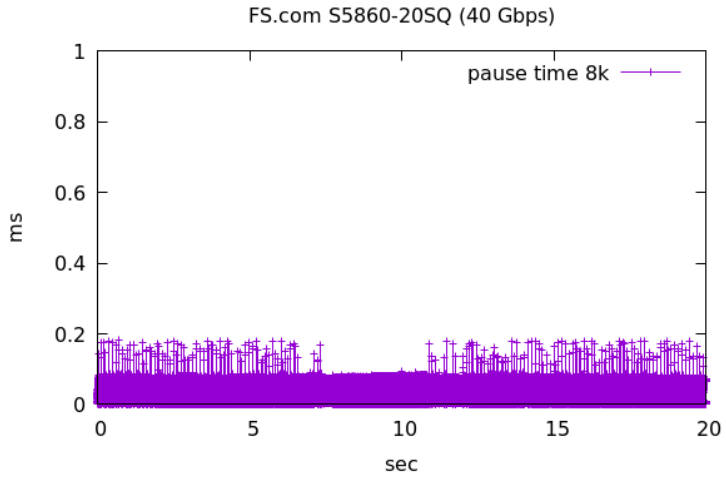
40 GbE optical スイッチ フローコントールON (ドライバ変更)

```
CPU:  AMD EPYC 7313P
NIC: Intel XL710 for 40GbE QSFP+ (rev 02)
```







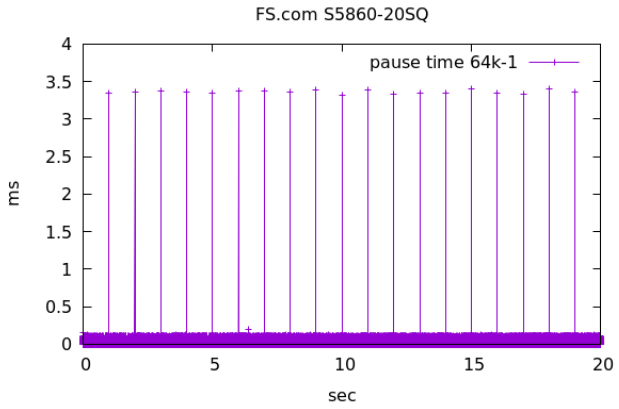


Workaroundをはずしたドライバをコンパイルして試してみた。

Intel 10GbE ドライバにもユーザープロセスがpauseを送れないようにするコードがある

```
drivers/net/ethernet/intel/ixgbe/ixgbe_sriov.c

int ixgbe_ndo_set_vf_spoofchk(struct net_device *netdev, int vf, bool setting)
{
    (略)
    /* Ensure LLDP and FC is set for Ethertype Antispoofing if we will be
     * calling set_ethertype_anti_spoofing for each VF in loop below
     */
     (略)
        IXGBE_WRITE_REG(hw, IXGBE_ETQF(IXGBE_ETQF_FILTER_FC),
                (IXGBE_ETQF_FILTER_EN |
                 IXGBE_ETQF_TX_ANTISPOOF |
                 ETH_P_PAUSE));

        hw->mac.ops.set_ethertype_anti_spoofing(hw, setting, vf);
    }

    return 0;
}
```

```
CPU: i7-6800K
NIC: Intel 10G X550T (rev 01)
```




```
static void ixgbe_configure_virtualization(struct ixgbe_adapter *adapter)
{
    (略) (pr_info()を入れて再コンパイル。出力はdmesgで見える)
    pr_info("entering ixgbe_configure_virtualization()\n");
    (略)
    for (i = 0; i < adapter->num_vfs; i++) {
        pr_info("entering ixgbe_ndo_set_vf_spoofchk() loop\n");

        /* configure spoof checking */
        ixgbe_ndo_set_vf_spoofchk(adapter->netdev, i, adapter->vfinfo[i].spoofchk_enabled);
       (略)
    }
}
```

```
[   42.483579] ixgbe 0000:02:00.0: registered PHC device on ens6f0
[   42.659445] ixgbe: entering ixgbe_configure_virtualization()
[   42.699118] ixgbe 0000:02:00.0 ens6f0: detected SFP+: 5
```

