

# Wireshark Developer and User Conference

## BI-8 Wireshark Software Case Studies

4:45p – 6:00p Mon June 25 2012

**Megumi Takeshita**

Founder | ikeriri network service co.,ltd.

**SHARKFEST '12**

UC Berkeley

June 24-27, 2012

# About Megumi Takeshita 竹下恵



- Founder, ikeriri network service co.,ltd ← Enterprize solution, nortel networks ← IS Bay Network
- 10+ books about packet capturing, analysis, inspection, and consulting
- Reseller of Riverbed Technology ( former CACE technologies ) and Metageek, and some in Japan
- Packet capturing Otaku ( geek ) start at junior school SharpX1(IPL)

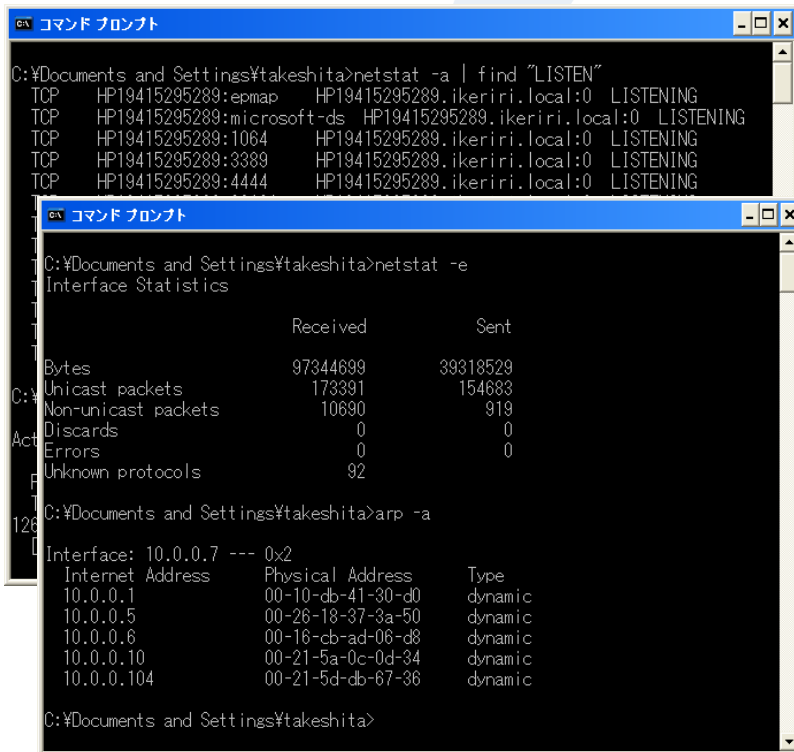


from Ethereal, 1<sup>st</sup> Sharkfest !



# before capturing

- Clear DNS C:¥>ipconfig /cleardns
- Stop firewalls, anti-spywares and others
- Stop service like UPnP(SSID), VPN and many



```
コマンド プロンプト
C:\Documents and Settings\takeshita>netstat -a | find "LISTEN"
TCP    HP19415295289:epmap    HP19415295289.ikeriri.local:0 LISTENING
TCP    HP19415295289:microsoft-ds HP19415295289.ikeriri.local:0 LISTENING
TCP    HP19415295289:1064    HP19415295289.ikeriri.local:0 LISTENING
TCP    HP19415295289:3389    HP19415295289.ikeriri.local:0 LISTENING
TCP    HP19415295289:4444    HP19415295289.ikeriri.local:0 LISTENING

コマンド プロンプト
C:\Documents and Settings\takeshita>netstat -e
Interface Statistics

              Received              Sent
Bytes          97344699          39318529
Unicast packets 173391            154683
Non-unicast packets 10690            919
Discards        0                  0
Errors          0                  0
Unknown protocols 92

C:\Documents and Settings\takeshita>arp -a
Interface: 10.0.0.7 --- 0x2
Internet Address Physical Address Type
10.0.0.1         00-10-db-41-30-d0 dynamic
10.0.0.5         00-26-18-37-3a-50 dynamic
10.0.0.8         00-16-cb-ad-08-d8 dynamic
10.0.0.10        00-21-5a-0c-0d-34 dynamic
10.0.0.104       00-21-5d-db-67-36 dynamic

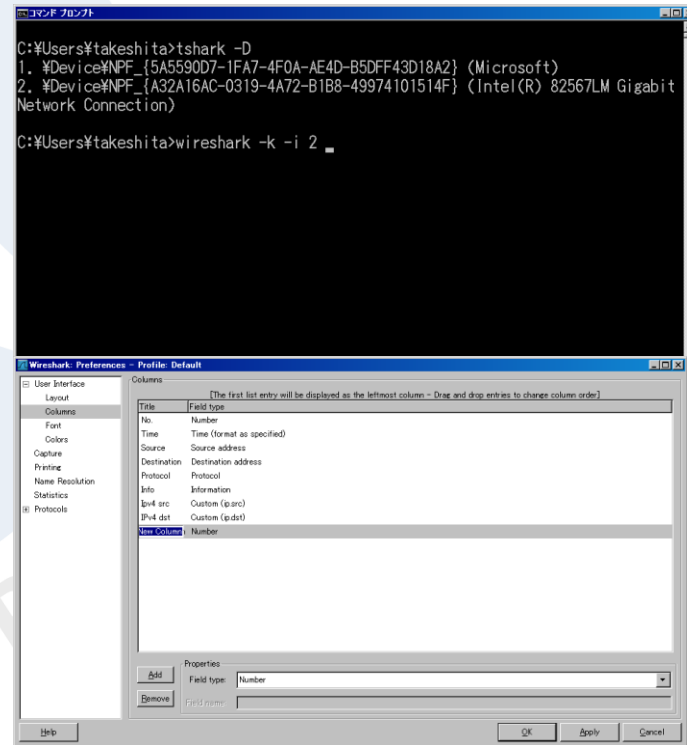
C:\Documents and Settings\takeshita>
```

