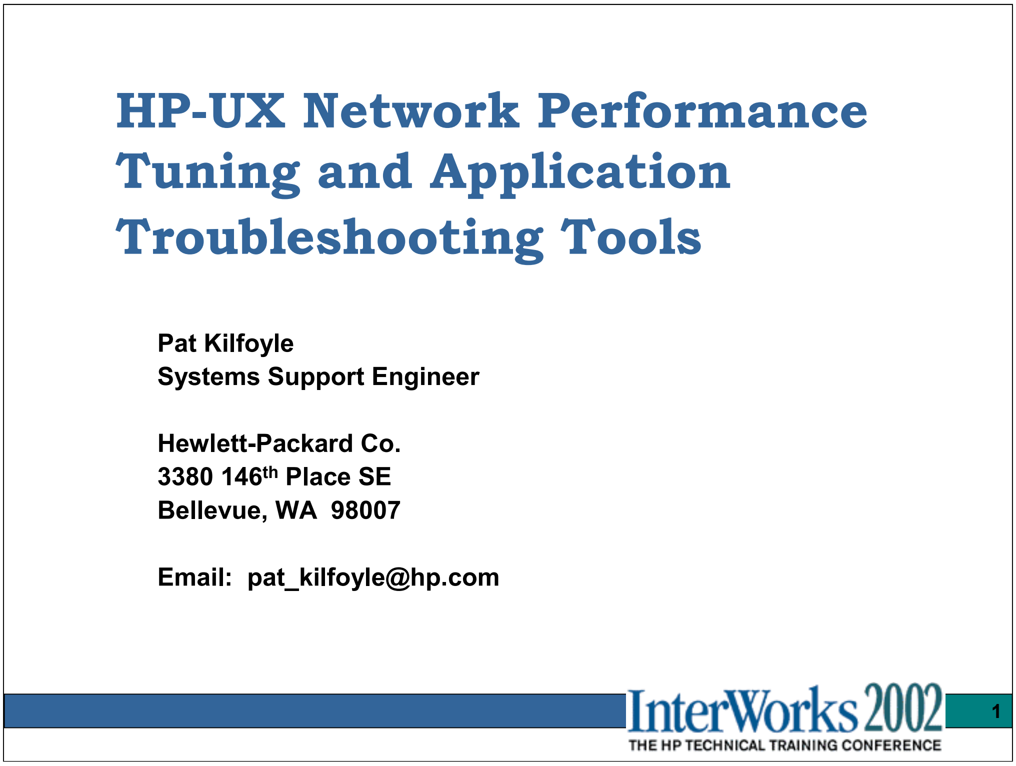


HP-UX Network Performance Tuning and Application Troubleshooting Tools

Pat Kilfoyle
Systems Support Engineer

Hewlett-Packard Co.
3380 146th Place SE
Bellevue, WA 98007

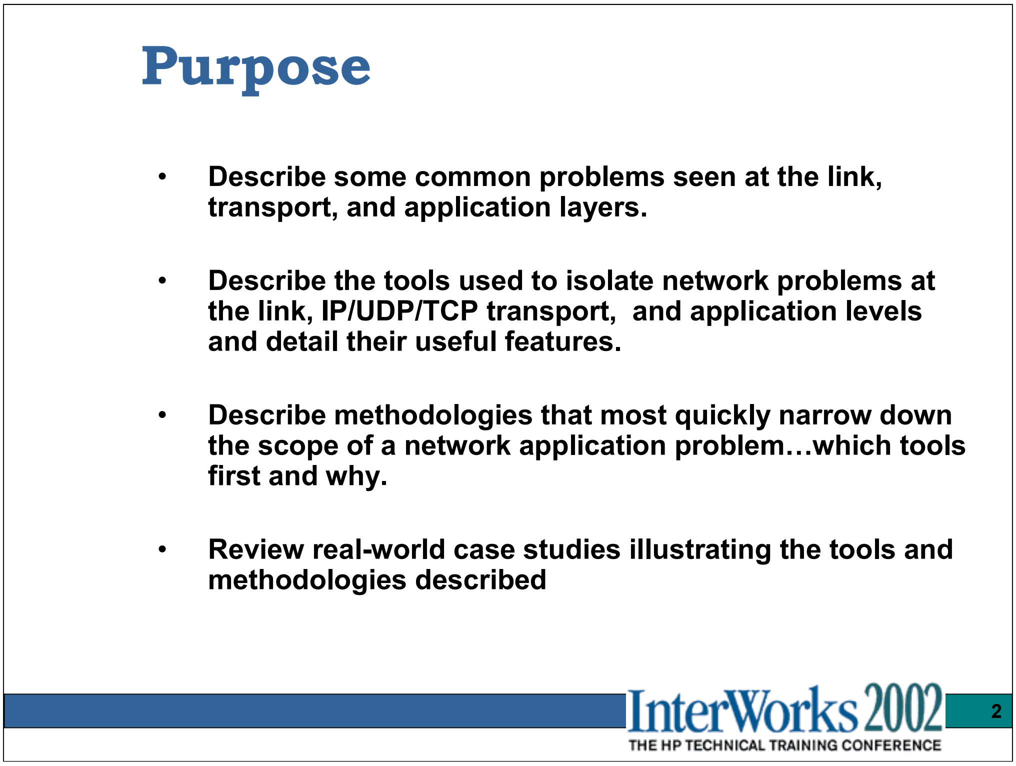
Email: pat_kilfoyle@hp.com

InterWorks 2002
THE HP TECHNICAL TRAINING CONFERENCE

Presented by:

