Detecting and troubleshooting

high CPU load

on ERS 8600

Version 1.4

14.02.2008 Juergen Arlt

Table of contents

Version History

Version	Owner	Date	Comment
0.4	jarlt	04.05.07	draft
0.5	jarlt	15.05.07	formatting completed
0.6	jarlt	20.05.07	included comments from lstevens
0.7	jarlt	22.05.07	included comments from sadams
1.0	jarlt	25.05.07	reviewed by rmorrison
1.1	jarlt	28.06.07	added ARU test task
1.2 -1.3	jarlt		Internal comments
1.4	jarlt	14.02.08	source MAC learning

CPU work

General comments:

Currently three different (enterprise) CPU/Switching Fabric models exist: 8690SF, 8691SF and 8692SF.
In general the CPU part (front blade of the combined CPU/Switching fabric module) is the same for 8692SF and 8691SF though the switching fabric part is different. The 8692SF and 8691SF provides an additional socket for an additional (separately orderable) CPU daughter card called 'Super Mezzanine card' that holds two 1GHz processors. The Mezzanine Card is however not supported on the 8691SF. The 8690SF is the first model of the CPU blades and less powerful than the new CPUs. The 8690SF will not be supported with 5.0 software.
There are some exceptions with the CPU for MERS 8600 which will not be covered here.

This document is based on the ERS 86XX product and Software version 4.x.x. Results with other Software versions may vary.

The CPU tasks

The CPU is responsible for maintaining all forwarding information by collecting and exchanging relevant data with other network devices or leaning the data by monitoring certain traffic. For this the CPU receives, exchanges and maintains ARP packets, MAC addresses, Routing tables (RIP, OSPF, BGP), Spanning tree, VRRP, IST, Multicast, VLAN information and filter entries from the configuration file and stores these in tables (records). Once all this is collected it is sent to the ASICs on the I/O blades where the forwarding decision is done independently from the CPU. The CPU however verifies the information and is responsible for refreshing/aging of the appropriate records.
Apart from normal routing protocols certain configurations require additional exchange of record information with peer devices. Examples would be IST clustering for MAC addresses between the IST peers and HA mode for layer 2 and layer 3 information between two CPUs in the same chassis.

The CPU handles all management traffic to the ERS 8600 itself such as SNMP polling and traps, Telnet, SSH, FTP, TFTP … and other network control traffic such as Radius, ICMP (reply, redirect) DHCP depending on the actual configuration and the traffic passing the switch.

If configured the CPU collects traffic relevant Data for RMON monitors, triggers and generates alarm messages.

The CPU monitors the hardware of the device polling the information of the whole switch such as the temperature power supplies, GBIC change, card change … Based on the hardware monitoring and software event the logging information is generated and stored in memory or on the file system.

The CPU controls the Out of Band Ethernet port and Console connectors and maintains the file system with the images logs and config data.

Other tasks performed by the CPU

- PCAP (only backup CPU)
- maintaining the Config information
- running the daemons for the different services such as Telnet, FTP, TFTP, SSH, SNMP
- running the CLI

The CPU will not (except in certain exceptional circumstances) forwarding data traffic. Data traffic is handled by the switching fabric on the same blade (back half of the 869xSF blade).

Detailed list of packets received or transmitted from the CPU

Management traffic to the 8600

ICMP request, reply, traceroute
Telnet,
SNMP interactive session, traps
SSH,
FTP,
TFTP
RMON, IPFIX collector traffic

Network Control Traffic

STP BPDUs
ARP
All frames on layer 2 interface where source address is unknown
ICMP redirects (generated from CPU when no E module is present)
ICMP destination unreachable
ICMP TTL expired
VRRP
IST message exchange (including IST keepalives)
Radius traffic for authentication, accounting
Topology discovery frames (proprietary SONMP or in later rev 802.1AB LLDP)
ICMP router discovery
DHCP/BootP proxy
IGMP join, leave, query
LACP, VLACP
SLPP

Routing protocols

RIP updates
OSPF Hellos, LS-Updates, Database Exchange
BGP (TCP-) keepalives, updates
Multicast routing protocol traffic (PIM BSR, RP, PIM updates DVMRP)

Single Data packets

IP packets with TTL 1 to be routed (as the TTL would go to 0 during the routing process)

IP packets to a local network when the ARP entry is not resolved yet
(if packet needs routing to local subnet but no ARP entry exists, the packet goes to CPU, the CPU ARPs out for destination (and sets a temporary discard ARP record to prevent more traffic for same destination to hit CPU), when ARP reply is received, the CPU updates ARP entries, and forwards the initial packet)

IP packets with the next hop on the same network where they are received from (ICMP redirect situation) when pre-E-module received the traffic

PIM first packet per stream (later switchover)

Data traffic through the CPU

- when unicast Routing tables are too large for ASICs then software routing in CPU
- Multicasts traffic when MGID table full

The CPU load

High CPU utilization in most cases, indicates problems in the network. However, a CPU load up to a certain amount is expected for processing the network control and forwarding information.

Depending on the size of the network the CPU might have to process a large amount of information which might lead to a significant CPU load. In large networks e.g. with 3000 nodes or more the CPU needs enough time and processor ticks to refresh all ARP and MAC entries in certain intervals. Based on the configuration a large amount of entries might age out at the same time which could cause a significant CPU load for several seconds.