- C:¥>netstat -a | find "LISTEN" ; netstat -ab
- Check NIC error, discards  
C:¥>netstat -e
- for /l %i in (1,1,10) do xxx  
is also useful



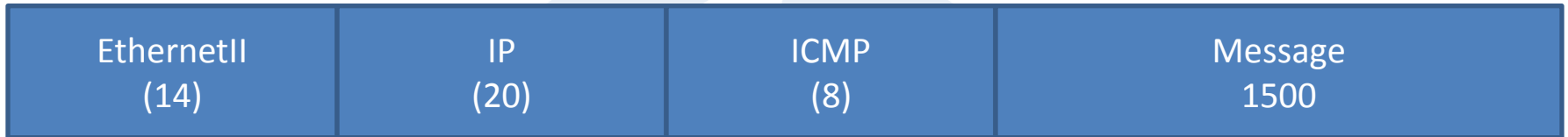
# Setting Wireshark

- Adding Wireshark program path into system variable ( set Path=%Path%;C:¥Pro... )
- Check interface index number ( thark -D )
- Add columns according to the field catching up
- To see latency, add fields tcp.time\_delta
- Set Time display format previous displayed packet

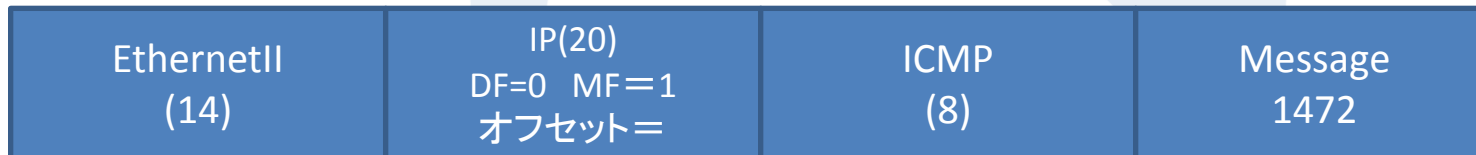


# Fragment

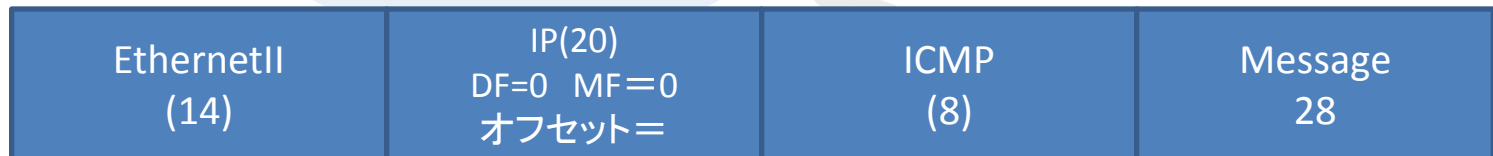
- Original frame



- Fragment 1/2 in Ethernet MTU



- Fragment 2/2 in Ethernet MTU



# Testing packet size

**ICMP** • ping host -l message size (MTU-28) -f

EthernetII (14)	IP (20)	ICMP (8)	message MTU=1500->1472
--------------------	------------	-------------	---------------------------