Patrick Kilfoyle, Systems Support Engineer   pat_kilfoyle@hp.com

Hewlett-Packard Co.
3380 146th Place SE
Bellevue, WA 98007



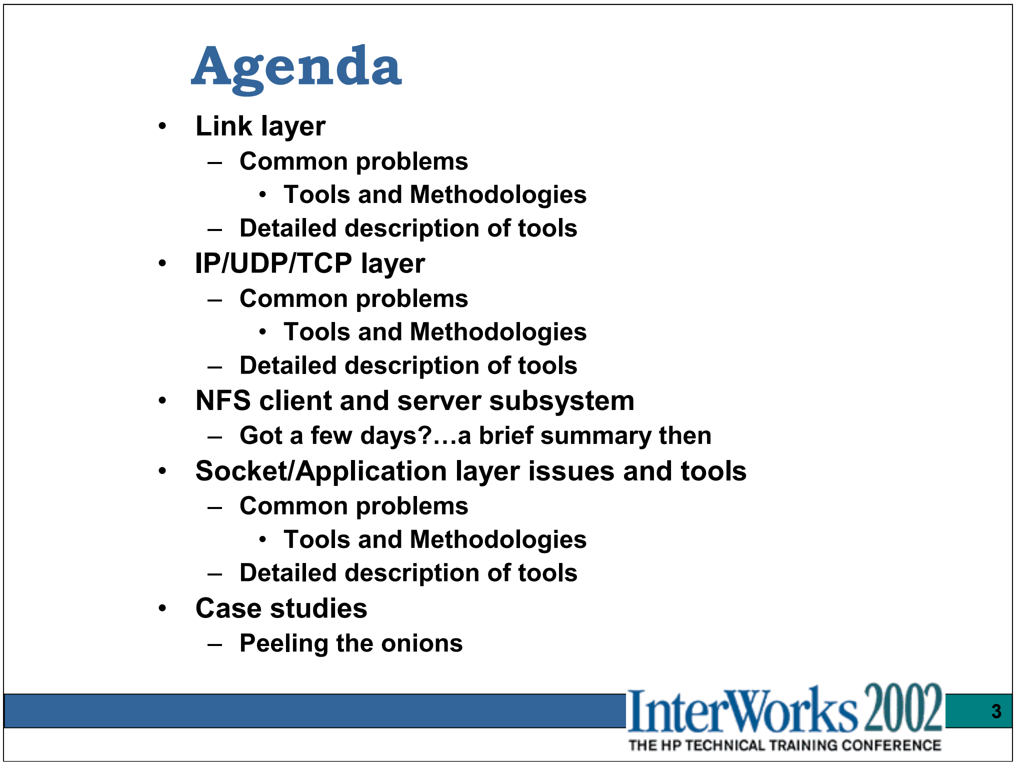
Purpose

- Describe some common problems seen at the link, transport, and application layers.
- Describe the tools used to isolate network problems at the link, IP/UDP/TCP transport, and application levels and detail their useful features.
- Describe methodologies that most quickly narrow down the scope of a network application problem…which tools first and why.
- Review real-world case studies illustrating the tools and methodologies described

The discussions will be limited to UDP/TCP/IP protocols over Ethernet (10/100/1000 BT/SX) network topologies, with some limited mention of FDDI.

The methodologies used within the HP Crisis Management Team are “Emergency room triage” approaches to isolate and stabilize an escalated site. The fine tuning is left as an exercise for the consultants.

Many of the tools mentioned have features that lend themselves to problem isolation in many different layers. A detailed description of the tools and features will help the user understand which tools are of value in various situations.



Agenda

- Link layer
  - Common problems
    - Tools and Methodologies
  - Detailed description of tools
- IP/UDP/TCP layer
  - Common problems
    - Tools and Methodologies
  - Detailed description of tools
- NFS client and server subsystem
  - Got a few days?…a brief summary then
- Socket/Application layer issues and tools
  - Common problems
    - Tools and Methodologies
  - Detailed description of tools
- Case studies
  - Peeling the onions

Link layer tools:

lanscan

lanadmin

linkloop

ndd

analyzers

nettl tracing and logging

switch statistics

network topology maps

* Case studies will include many of these tools and the investigations will lead through some or all three of these different layers.

IP/UDP/TCP layer tools:

netstat

arp

nettl tracing and logging

ftp, rcp, ttcp, netperf

ndd

q4

sample socket code in /usr/lib/demos/networking/socket

Socket/Application level tools:

ttcp, netperf

tusc

lsof

ndd

glance

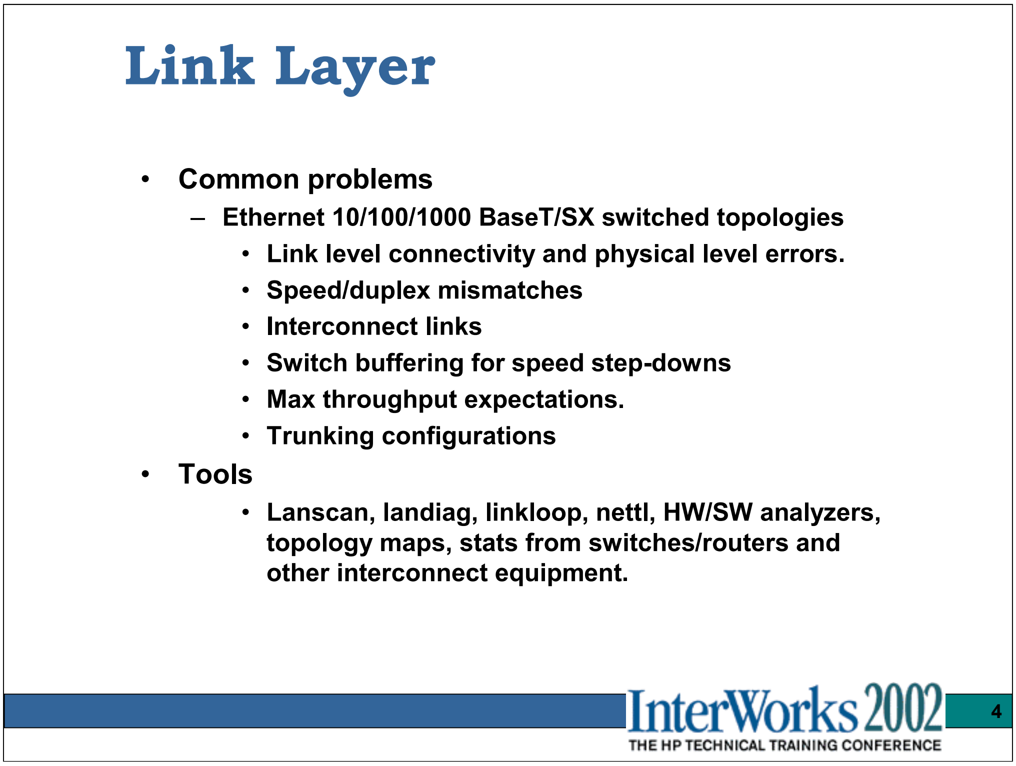
nfsstat

q4

nslookup, dig

chatr, nm, strings, what

sample socket code in /usr/lib/demos/networking/socket



Link Layer

- Common problems
  - Ethernet 10/100/1000 BaseT/SX switched topologies
    - Link level connectivity and physical level errors.
    - Speed/duplex mismatches
    - Interconnect links
    - Switch buffering for speed step-downs
    - Max throughput expectations.
    - Trunking configurations
- Tools
    - Lanscan, landiag, linkloop, nettl, HW/SW analyzers, topology maps, stats from switches/routers and other interconnect equipment.

Speed mismatches are obvious since nothing gets through. Duplex mismatches are configuration issues.

Duplex mismatches can be unnoticed at low throughput rates.

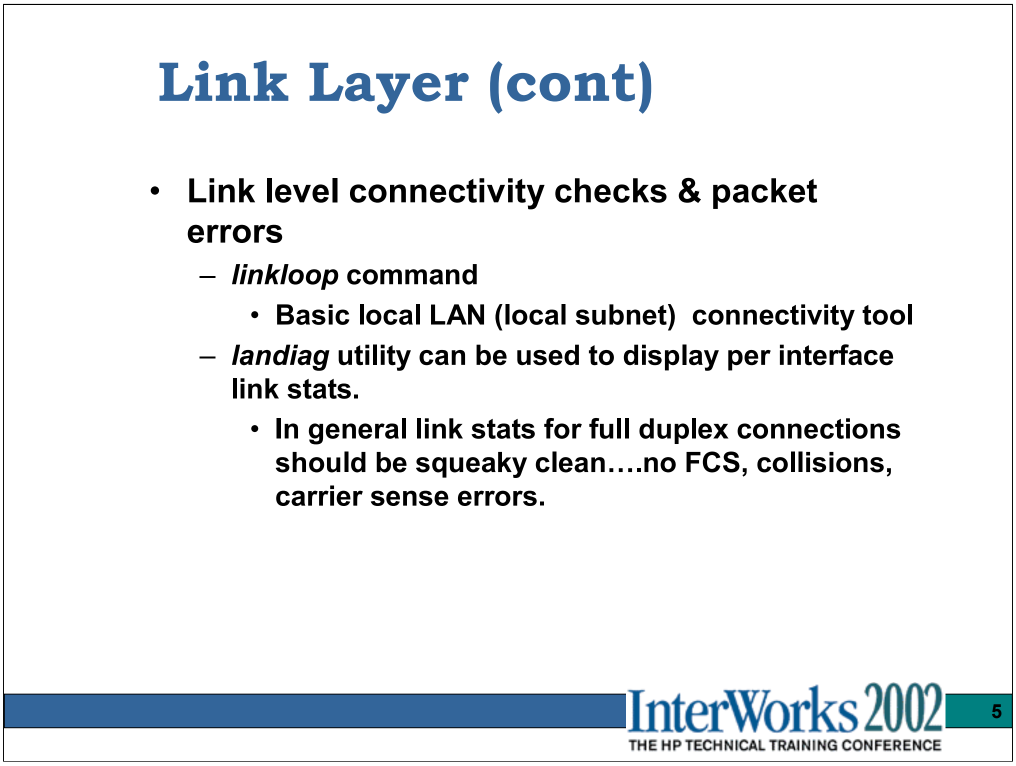
check system and switch config settings…they must match

either autoconfig or nailed. Most sites prefer to nail them.

Switch interconnect links at 100BT with individual switch nodes having 1000BT/SX

DMA rates in and out of cards are not always full link bandwidth….know what to expect.

Throughput expectations with trunking depend highly on the load balancing algorithm in use, and the connection mix being presented by the system.



Link Layer (cont)

- **Link level connectivity checks & packet errors**
  - ***linkloop* command**
    - **Basic local LAN (local subnet) connectivity tool**
  - ***landiag* utility can be used to display per interface link stats.**
    - **In general link stats for full duplex connections should be squeaky clean….no FCS, collisions, carrier sense errors.**