During these times the CPU load might spike to an amount of 100% and may even stay there for some seconds but should then return to a normal state which is likely below 20% in smaller networks and below 40% in larger networks. If the CPU however shows 100% load over a longer period of time or an average load of more than 50% usually problems exist in the network and need to be resolved.

Reaching the 100% load for a certain time does not necessarily cause problems in the network or on the affected ERS 8600. The data is still sent according to the forwarding information through the I/O modules and the switching fabric and the control data should still be processed according to their priority in the CPU using a scheduler that allows all processes to gain CPU ticks.

However these conditions should be monitored and addressed with configuration or network changes or if the load is caused by a process that does not release the CPU correctly with a software fix.

Monitoring is either done trough the execution of ‘show sys perf’ or polling the MIB variable 1.3.6.1.4.1.2272.1.1.20
(iso(1).org(3).dod(6).internet(1).private(4).enterprises(1).rapidCity(2272).rcMgmt(1).rcSystem(1).rcSysCpuUtil(20)).
Both methods provide the actual CPU load at polling time, there is no average over a certain amount of time.
To gather information over a longer period of time continuous polling is required, which can be done though a script or any MIB collecting and/or graphing tool. With JDM the CPU load can be monitored under Graph-> Chassis->CPU Util.

Note: If polling of the CPU from a network management system takes place together with polling of various other MIB variables it may result in a higher than expected CPU load as the polling of many values simultaneously is in itself, CPU intensive.

Symptoms of high CPU load

Direct monitoring of the CPU load shows a high load over several seconds or spikes.

Responses on Telnet or SNMP are slow or timing out as a result the 8600 in JDM may turn gray for all ports.

Ping outages to the 8600 itself where typically the traffic through the 8600 is not affected during the first seconds of a high CPU load situation.

VLACP ports on one 8600 are shut down at the same time to different locations without link loss. This happens depending on the VLACP timers eventually after 3 sec.

VRRP switchover messages are seen on the affected 8600 from Backup to Master and/or on peer VRRP router from Backup to Master due to not receiving the VRRP hellos from the master so that at the same time two master VRRP routers are present in the network. With default timers this happens 3 seconds after the high CPU load started.

```
SNMP INFO Vrrp State Transition Trap(Port/Vlan=19, Type=3, Cause=5, VrId=110,
VrIpAddr=192.168.33.110, Addr=192.168.33.108)
```

VRRP Type 3 = Backup to Master
VRRP Cause 5 = Master Down interval

40 seconds (default OSPF dead interval) after the CPU load went up and is not going back to normal typically OSPF transitions are seen in the log stating that the OSPF neighbor has been lost due to expired inactivity timer.

```
OSPF INFO Ospf Nbr State Change: rtid:10.244.30.131, ipa:10.240.127.33 nbr-
rtid:10.244.30.130 FULL->DOWN
```

When OSPF neighbors lose their connectivity, routing information is lost which may have a severe impact on the network. Remote devices (typically in other VLANs are not reachable whereas devices in the same VLAN can still be reached.)

If the CPU remains in a loaded state for 50 seconds the IST will time out, changing all SMLT states to down. In the log no port down is reported in advance (otherwise it might be a physical link problem)

```
CPU5 [07/14/06 09:19:52] MLT INFO smltIstSessionDown
CPU5 [07/14/06 09:19:56] MLT INFO All the SMLTs are down
```

If the IST is down certain traffic might not reach the destination, as forwarding tables can no longer be synchronized between the 8600s in the cluster. In symmetrical environments there might be less impact to network connectivity.

A full cycle of the above mentioned symptoms:

```
Apr  7 09:10:15 ppbc100-1 CPU6 [04/07/06 07:10:16] SNMP INFO Vrrp State Transition
Trap(Port/Vlan=993,     Type=3,     Cause=5,     VrId=93,     VrIpAddr=10.128.93.1,
Addr=10.128.93.3)
Apr  7 09:10:15 ppbc100-1 CPU6 [04/07/06 07:10:16] SNMP INFO Vrrp State Transition
Trap(Port/Vlan=995,     Type=3,     Cause=5,     VrId=95,     VrIpAddr=10.128.95.1,
Addr=10.128.95.3)
Apr  7 09:10:15 ppbc100-1 CPU6 [04/07/06 07:10:16] SNMP INFO Vrrp State Transition
Trap(Port/Vlan=997,     Type=3,     Cause=5,     VrId=97,     VrIpAddr=10.128.97.1,
Addr=10.128.97.3)
Apr  7 09:10:42 ppbc100-1 CPU6 [04/07/06 07:10:43] OSPF INFO Ospf Nbr State Change:
rtid:10.128.0.2,    ipa:10.128.82.3    nbr-rtid:10.128.0.1       FULL->DOWN    Event
INACTIVITY_TIMER_EVENT
Apr  7 09:10:47 ppbc100-1 CPU6 [04/07/06 07:10:48] OSPF INFO Ospf Nbr State Change:
rtid:10.128.0.2,    ipa:10.128.1.2    nbr-rtid:10.128.0.1       FULL->DOWN    Event
INACTIVITY_TIMER_EVENT
Apr  7 09:10:47 ppbc100-1 CPU6 [04/07/06 07:10:48] OSPF INFO Ospf Nbr State Change:
rtid:10.128.0.2,    ipa:10.128.51.2    nbr-rtid:10.128.0.1       FULL->DOWN    Event
INACTIVITY_TIMER_EVENT
Apr  7 09:11:04 ppbc100-1 CPU6 [04/07/06 07:11:04] MLT INFO smltIstSessionDown
```