## TCP

EthernetII (14)	IP (20)	TCP (20)	Segment size MSS=1460
--------------------	------------	-------------	--------------------------

## UDP

EthernetII (14)	IP (20)	UDP (8)	Datagram size MTU=1500->1472
--------------------	------------	------------	---------------------------------

- NTT East MTU 1454Bytes (MSS 1414)
- NTT West FTTH MTU 1438Bytes (MSS 1398)
- GRE + IPsec (transport mode)1440
- GRE + IPsec (tunneling mode)1420



# Check negotiation of TCP

See first 2 packet of 3way handshake

(初期ウィンドウサイズはOS等で指定)

- Window Size, SACK, MSS, Window Scaling
- Some router may rewrite this section via NAT
- Follow TCP stream and Use coloring and Ctrl+Space

The screenshot shows the Wireshark interface with a packet capture of a TCP 3-way handshake and an HTTP POST request. The packet list pane shows the following packets:

No.	Time	Source	Destination	Protocol	Info
10	0.000000	172.16.6.13	202.51.3	TCP	rsock > http [SYN, Seq=0 win=64240 Len=0 MSS=1460 WS=0
20	0.002444	202.51.3	172.16.6.13	TCP	http > rsock [SYN, ACK] Seq=0 Ack=1 win=5840 Len=0 MSS=
30	0.000033	172.16.6.13	202.51.3	TCP	rsock > http [ACK] Seq=1 Ack=1 win=64240 Len=0
40	0.037035	172.16.6.13	202.51.3	HTTP	POST /rms/mall/basket/vc HTTP/1.1 (application/x-www-
50	0.002904	202.51.3	172.16.6.13	HTTP	http > rsock [ACK] Seq=1 Ack=1004 win=7021 Len=0

The packet details pane for the selected packet (No. 40) shows the following information:

- Ethernet II, Src: PNIC8A:9f:1e (08:00:0c:28:9f:1e), Dst: PNIC8A:9f:1e (08:00:0c:28:9f:1e)
- Internet Protocol Version 4, Src: 172.16.6.13 (172.16.6.13), Dst: 202.51.3.89
- Transmission Control Protocol, Src Port: 4944, Dst Port: 80

The packet bytes pane shows the raw data of the packet:

```
0000 00 14 22 1c a2 99 00 0b 97 94 9f 1e 08 00 45 00 ..
0010 00 34 ed 36 40 00 80 06 5e 01 ac 10 06 0d ca 48 .4
0020 33 26 0b 49 00 50 05 46 9b 06 00 00 00 80 02 3&
0030 fa f0 18 b6 00 00 02 04 05 b4 01 03 03 00 01 01 ...
0040 04 02 ..
```