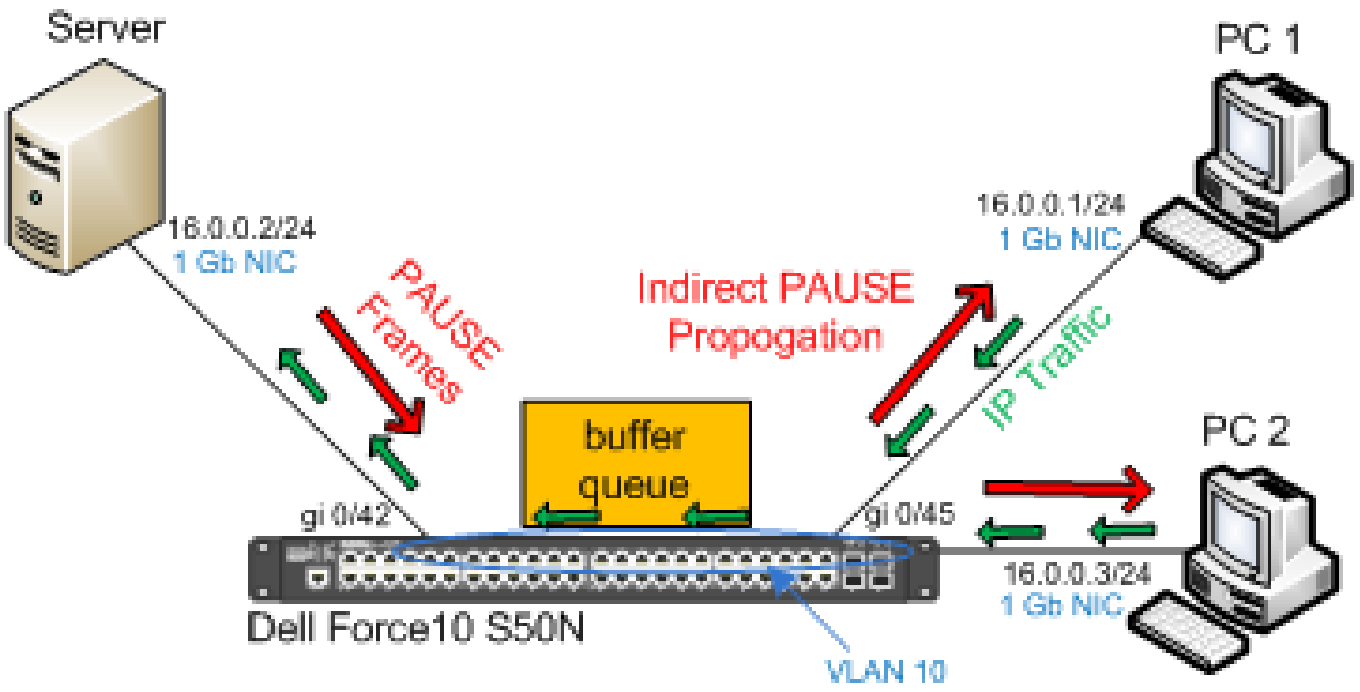
SR-IOVが無効になっていたおかげでpause無効化コードを通過していなかった。。
40GbEのドライバi40eでは問答無用でpause frameを送れないようにセットされている（ような気がする）

https://humairahmed.com/blog/?p=5316

PAUSE Frame

Preamble (7 bytes)
Start Frame Delimiter (1 byte)
Destination Address (01:80:C2:00:00:01)
Station MAC Address (6 bytes)
Ethertype = 0x8808 (2 bytes)
Opcode = 0x0001 (2 bytes)
Pause_Time (2 bytes)
Pad (42 bytes)
CRC (4 bytes)

Pause_Time: 単位は512ビット時間 (1GbEなら512ns。10GbEなら51.2ns)



PAUSE frames do not propagate directly from link to link. The switch starts to build a queue and once that queue reaches a certain threshold, the switch is forced to send a PAUSE frame to the PC to avoid dropping frames. By this mechanism, PAUSE frames are propagated indirectly.