In the log warning messages are generated that messages are lost due to high CPU load

```
WARNING: 130 log messages lost due to HIGH CPU LOAD.
```

Common reasons for high CPU load

Broadcast and multicast storms

Broadcast and multicast storms are the most common cause of CPU loading. Broadcasts and (depending on the configuration) multicast packets are flooded onto all ports of the VLAN. In addition, as there is no limited packet lifetime in layer 2 environments (such as the TTL in IP) they may loop forever under certain conditions. Depending on the speed of the links and the forwarding capability of the devices several thousands of the same packets per second can be seen. If these packets have to be processed by the 8600 CPU (see list of packets) the CPU might not be able to process this information fast enough and may therefore continuously busy until the loop has been broken. Depending of the type of the looping packets they might be prioritized over other traffic coming into the CPU and may therefore block other important traffic or processes. An example would be a looping VRRP packet which has a very high internal priority.
Broadcast/multicast loops can easily be detected using port statistics; the huge amount of non unicast packets should be immediately visible.
Along with the high CPU load, the CPU buffer consumption typically increases and may reach 100%.

To prevent CPU load from loops the CP-limit or extended CP-limit feature together with loop breaking technologies such as SLPP, Spanning Tree at the edge, BPDU filtering and Tagging with discard untagged frames for switch to switch connections should be used.

The majority of loops are created due to incorrect cabling in the core and at the edge or invalid device configuration including cases where devices start in default configurations or without configured MLTs.

ARP

The ERS 8600 as a Layer 3 switch keeps track of both Layer 2 (source MAC address /port mappings) stored in the forwarding tables as well as Layer 3 (IP address to MAC address) in the ARP table. In order to ensure consistency between tables, whenever an entry ages out of the layer 2 forwarding database (by default after 300 seconds) the ERS will ARP for this entry in order to validate that the device is still active on the network. In a network with large numbers of entries in the forwarding table this behavior will, by default, generate sequences of IP ARP broadcasts.

If the network configuration is relatively static (i.e. Stations are rarely moved between physical ports) the fdb timer can be increased to reduce this traffic. Command: config vlan <vlan-nr> fdb-entry aging-time 21660

Network Management systems tend to discover network devices by ARP-ing or pinging whole network ranges or pinging network addresses. To resolve these entries the ERS may have to send ARP requests for each station and potentially needs to handle a huge amount of replies at the same time. Depending on the network size and the polled range this may cost a significant amount of CPU ticks for processing the requests and replies.
Network Management systems should be configured to discover only existing networks and not at the highest possible discovery rate though it is nice if a discovery process would take less time.

SNMP polling

A high amount of SNMP polling might add significant load to the ERS system if the management station is polling a high number of entries in a short amount of time such as the routing, ARP or forwarding table in a large network.

The increased CPU load is occurring in fixed intervals (typically all 10 min) and lasts for some seconds depending on the polled information. It is often seen on all other network devices with a certain time difference.
Most management systems allow the definition of the polled information and the number of requests per second to offload the polled devices.

Spanning Tree

A common cause of slow performance as well as high CPU load on the ERS 8600 can be underlying spanning tree instability in the network.
For example BayStack switches create a TCN (Topology change notification) message whenever a spanning tree enabled port changes its status. The root bridge upon receiving TCNs will send out BPDUs with the topology change flag set, which in turn will cause all switches in the spanning tree domain to fast-age (15 sec) their forwarding databases. This could for the user result in slow performance, excessive flooding of traffic by switches and a high CPU load on the ERS 8600 (as the unit is constantly busy updating the forwarding databases).

ICMP redirects / unreachable

When multiple routers are connected to the same VLAN a large amount of ICMP redirects may be generated depending on the configuration of the non routing devices and their default route. If a packet is received by one router and has according to the routing table to be sent to another router on the same network where the packet was received on, the sending station was obviously not using the best next hop for that packet. Therefore the first router tries to inform the sending station to send the next frame to the second router directly. This is done with an ICMP redirect frame. To generate this frame on the ERS 8600 a non-E-module is forwarding the frame but also copying the frame to the CPU for letting it generate an ICMP redirect frame. In large networks these frames could lead to a considerable load of the CPU.
E- and R- modules are generating the ICMP frames independently of the CPU and are therefore offloading the CPU from that frame processing.

In certain configurations a route summary router can attract traffic for which it does not have a specific destination route. If ICMP unreachable message generation is enabled on the switch, this may result in a high CPU utilization. To avoid such a scenario, Nortel recommends that a black hole static route is used whenever a route summary is configured.

Creation of table entries

The creation of any table entry requires a certain amount of CPU cycles. If a huge number of new table entries in the 8600 is generated in a relatively short amount of time the CPU may spike to high values. This has been seen at a customer where a virus on many PCs was generating always new multicast packets causing the 8600 to add permanently new multicast table entries.