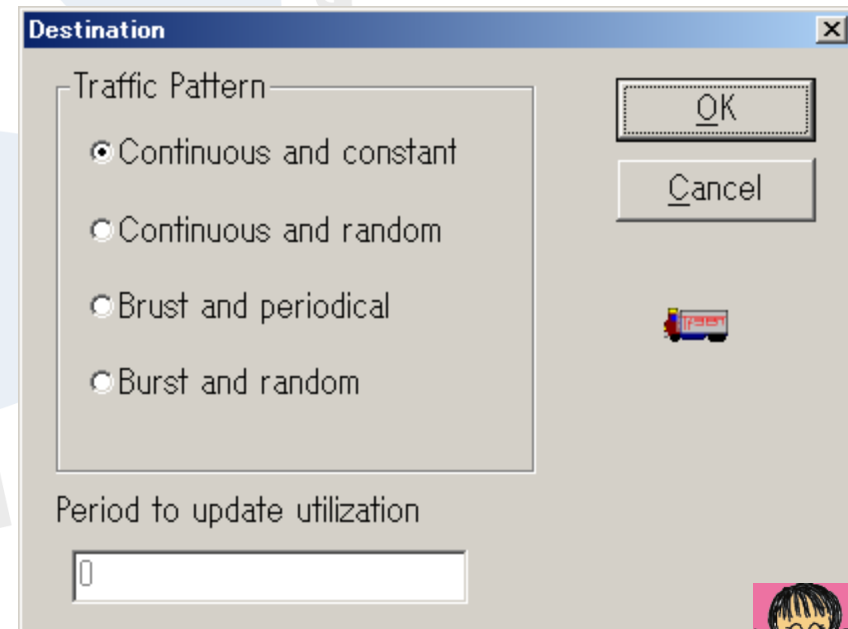
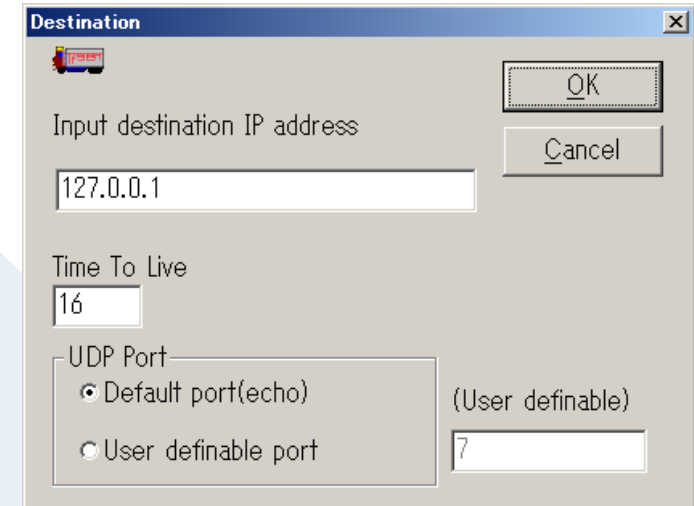
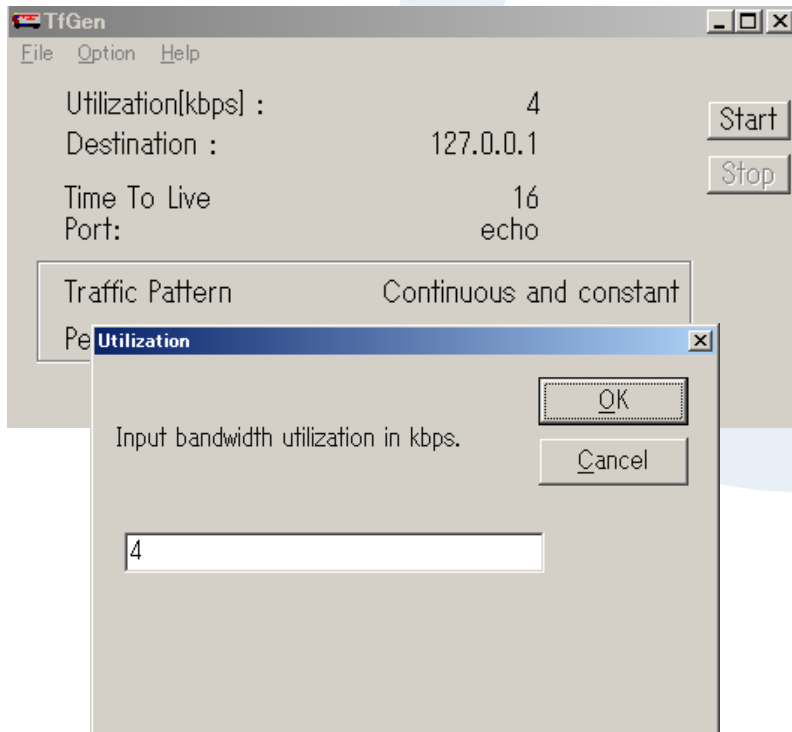
The packet bytes pane also shows the raw data of the packet, including the HTTP POST request body:

```
POST /rms/mall/basket/vc HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-
shockwave-flash, application/vnd.ms-excel, application/vnd.ms-powerpoint,
application/msword, */*
Referer: http://item.rakuten.co.jp/nigari612/andino700creamer/
Accept-Language: ja
Content-Type: application/x-www-form-urlencoded
UA-CPU: x86
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR
1.1.4322; .NET CLR 2.0.50727)
Host: order.step.rakuten.co.jp
Content-Length: 78
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: st=4cf018409fd34c30021f0872c10176; c=ccstruc; c=cc-rakutencojwain%3d%3d
```



# Using iperf and tfgen (made in Japan)

- Throughput ->iperf  
assuming packet size
- Traffic ->tfgen  
influences about UDP