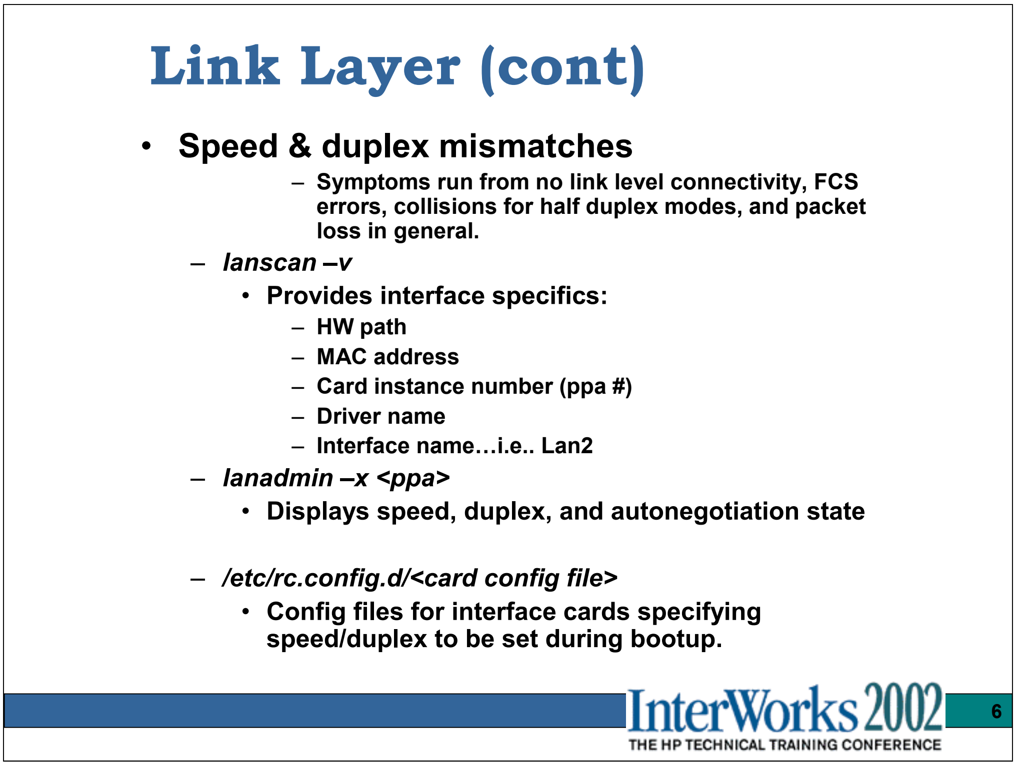
The ***linkloop*** command uses IEEE 802.2 link-level test frames to check connectivity within a local area network.. It is independent of the Streams based transport, sending its data through a direct DLPI connection to the interface card. If IP level connectivity fails, start here.

linkloop –i <instance# of local interface to use> <target MAC>

```
# linkloop -i 0 0x08000962d46e
Link connectivity to LAN station: 0x08000962d46e
 -- OK
```

To obtain the MAC addresses for the interfaces on a system use the ***lanscan*** command:

```
# lanscan
Hardware Station        Crd Hdw   Net-Interface  NM  MAC       HP-DLPI DLPI
Path     Address        In# State NamePPA        ID  Type      Support Mjr#
0/0/0/0  0x00306E0625F4 0   UP    lan0 snap0     1   ETHER     Yes     119
0/2/0/0  0x00306E036EF4 1   UP    lan1 snap1     2   ETHER     Yes     119
```



Link Layer (cont)

- **Speed & duplex mismatches**
    - **Symptoms run from no link level connectivity, FCS errors, collisions for half duplex modes, and packet loss in general.**
  - ***lanscan –v***
    - **Provides interface specifics:**
      - **HW path**
      - **MAC address**
      - **Card instance number (ppa #)**
      - **Driver name**
      - **Interface name…i.e.. Lan2**
  - ***lanadmin –x <ppa>***
    - **Displays speed, duplex, and autonegotiation state**
  - ***/etc/rc.config.d/<card config file>***
    - **Config files for interface cards specifying speed/duplex to be set during bootup.**

Use ***lanscan –v*** to get a quick snapshot of the cards installed in the system (not necessarily in use, but at least installed). Then use the ppa or card instance number in subsequent commands such as ***lanadmin***. The name of the config file in /etc/rc.config.d that controls a specific flavor of interface card, is not always obvious. For reference:

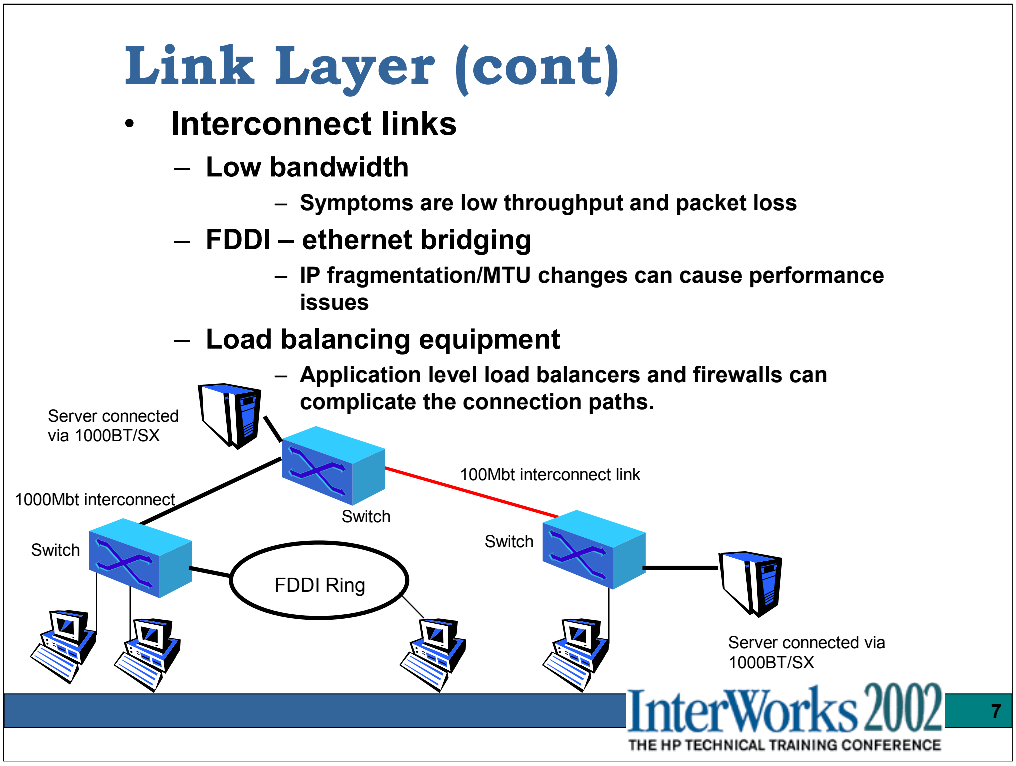
Variables

xxx_INTERFACE_NAME : Name of interface (lan0,lan1...)