Permanent MAC address learning

A packet with an unknown source MAC address or a known source MAC on a new port will be sent to the CPU. The CPU then programs the source MAC address into the ARUs to have the address in the forwarding database. In case of a network mis-configuration the same source MAC address may permanently appear on different ports. This will force the ARU to constantly send the packets with the ‘new’ port to the CPU generating a high load. The CPU trace would show ‘normal’ packets with the same source MAC on different ports.

```
[02/12/08 16:29:51:16] tMainTask CPP: cpp.c     : 2353: cppProcRxFrame: dst=00-09-
97-b4-72-08 src=00-04-38-75-56-00 typ=0800 len=63
port=1/7 vid=0xc1c pid=1 code=0101 qos=0 pkthdr 0x8006004b 0x1c1ca840
[02/12/08 16:29:51:16] tMainTask CPP: cpp.c     : 2353: cppProcRxFrame: dst=00-09-
97-b4-72-08 src=00-04-38-75-56-00 typ=0800 len=60
port=2/8 vid=0xc1c pid=1 code=0509 qos=1 pkthdr 0x808f0044 0x19e34240
[02/12/08 16:29:51:16] tMainTask CPP: cpp.c     : 2353: cppProcRxFrame: dst=00-09-
97-b4-72-08 src=00-04-38-75-56-00 typ=0800 len=62
port=2/8 vid=0xc1c pid=1 code=0409 qos=1 pkthdr 0x808f004d 0x1c1c0240
[02/12/08 16:29:51:16] tMainTask CPP: cpp.c     : 2353: cppProcRxFrame: dst=00-11-
58-54-b2-11 src=00-04-38-75-56-00 typ=0800 len=60
port=2/8 vid=0xc1c pid=1 code=0409 qos=1 pkthdr 0x808f0044 0x1c1c0240
```

Reasons are either incorrect MLT configurations, unsupported server clustering/load balancing configurations, loops in the network or incorrect mirroring configurations. The path to the toggling source MAC addresses needs to be determined and corrected.

Process in the CPU is not giving back control to other processes

Though there have been a lot of improvements in software 3.7 and 4.1 it may still happen that one process is not giving back control to the CPU after a certain amount of time. This would lead to a situation that other processes are not able to work on their tasks such as monitoring keepalives or keeping adjacencies up. These issues can only be detected in the shell mode. They are however typically a result of certain traffic to the CPU.
An example is Q01541086 where a broken communication to the IPFix collector may not release the CPU control leading to outages in VRRP and IST communication. The problem is scheduled to be fixed in 4.1.4.0.

Troubleshooting

Monitoring the Interval of high CPU load

The first troubleshooting step should be to determine the times and intervals of the high CPU load. Most CPU issues can quickly be identified once knowing if the high load is reoccurring at regular intervals and correlate these to network timers (e.g. MAC address refresh) or actions (e.g. server backups).
To detect these intervals polling of the MIB variable every 1 or 2 seconds with a graphical tool is the preferred method. Polling the CPU every 1 or 2 seconds will not add a significant load onto the device so it is not critical for the network.

Significant network intervals (default timers)

30 sec	RIP update
6 min	MAC address timeout
6 min	MAC address refresh in IST clusters
10 min	typical SNMP polling intervals
30 min	OSPF LS refresh

8:00 – 10:00 and 13:00 - 15:00 highest user traffic in networks
0:00 – 3:00 typical server backups

Monitoring port statistics

During network loop situations that may cause a significant CPU load the port statistics may identify the affected part of the network showing a huge number of broadcast and multicast packets on certain ports.

These statistics can be read either through CLI or SNMP based tools querying the interface entries of the MIB II. The Java device Manager will display these statistics under Graph -> Port -> Interface.

To start from a defined point it is always useful to clear the port statistics either trough CLI (‘clear port stat’) or within the SNMP tool.

The command ‘show port stat interface’ with the required sub-context displays information such as the number of packets and bytes in total and unicast and multicast frames sent ore received.

```
ERS-8610:6# show port stat interface main port 1/1

================================================================================
                          Port Stats Interface
================================================================================
PORT     IN         OUT        IN        OUT         IN        OUT     OUTLOSS
NUM    OCTETS     OCTETS     PACKET     PACKET     FLOWCTRL  FLOWCTRL  PACKETS
--------------------------------------------------------------------------------
1/1    762778     119832     1613       1534       N/A      N/A      0

ERS-8610:6# show port stat interface extended port 1/1

================================================================================
                          Port Stats Interface Extended
================================================================================
PORT_NUM IN_UNICST  OUT_UNICST IN_MULTICST  OUT_MULTICST IN_BRDCST  OUT_BRDCST
--------------------------------------------------------------------------------
1/1      1026       1023       567          525          32         0
```

Alternatively the CLI ‘monitor’ command can be used to monitor port statistics in real time:

```
monitor port stat interface util  1/1-/18
```

Typically for a high CPU load the number of multicast or broadcast packets in the extended sub-context is the most relevant information. A high number of packets (several hundreds per second or more) typically indicate loops or stations that are driving up the ERS CPU load.

CPU Trace

Working with the CPU trace

As high CPU load is most commonly a result of packets received or transmitted by the CPU this traffic can be traced from the CLI of the ERS 8600.