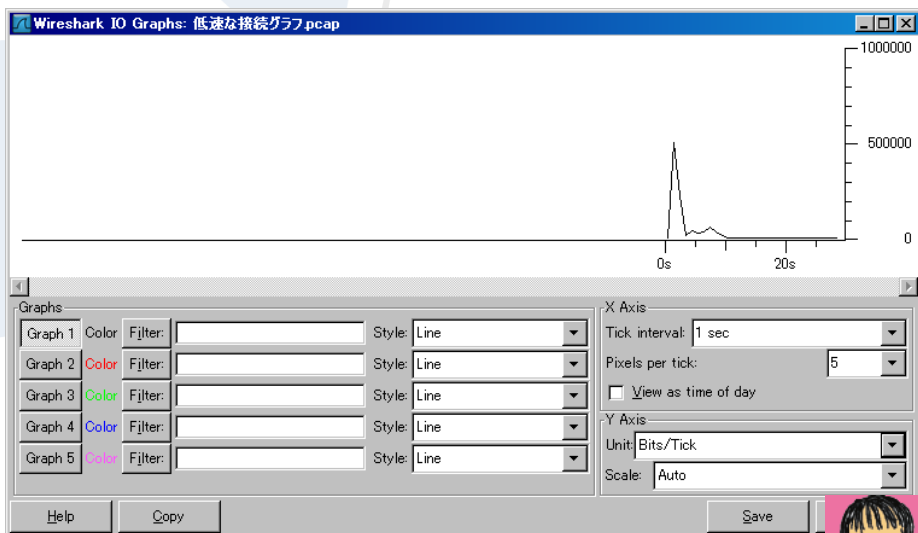
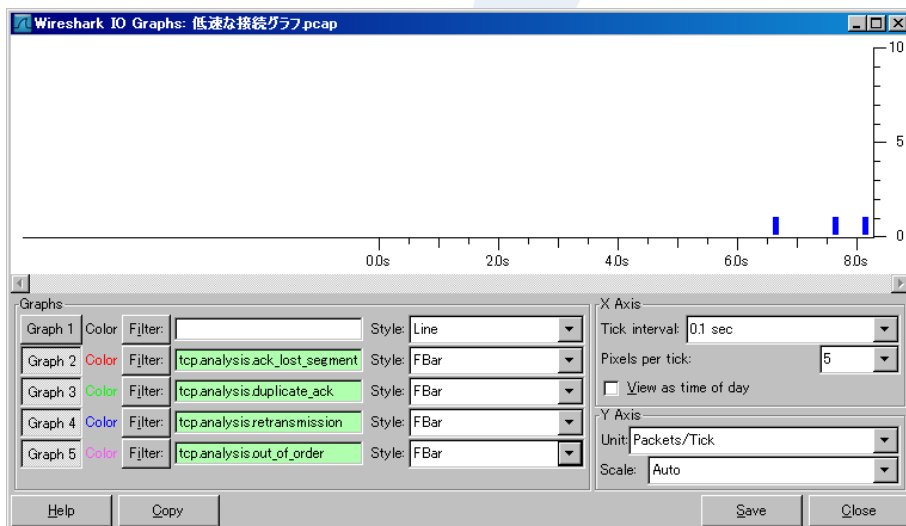
# Overview of troubleshoot

- Amount, place, time is important !!  
ask for the person and know the issue.
- If we know the error obviously,  
see difference from OK and NG packet  
to see packet in micro range ( field )
- No idea of trouble  
capture packet at more than 2 location  
to see packet in macro range ( statistics )
- Expert Info say many things automatically
- Think of packet lost -> Ignore (Ctrl+I)



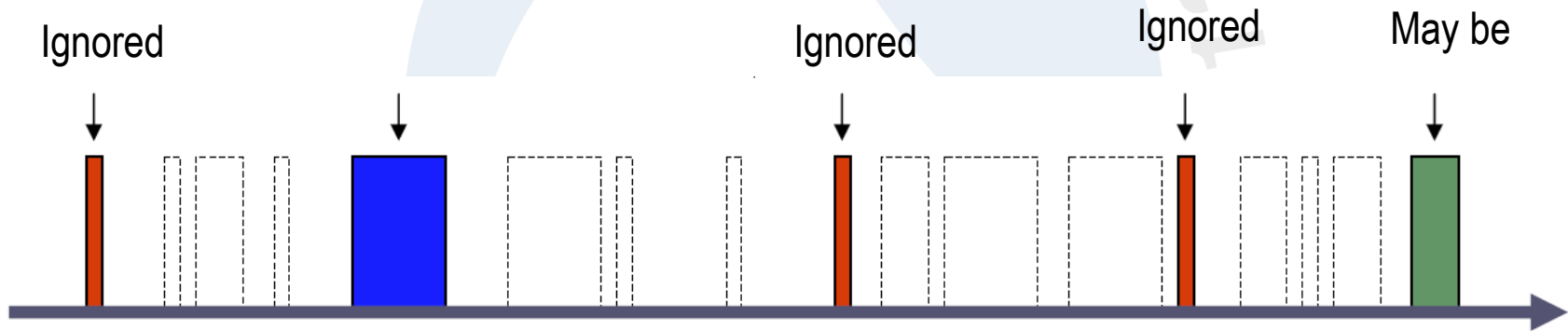
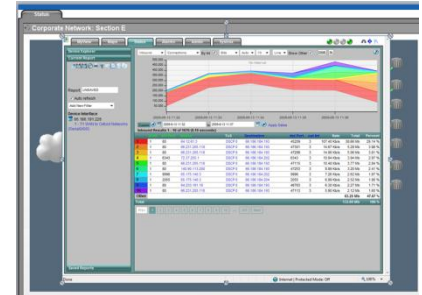
# Expertize IO graph