xxx_SPEED : set the card speed. Values are : 10HD, 10FD,100HD, 100FD, auto_on

Driver name	**Configuration File**
btlan	hpbtlanconf
btlan0	hpeisabtconf
btlan1	hpbasetconf
btlan3	hpbase100conf
btlan4	hpgsc100conf
btlan5	hppci100conf
btlan6	hpsppci100conf
igelan	hpigelanconf
gelan	hpgelanconf

After the system startup scripts in /sbin/init.d have run, the ***lanadmin –x<ppa>*** output should match the config file setting AND the switch port we are connected to.



Link Layer (cont)

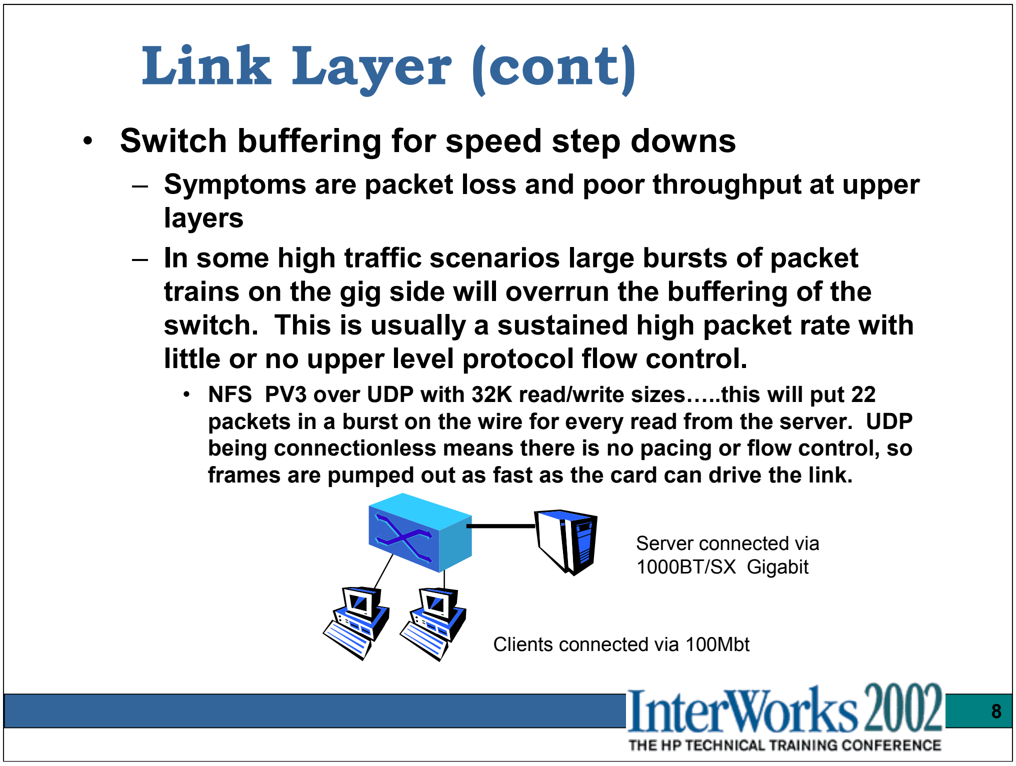
- Interconnect links
  - Low bandwidth
    - Symptoms are low throughput and packet loss
  - FDDI – ethernet bridging
    - IP fragmentation/MTU changes can cause performance issues
  - Load balancing equipment
    - Application level load balancers and firewalls can complicate the connection paths.

A good network topology map is the best place to start in understanding what paths two nodes will be using through the network. If they are on the same subnet, then this switch topology is less obvious. In an environment with a lot of change/growth/moves/additions, a temporary interconnect can go unnoticed as long the basic connectivity is there.

The switch statistics may show some buffer overruns and packet drops. In a pure switched, same subnet, non WAN environment, there should be no packet loss, retransmissions etc.

Where FDDI bridging occurs, the need for IP fragmentation inside the switch needs to be understood. Some switches do not perform IP fragmentation, so the MTU sizes used by the nodes on the FDDI ring need to be managed.

Load balancing equipment and firewalls can have configurable parameters that terminate (non-gracefully) connections and dictate path and server loads. Care needs to be taken to understand the load balancing scheme in use for connection setup, tear down and for idle connections.



Link Layer (cont)

- Switch buffering for speed step downs
  - Symptoms are packet loss and poor throughput at upper layers
  - In some high traffic scenarios large bursts of packet trains on the gig side will overrun the buffering of the switch. This is usually a sustained high packet rate with little or no upper level protocol flow control.
    - NFS PV3 over UDP with 32K read/write sizes…..this will put 22 packets in a burst on the wire for every read from the server. UDP being connectionless means there is no pacing or flow control, so frames are pumped out as fast as the card can drive the link.