The most informative trace level is level 9 3. It outputs description lines of packets that hit the CPU port, both receive and transmit.

```
[000   19:38:00:752]   tCppRxTask   CPP:   cppRxTask:   dst=ff:ff:ff:ff:ff:ff(1dd0400,
1dd0408) src=20:02:a2:30:0c:a6 typ=0806 len=50 port=2/8 (Quid 05 Port 3) vid=010
pid=2 ipec=00 squidport=2
```

The output shows source and destination MAC-addresses, the Ethernet length or protocol type and the port where the frame was received or to which it was transmitted. Further details about the frames can be obtained by encoding the diagnostic code.

Additional trace modules may help further troubleshooting. Refer to the documentation for additional information about all available trace modules.

Note: Be aware that tracing on the CPU will add load to the CPU as well, so the usage of tracing should be agreed by the customer as well.

There are two ways to obtain trace information. The trace is automatically written into a 64K buffer which wraps when full, which means older messages are lost. The trace can constantly be redirected to the console screen by using the 'screen on' option. With a PC connected to the console, which has a terminal program running that has capability to record text into a file, trace can be captured over a longer period.

Note: Tracing to screen is more CPU intensive especially if the amount of incoming data is higher than the speed it can be displayed. In critical situations it is therefore recommended to trace to the buffer and display the buffer several times in a row to get a broad picture.

Note: It is not recommended to do screen tracing over Telnet connections as every frame that is sent to/received from the Telnet client will cause an additional CPU trace entry.

To configure CPU tracing and displaying the results of the buffer the following configuration can be used:

```
config
trace
clear
level 9 3
```

Wait 30 seconds

```
level 9 0
info
```

If the trace directly to the console is needed (see notes above) the configuration would be like this.

```
config
trace
clear
level 9 3
screen on
```

To disable the screen tracing one of the following commands can be used:

```
level 9 0
screen off
```

The commands have to be typed during the running output of the screen.

To turn off the tracing completely the trace level has to be set back to 0.

```
level 9 0
```

CPU tracing configurations are not saved in the config file, they are lost after the next reboot.

Auto-tracing of the CPU

To gather information automatically when the CPU load reaches a certain percentage auto-tracing can be used. It will start tracing the configured modules (usually this will be the packets processed by the CPU which is module 9) when the high percentage threshold is exceeded for high-track-duration time (5 sec by default). When the CPU load drops below the low-percentage value for the configured time low-track-duration (5 sec by default) the tracing will be stopped.

```
config
trace
auto-enable
add-module 9 3
high-percentage 80
low-percentage 60
auto-trace enable


Sub-Context:
Current Context:

    add-module <modid> <level>
    auto-trace <enable|disable>
    high-percentage <percent>
    high-track-duration <seconds>
    info
    low-percentage <percent>
    low-track-duration <seconds>
    remove-module <modid>
```

Saving the CPU traces

To save the traces automatically a boot flag can be set:

```
conf boot flag trace-logging true
```

Note: If the auto-trace is constantly triggered, this file will grow rapidly. Therefore the file size and free space of the file system has to be monitored closely.

Note: If the number of events is higher than the data rate that can be written to the PCMCIA it may cause severe problems on the 8600, increase the CPU load and cause other processes to fail.

The traces can also be saved manually:

```
save trace file /flash/trace
```

or if an TFTP server is available:

```
save trace file 192.168.17.5:/trace
```

Expected entries in the CPU trace

When reading CPU traces the amount of packets within a certain time is the key to troubleshoot high CPU load situations successfully. It is therefore vital to check the timestamps for the trace entries before making assumptions about load caused by packets. Depending on the packet type a CPU has been tested to be able to handle up to 90,000 packets per second before reaching 100% load.

Certain packets will always appear in CPU traces sometimes in large numbers, depending on the network configuration.

Nortel proprietary Topology discovery frames are sent and received on every port all 10 seconds. Two frames with different destination multicast MAC addresses (01-00-81-00-01-00 and 01-00-81-00-01-01) are usually seen in the traces.

```
[04/05/07 09:20:12:433] tMainTask CPP: cpp.c     : 2492: cppProcRxFrame: dst=01-00-
81-00-01-00  src=00-04-38-01-84-48  typ=0013  len=60  port=1/2  vid=0x0fa  pid=254
code=0460 qos=1 pkthdr 0x00810040 0x00fa1800
[04/05/07 09:20:12:433] tMainTask CPP: cpp.c     : 2492: cppProcRxFrame: dst=01-00-
81-00-01-01  src=00-04-38-01-84-48  typ=0013  len=60  port=1/2  vid=0x0fa  pid=254
code=0460 qos=1 pkthdr 0x00810040 0x00fa1800
```

ARP frames are sent and received from the CPU. Usually the CPU refreshes the ARP entry all 6 minutes before aging MAC addresses so depending on the network size a large number of theses frames might be seen. Protocol for ARP is 0806.

```
[04/05/07 09:20:35:149] tMainTask CPP: cpp.c     : 2492: cppProcRxFrame: dst=ff-ff-
ff-ff-ff-ff src=00-04-dc-9b-ee-01 typ=0806 len=60 port=1/1 vid=0x384 pid=2 code=080a
qos=2 pkthdr 0x81000044 0x43840280
```