- To see errors and counting the number of packet, set Y axis to packet/sec ( histogram )
- To see performance and throughput, set Y axis to bit/sec ( line )



# No sampling, non-sampling

- In old days we use sampling technologies like SNMP, MRTG, and many flow analysis such as Cisco NetFlow, sFlow, iFlow

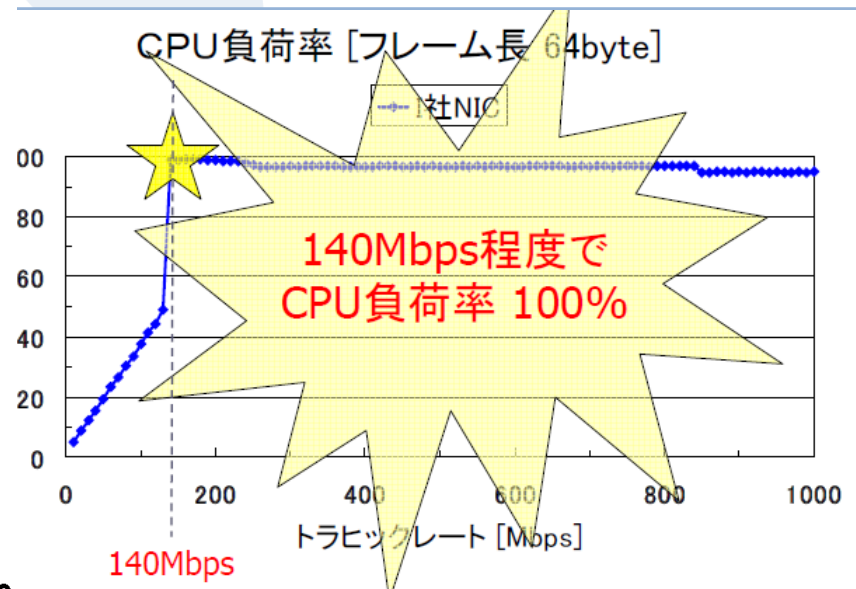
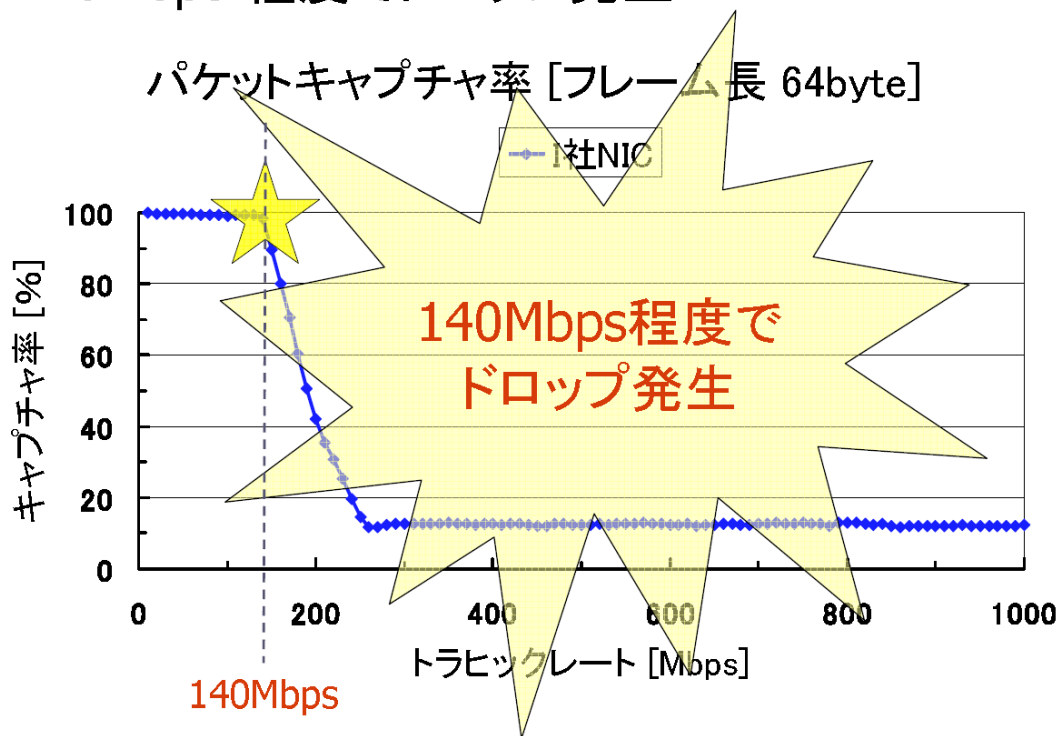