Server connected via
1000BT/SX Gigabit

Clients connected via 100Mbt

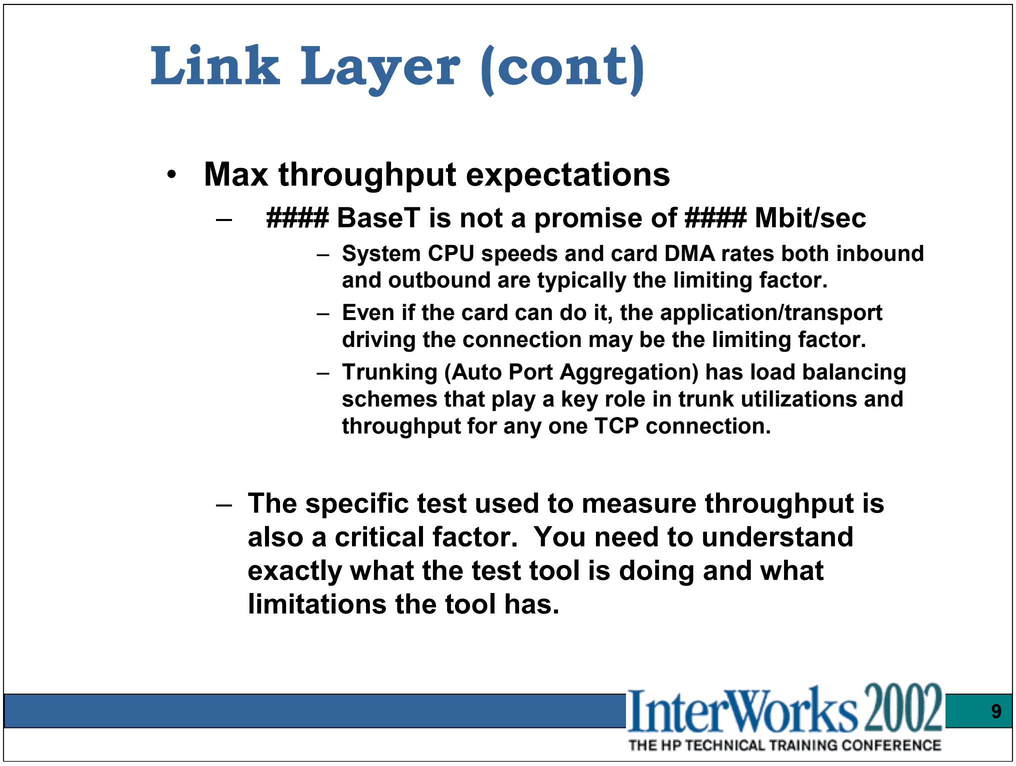
Packet loss due to buffering limits is tough to prove. Typically an analyzer on the gig side shows full utilization and no loss, but the frames will not show up on the 100bt side. Dual analyzer traces or SW network tracing is required to prove it is really the switch. Of course we expect the switch vendor to be honest and accurate in its statistical reporting of buffer overrun/packet drops.

While the buffering is being stressed it can of course affect different ports and connections depending on the buffer management algorithms in use by the switch. Look for TCP retransmissions on the sending side, and NFS client side retransmissions and timeout using the ***nfsstat*** command. Also if NFS is part of the environment, the IP level stats on the HP systems might show “fragments dropped after timeout” when ***netstat –sp ip*** is run.

```
ip:
        124918984 total packets received
        0 bad IP headers
        2237756 fragments received
        0 fragments dropped (dup or out of space)
        18 fragments dropped after timeout
```

Key data:

Switch buffer statistics, client and server side transport stats showing retransmissions



Link Layer (cont)

- Max throughput expectations
  - #### BaseT is not a promise of #### Mbit/sec
    - System CPU speeds and card DMA rates both inbound and outbound are typically the limiting factor.
    - Even if the card can do it, the application/transport driving the connection may be the limiting factor.
    - Trunking (Auto Port Aggregation) has load balancing schemes that play a key role in trunk utilizations and throughput for any one TCP connection.
  - The specific test used to measure throughput is also a critical factor. You need to understand exactly what the test tool is doing and what limitations the tool has.

The tools to measure throughput are typically UDP/TCP/IP based tools and vary greatly in their accuracy. Knowing the limitations of each tool in measuring throughput is important.

Quick and dirty, but course measurements of throughput:

ftp – file system and buffercache usage will influence results as will socket buffer sizing. (64k max)

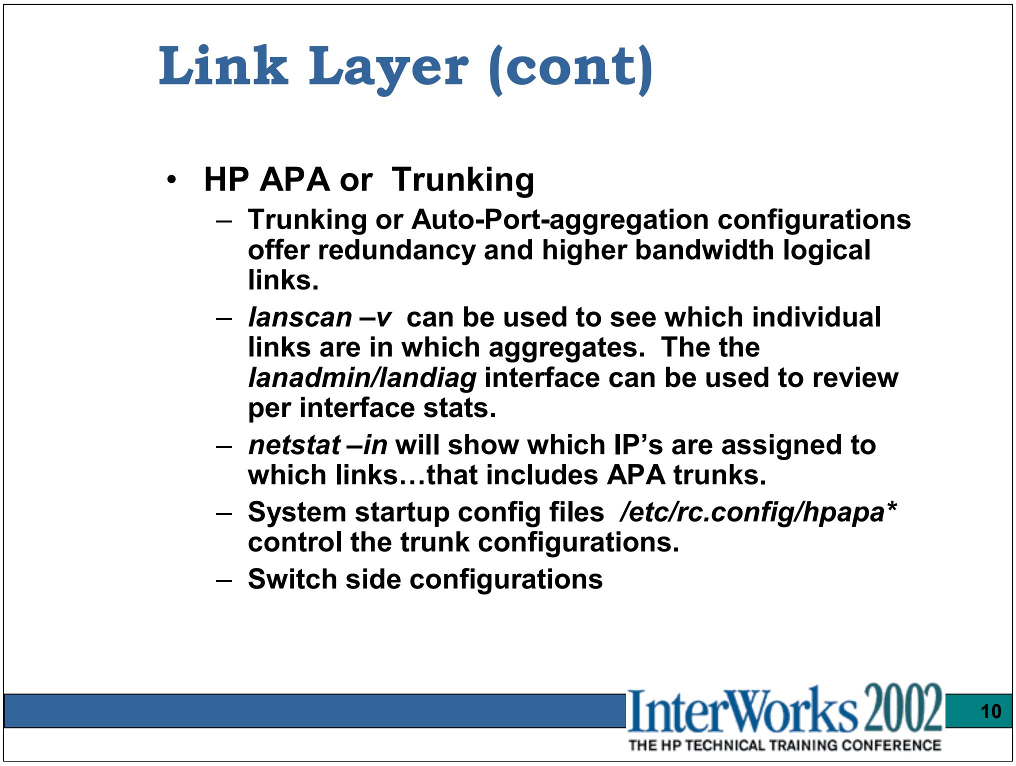
rcp – file system and buffercache usage will influence results. Socket buffer settings need to be tuned for the test.