VRRP hello frames are either received or transmitted (depending on the VRRP state of the ERS) on every VLAN all second. If VRRP fast timers are used even more packets will be seen. The destination MAC address is 01-00-5e-00-01-xx, QoS = 7

```
[03/14/07 18:13:14:349] tMainTask CPP: cpp.c     : 2889: cppProcRxFrame: dst=01-00-
5e-00-00-12 src=00-00-5e-00-01-65 typ=0800 len=60 port=1/4 vid=0x065 pid=1 code=1c00
qos=7 pkthdr 0x83830040 0x00650000, PktProcCode 0
```

If Spanning Tree is enabled the BPDUs are received and transmitted on every port all 10 seconds. The destination MAC address for STP BPDUs is 01-80-c2-00-00-00.

```
[04/27/07 11:50:15:499] tMainTask CPP: cpp.c     : 2492: cppProcRxFrame: dst=01-80-
c2-00-00-00  src=00-04-38-db-a1-5e  typ=0026  len=60  port=1/5  vid=0xfff  pid=252
code=1c40 qos=7 pkthdr 0x03840040 0x0fff1000
```

Telnet/FTP/TFTP connections to the switch itself are logged with the destination address of the switch itself and protocol 0800

```
[04/05/07 09:20:34:249] tMainTask CPP: cpp.c     : 2492: cppProcRxFrame: dst=00-04-
dc-99-06-00 src=00-04-dc-9b-ce-00 typ=0800 len=60 port=1/1 vid=0x3e8 pid=1 code=1f80
qos=7 pkthdr 0x03800044 0xe3e8e000
```

Unexpected entries in the CPU trace

Most of the following frames are still expected to be received and processed by the CPU however huge amount of these frames in the CPU trace may indicate problems in the network and might be the reason of a high CPU load.

As a rule of thumb a high number of packets is about 50 or more packets of the same type in an interval of a second or less.

Note: Certain packets such as ARP, VRRP or Spanning Tree might be seen more often in a short interval depending on the network size and configuration.

Sample trace for ICMP redirect issue. In this case the CPU receives packets that are destined to the router instance of the ERS 8600 (here the VRRP functional address 00-00-5e-00-01-7a). Normally the 8600 should route this packet in hardware and the CPU should not see it. However the code 0103 indicates that the packet was sent to the CPU to generate an ICMP redirect message for the source of the packet to send the next to another router on the same network. If the source does not care about the ICMP redirects (it might be a firewall which is doing this for security reasons) and continue to send all further packets to the 8600 the CPU is kept busy receiving the packets and sending ICMP redirects.

```
[04/27/07 11:50:19:433] tMainTask CPP: cpp.c     : 2492: cppProcRxFrame: dst=00-00-5e-00-01-7a  src=00-0b-cd-c5-72-b5  typ=0800  len=1514  port=1/4  vid=0x07a  pid=1 code=0103 qos=0 pkthdr 0x800305f6 0x007a40c0
[04/27/07 11:50:19:433] tMainTask CPP: cpp.c     : 2492: cppProcRxFrame: dst=00-00-5e-00-01-7a  src=00-0b-cd-c5-72-b5  typ=0800  len=1514  port=1/4  vid=0x07a  pid=1 code=0103 qos=0 pkthdr 0x800305f6 0x007a40c0
[04/27/07 11:50:19:433] tMainTask CPP: cpp.c     : 2492: cppProcRxFrame: dst=00-00-5e-00-01-7a  src=00-0b-cd-c5-72-b5  typ=0800  len=1514  port=1/4  vid=0x07a  pid=1 code=0103 qos=0 pkthdr 0x800305f6 0x007a40c0
```

Sample for IGMP packets hitting the CPU

```
[03/28/06  13:43:13:683]  tMainTask  IGMP:  igmp_main.c:  455  :  igmpTurnAroundFrame: src_port = 3/7 sipa = 10.199.49.74 dipa = 224.0.1.60 group = 224.0.1.60
[03/28/06  13:43:13:683]  tMainTask  IGMP:  igmp_main.c:  455  :  igmpTurnAroundFrame: src_port = 2/6 sipa = 10.199.47.157 dipa = 224.0.1.60 group = 224.0.1.60
[03/28/06  13:43:13:683]  tMainTask  IGMP:  igmp_main.c:  455  :  igmpTurnAroundFrame: src_port = 2/5 sipa = 10.199.41.52 dipa = 224.0.1.60 group = 224.0.1.60
```

Novell Service Locator Protocol

```
[03/09/06 10:59:14:683] tMainTask CPP: cpp.c     : 2348: cppProcRxFrame: dst=01-00-5e-00-01-16 src=00-04-38-09-b6-10 typ=0800 len=176 port=8/8 vid=0x04f pid=1 code=078c qos=1 pkthdr 0x00af00b8 0x004fe300
[03/09/06 10:59:14:683] tMainTask CPP: cpp.c     : 2348: cppProcRxFrame: dst=01-00-5e-00-01-16 src=00-04-38-09-b6-10 typ=0800 len=176 port=8/8 vid=0x04f pid=1 code=078c qos=1 pkthdr 0x00af00b8 0x004fe300
[03/09/06 10:59:14:683] tMainTask CPP: cpp.c     : 2348: cppProcRxFrame: dst=01-00-5e-00-01-16 src=00-04-38-09-b6-10 typ=0800 len=176 port=8/8 vid=0x04f pid=1 code=078c qos=1 pkthdr 0x00af00b8 0x004fe300
```

spyReport

Working with spyReport

The spyReport, accessed using privilege mode can be used to get information about the process load of the device. It lists all current running processes of ERS 8600 with their corresponding total and delta (since the last spyReport) CPU cycles (ticks).