- But small packet ( 64 bytes – 100 bytes ) may be ignored. Some small packet is important symptom of analysis ( ARP / TCP SYN / HTTP GET and others )

# Actual capture rate

- Typical Intel's GigaNIC (e1000), typical Dell PowerEdge2850 / Xeon 2.8GHz RAM 1GB (PC3200, DDR2, 400MHz)
- Threadshoud is 140Mbps in Frame size = 64

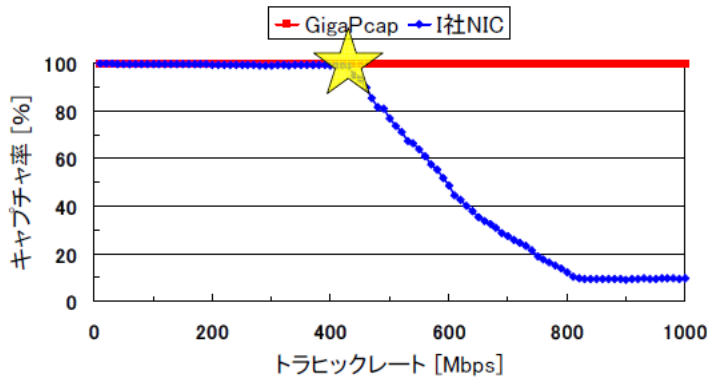
140Mbps 程度でドロップ発生



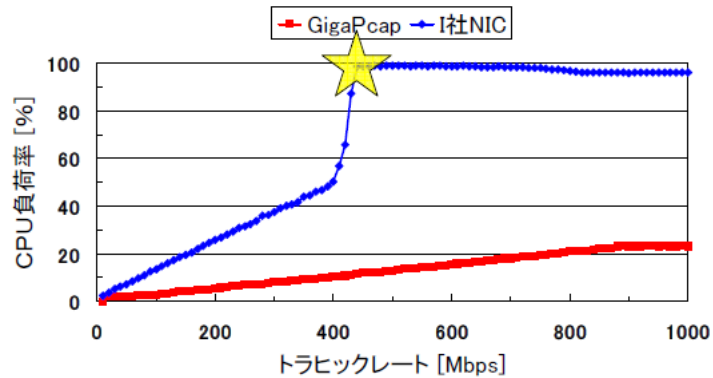
# Another frame size

- Frame size = 200 , actual rate 400Mbps
- Frame size = 1500 , may be ok, no problem.

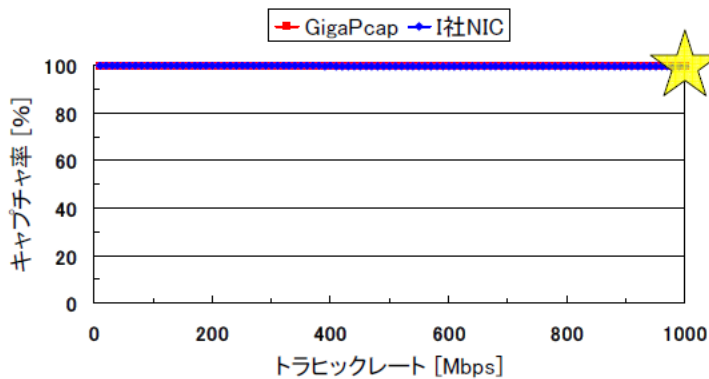
パケットキャプチャ率 [フレーム長 200byte]



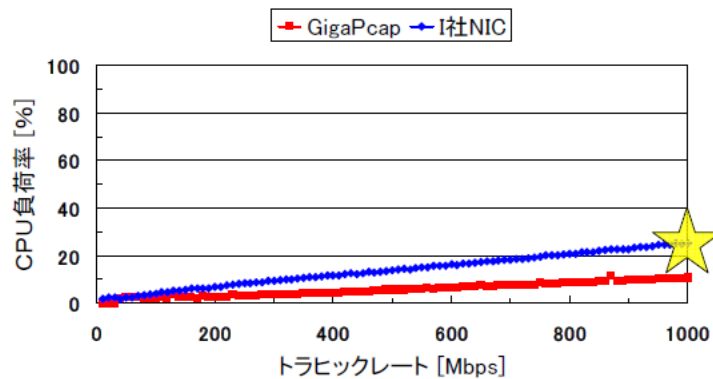
CPU負荷率 [フレーム長 200byte]