nfs copy – influenced by local file system, buffercache, biod process load, mount type, mount protocol, and other processes using NFS….and that's just the client side. There are a similar list of issue on the server.

More precise tools:

ttcp – "Test TCP" a simple sockets-based test tool that allows you to specify TCP or UDP transfers to various port number. It does not use disc buffercache so is a better test of pure network throughput, has tunable socket buffer settings and data size settings. It does the throughput calculations for you.

netperf – a benchmarking tool that can be used to measure the performance of many different types of networking. It provides tests for both unidirectional throughput, and end-to-end latency. Primary performance tool of HP unix network labs.



Link Layer (cont)

- HP APA or Trunking
  - Trunking or Auto-Port-aggregation configurations offer redundancy and higher bandwidth logical links.
  - *lanscan –v* can be used to see which individual links are in which aggregates. The the *lanadmin/landiag* interface can be used to review per interface stats.
  - *netstat –in* will show which IP's are assigned to which links…that includes APA trunks.
  - System startup config files */etc/rc.config/hpapa** control the trunk configurations.
  - Switch side configurations

HP's Trunking product is called APA for Auto Port Aggregation and is compatible with Cisco Fast EtherChannel (FEC) and the IEEE 802.3ad Link Aggregation Control Protocol (LACP) standards. Typically one or more like-type links (FastEther or Gigabit) are grouped into a trunk and appear to the system as one logical interface card. IP address(es) can then be assigned like any other interface.

To see which individual links are grouped into trunks, use the ***lanscan –v*** command.

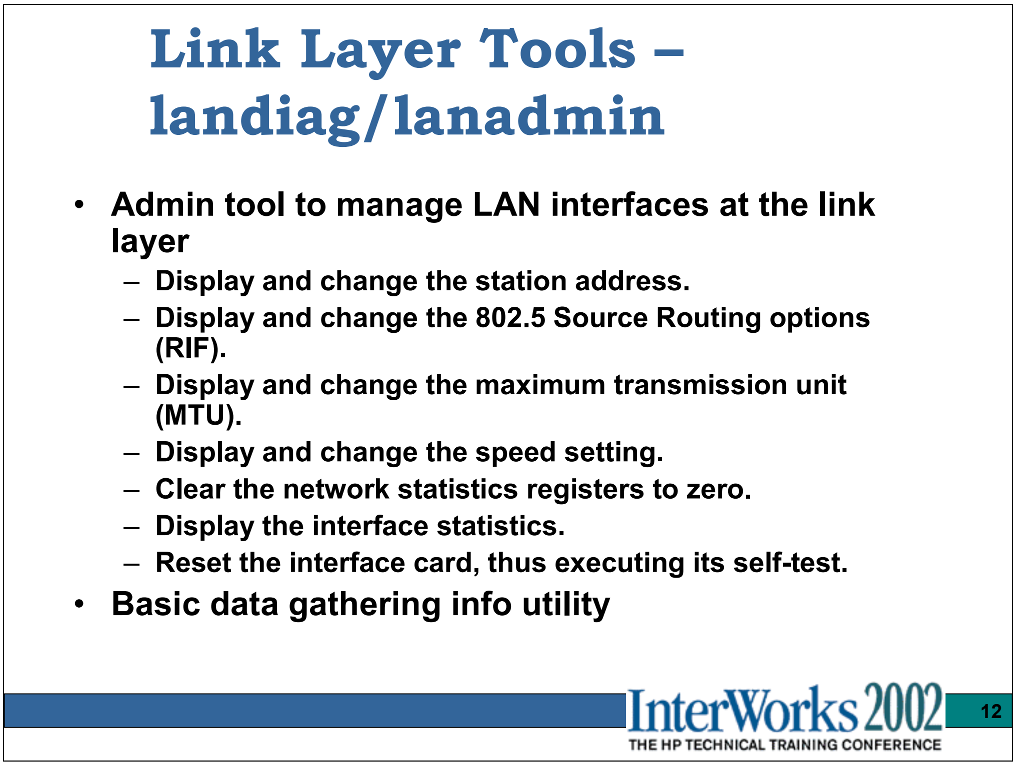
There are different load balancing algorithms to choose from which will ultimately determine which connections use which link inside the trunk, and the efficient use of the trunk. Most switch environments use the MAC source, destination, or a combination of the MAC source and destination address to distribute the traffic. Many layer 3 switches also support distributions algorithms based on the IP address of a packet contained in a frame. Whether the algorithm uses the MAC address or IP address the concept of the address-based distribution is the same. Often, these methods for load balancing share **one important limitation**—they are static. They generally neither adjust to reflect traffic volume through the individual links, nor do they evaluate an individual conversation to determine which link would be best at a given moment. Instead, the selected algorithm distributes the conversations across the links with the expectation that statistically, *with multiple conversations*, the load will be balanced.