To get into privilege mode a password is required which can be obtained by Nortel technical support. This password will last for a limited time only (end of the month) and requires the chassis serial number, the software version running on the 8600 and the actual date of the 8600. (use the ‘show sys info’ and ‘date’ commands)
For entering the privilege mode the command ‘priv’ is used and the password is requested. The privilege mode is indicated by an asterisk in front of the CLI prompt.

```
ERS-8610:6# priv
Password: ******
Entering privilege command mode
*ERS-8610:6#
```

Note: Working in privilege mode does not introduce a particular risk to the device (as opposed to the shell mode) but it is recommended logging out after finishing the spyReport work.

The command is ‘spyReport’ without any additional parameter. It will return the following list:

```
*FTMCS8600_1:5# spyReport

NAME          ENTRY          TID   PRI   total % (ticks)  delta % (ticks)
--------     --------      -----   ---   ---------------  ---------------
tExcTask     excTask     14261c8    0      0% (       7)    0% (       0)
tLogTask     logTask     1427f98    0      0% (       0)    0% (       0)
```

```
tWdtTask     wdtTask     15f9398    1     0% (    5263)    0% (       2)
pcmcia       CS_callbac  14c1c90    6     0% (       0)    0% (       0)
pcmcia_poll  CS_poll_ta  14c5ea0    6     0% (     319)    0% (       1)
rtMainTask   realMainTa  1e12718   48     0% (    1092)    0% (       0)
tNetTask     netTask     143da60   50     0% (    2711)    0% (       0)
tTftpdTask   tftpdTask   150d310   55     0% (       0)    0% (       0)
tFtpdTask    ftpdTask    1510400   55     0% (       0)    0% (       0)
tCppSend     cppSocketT  1e17928   79     0% (     926)    0% (       1)
tChasServ    chServiceT  261d2d8   80     0% (   68789)    0% (       1)
tTelnetd     telnetd     ffbc530   89     0% (       0)    0% (       0)
tRlogind     rlogind     1507810   89     0% (       0)    0% (       0)
tRshd        rshd        150a220   89     0% (       0)    0% (       0)
sshd         sshdServer  2753d60   89     0% (      94)    0% (       0)
tPosMsgRx    posSmMsgRe  160a230   90     0% (       0)    0% (       0)
tSioMsgRx    smMsgRecei  160f440   90     0% (       0)    0% (       0)
tSnmpd                   173c9e8   90     0% (     857)    0% (       1)
tShell       start_cli   2783d68   90     0% (      20)    2% (      11)
tShell       start_cli   27a1f78   90     0% (       0)    0% (       0)
CardType     chCardInse  2674b48   95     0% (     126)    0% (       0)
tTffsPTask   flPollTask  143ec98  100     0% (     959)    0% (       0)
tChassis     chTask      26130c8  100     0% (     130)    0% (       0)
smltMaster   smltMaster  29079c0  100     0% (     463)    0% (       0)
tTdpTimer    tdpTimerHa  266f0f0  102     0% (     997)    0% (       0)
tSpfTimer    rcOspfSpfT  2679d58  102     0% (      58)    0% (       0)
tBgpTask     bgpWorkDis  267ef68  102     0% (     128)    0% (       0)
tMainTask    cppMainTas  1dfe508  103     0% (   76207)    2% (      14)
WsmPreConfig wsmPreConf  2829af8  106     0% (       0)    0% (       0)
tIMCStartup  IMCStartup  28158e8  110     0% (       0)    0% (       0)
tLoggerTask  loggerTask  1536c98  112     0% (      12)    0% (       0)
tNtpInterval ntpInterva  2765b58  112     0% (      14)    0% (       0)
tTrapd       trapHandle  1fac898  115     0% (       0)    0% (       0)
tWebSrv      WC_START_W  26cd568  150     0% (      55)    0% (       0)
Http0        SOCKET_Per  28b46a8  150     0% (       0)    0% (       0)
Http1        SOCKET_Per  28bd170  150     0% (       0)    0% (       0)
Http2        SOCKET_Per  28c5c38  150     0% (       0)    0% (       0)
Http3        SOCKET_Per  28ce700  150     0% (       0)    0% (       0)
Http4        SOCKET_Per  28d71c8  150     0% (       0)    0% (       0)
Http5        SOCKET_Per  28dfc90  150     0% (       0)    0% (       0)
Http6        SOCKET_Per  28e8758  150     0% (       0)    0% (       0)
Http7        SOCKET_Per  28f1220  150     0% (       0)    0% (       0)
Http8        SOCKET_Per  28f9ce8  150     0% (       0)    0% (       0)
Http9        SOCKET_Per  29027b0  150     0% (       0)    0% (       0)
tDcacheUpd   dcacheUpd   ffb8d80  175     0% (     797)    0% (       0)
rarMacTask   rarDelMacT  19c4a80  175     0% (    2702)    0% (       0)
tSnmpTmr                 17287d8  200     0% (       0)    0% (       0)
KERNEL                                    0% (    5939)    0% (       0)
INTERRUPT                                 0% (    1669)    0% (       0)
IDLE                                     97% ( 8375627)   94% (     516)
TOTAL                                    97% ( 8547310)   98% (     547)
```