パケットキャプチャ率 [フレーム長 1518byte]



CPU負荷率 [フレーム長 1518byte]



# Ooops in non-sampling

- In case of frame size is 1500, there may be some drops ( it is not non-sampling )
- Actually, customer want to see most highest point of traffic, so if the pcap file do not contains all packet ( some ignored ) no use.
- Use TurboCaps and other capture designed NIC based on FPGA ( ikeriri is a reseller !!)
- No less than 200MB is good for browse.
- Once get pcap, then make report and drop it.



# Tradeoff between pcap size

Off course there are many packet aggregator and data collection devices such as 19 inch rack mount one or Openflow packet tracer and we can use additional RAID harddisks ready for exabytes and more than a year

## Time and Cost is important



Small factor PC with SSD and TurboCap or some can do it

**Some customer has 10 gigabytes in a hour, how about making report not weekly, daily but hourly**

1. get pcap
2. Add record to RDB
3. Drop it
4. Indexing
5. make report

# For non-sampling inspection

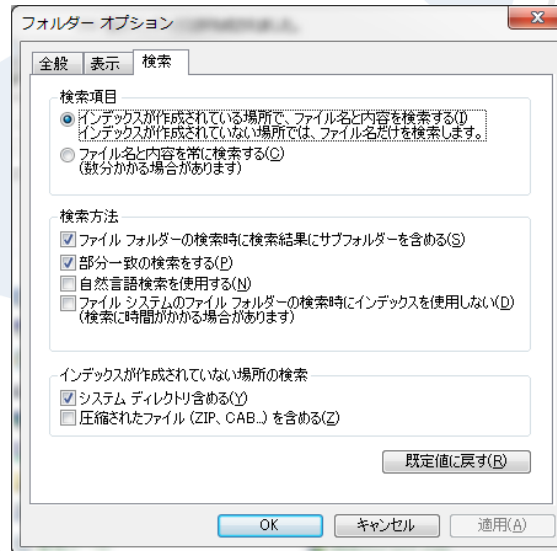
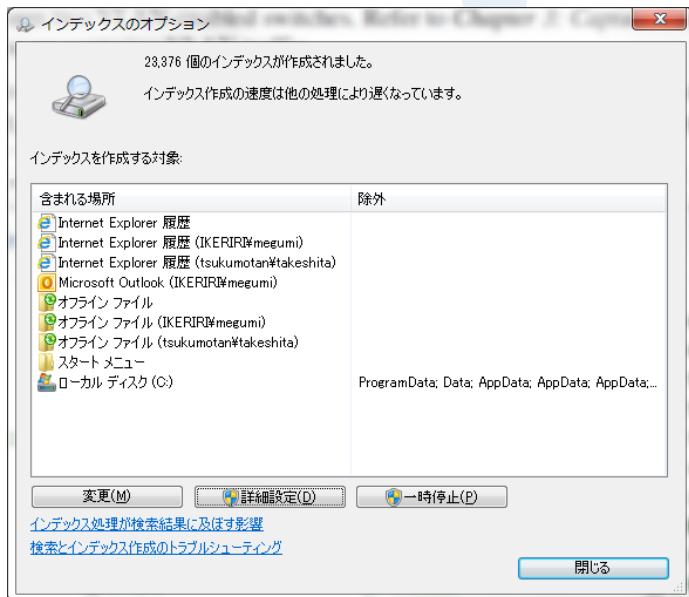
- I experienced “MMMM packets received by the application NNNN packets accepted by the filter and dumped to disk ummm” via turboCap API
- Optimise I/O access flow using FIFO(queue) packet -> IRQ -> SVC -> driver -> OS
- Use 6 cores Xeon-L5640 and 24GB RAM !  
( power resolve things and no page files )
- Stop tcpdump and create program using pcap libraries in C/C++ basically and low level one
- Pcap -> standard output -> FIFO -> SQLite
- 3 month no problem good





# One more

- Using Window Search !! Cool  
To add extension of cap and pcap,  
set type as clear text search,  
We can search pcap/cap files like Google !  
off course in multibytes ( in Japanese )
- Control panel -> index option / folder option



# Thank you

# ありがとうございます！



# Arigato gozaimasu