Most interesting for determination of problems is the delta during a time of high CPU load. As this delta is measured since last spyReport (or if never used on that device - since last reboot) the value is an average between the two runs of the spyReport. Therefore the ideal collection of data would be to run the report shortly before and immediately after a high CPU spike. If there is no known interval of high CPU load continuous polling of the spyReport is necessary. This can be achieved through a script on the terminal application.

Note: it is not recommended to run the spyReport more often than every 10 seconds as the process consumes additional CPU cycles.

Note: Running a spyReport resets the MIB value for the CPU load to zero which could lead to misinterpretation of the MIB polling results.

For the interpretation of the result the delta percentage of the IDLE process (though idle is not really a process) is verified first. This will show the percentage of the CPU not working on any of the other processes above. If the IDLE percentage is low the CPU load is high. In this case the single processes can be seen to be consuming the highest CPU percentage.

An example for a single process taking an unexpected high amount of CPU cycles:

```
tSnmpd                  173c9e8   90    67% (  824168)   67% (  824168)
```

Selected processes from the spyReport:

Some processes can be easily identified by name and will immediately indicate the source of high CPU load if the percentage is increasing.

```
tSpfTimer    rcOspfSpfT          OSPF routing table calculation
tBgpTask     bgpWorkDis          BGP routing updates
tTftpdTask   tftpdTask           TFTP file transfer
tFtpdTask    ftpdTask            FTP file transfer
tSnmpd                           Polling of MIB variables
```

Other tasks are less obvious but have often seen to be a high CPU source

```
tMainTask    cppMainTas
tChasServ    chServiceT
tShell       start_cli
tNetTask     netTask
rarMacTask    rarDelMacT
```

tMainTask/cppMainTas

```
tMainTask    cppMainTas  3d944d0  103    12% (  881369)   73% (     193)
```

tMainTask/cppMainTas is the main task to process packets on the CPU itself. When finding a high percentage of delta CPU ticks the CPU is receiving or transmitting a large number of network control or management packets (see packet type list earlier in that document). Typically the buffer utilization of the CPU (show sys performance) increases during that time as the CPU needs to hold a certain amount of packets before having all processed.
The main sources of packets are broadcast or multicast loops, a high number of ARP requests/responses, BPDU loops or IST message exchange.
In all cases the next step is to perform a CPU trace to identify the packets coming into the CPU and disable extensive SNMP polling from management stations.

tChasServ/chServiceT

```
tChasServ    chServiceT  45da518   80     3% (  212402)   20% (      20)
```

tChasServ/chServiceT is a task responsible for polling the hardware of the switch such as modules, fans, temperature, power supplies etc. It will identify all modules be invoked when hardware changes and when the configuration of the switch is read or saved. Complete polling of the hardware happens by default all 20 sec when a Device Manager session is connected to an 8600.
On fully equipped chassis polling of several device managers may result in a 15% load of the CPU in itself.

When experiencing high load in this situation the first action would be to disable polling from all management stations and measure again.

There have been some issues in the past where the polling failed to read certain EEPROMs from different parts in the chassis (e.g. the power supply). Then the CPU kept trying reading these values congesting the management bus and running into high CPU loads. This problem has been resolved in software.

tShell/start_cli

```
tShell       start_cli   24a8a50    2     0% (     182)   15% (      25)
```

Sometimes additional load is generated by heavy CLI work such as querying the config file or certain tables continuously.

rarMacTask/rarDelMacT

```
rarMacTask   rarDelMacT  395aa48  175     1% (  129691)   22% (      37)
```

rarMacTask/rarDelMacT is a task responsible for refreshing the MAC address table and aging the MAC entries. In large networks this task is consuming a visible amount of CPU cycles when running at 6 minute intervals.

tNetTask/netTask

```
tNetTask     netTask     16364f8   50     0% (   35709)   19% (      41)
```

tNetTask/netTask is the network stack of the CPU itself which is responsible for building packets and sending these mainly onto the Out-of-band management port located on the CPU blade. Heavy traffic on the management port may lead to CPU load caused by this task.

tAruTest/arConsiste

This task is used to perform ARU test for all I/O modules. It will likely boost the CPU load to 100% though all tasks will perform correctly. The scheduler will pass the CPU to all other processes if needed so this process will only take the remaining idle time.
The ARU test has to be manually enabled with the command ‘test artable’ and manually disabled otherwise it will run forever.
An indication for a running test is the ‘show test artable’ which will return the following output:

```
Running an ar table test...
Current test results:
  IfIndex: 0
  Result: inProgress
  PassCount: 12549953
  FailCount: 0
```

To stop the ARU testing use the command ‘test stop artable’ and verify the results.

```
Currently no test is running.
Last test results:
  IfIndex: 0
```

```
Result: success
PassCount: 14690268
FailCount: 0
```