Link Layer Tools - lanscan

- **Displays information about LAN interfaces installed that the system SW supports/recognizes**
- **Typical usage:**

  lanscan –v* | *more or simply ***lanscan***
- **Key fields:**
  - **HW IO path for card**
  - **HW MAC address**
  - **Instance # or 'PPA #'**
  - **Netname…ie. lan1 etc.**
  - **Driver name for this card**
  - **APA port assignment**

```
# lanscan
Hardware Station        Crd Hdw   Net-Interface  NM  MAC       HP-DLPI DLPI
Path     Address        In# State NamePPA        ID  Type      Support Mjr#
0/0/0/0  0x00306E0625F4 0   UP    lan0 snap0     1   ETHER     Yes     119
0/2/0/0  0x00306E036EF4 1   UP    lan1 snap1     2   ETHER     Yes     119

# lanscan -v
-------------------------------------------------------------------------------
Hardware Station        Crd Hdw   Net-Interface  NM  MAC       HP-DLPI DLPI
Path     Address        In# State NamePPA        ID  Type      Support Mjr#
0/0/0/0  0x00306E0625F4 0   UP    lan0 snap0     1   ETHER     Yes     119

Extended Station                           LLC Encapsulation
Address                                    Methods
0x00306E0625F4                             IEEE HPEXTIEEE SNAP ETHER NOVELL

Driver Specific Information
btlan
-------------------------------------------------------------------------------
Hardware Station        Crd Hdw   Net-Interface  NM  MAC       HP-DLPI DLPI
Path     Address        In# State NamePPA        ID  Type      Support Mjr#
0/2/0/0  0x00306E036EF4 1   UP    lan1 snap1     2   ETHER     Yes     119

Extended Station                           LLC Encapsulation
Address                                    Methods
0x00306E036EF4                             IEEE HPEXTIEEE SNAP ETHER NOVELL

Driver Specific Information
gelan
```



Link Layer Tools – landiag/lanadmin

- Admin tool to manage LAN interfaces at the link layer
  - Display and change the station address.
  - Display and change the 802.5 Source Routing options (RIF).
  - Display and change the maximum transmission unit (MTU).
  - Display and change the speed setting.
  - Clear the network statistics registers to zero.
  - Display the interface statistics.
  - Reset the interface card, thus executing its self-test.
- Basic data gathering info utility

InterWorks 2002
THE HP TECHNICAL TRAINING CONFERENCE

The ***lanadmin*** flavor of this utility has command line options to alter MAC addr, MTU, speed, duplex settings.

Typically used to display the link specific configuration settings (speed/duplex/MTU)

and displaying the traffic stats. It is run interactively as follows…I'll spare you the stdout menu's:

landiag -> lan -> ppa 0-> dis

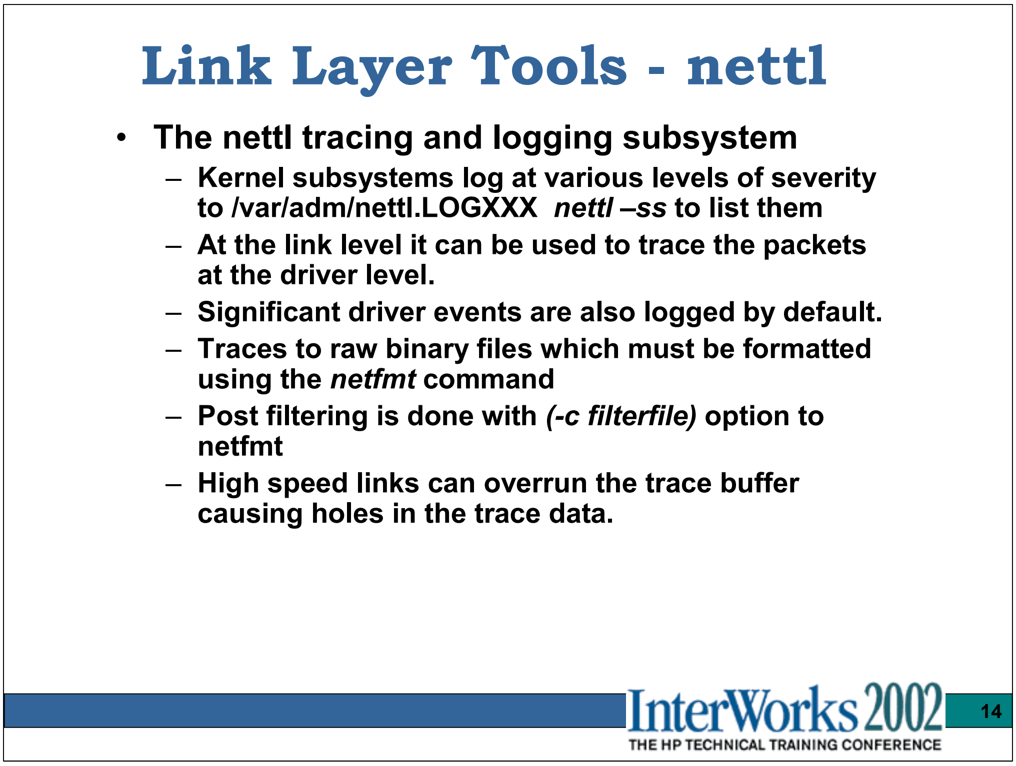
```
              LAN INTERFACE STATUS DISPLAY
PPA Number                      = 0
Description                     = lan0 HP PCI 10/100Base-TX Core [100BASE-TX,FD,
AUTO,TT=1500]
Type (value)                    = ethernet-csmacd(6)
MTU Size                        = 1500
Speed                           = 100000000
Station Address                 = 0x306e0625f4
Administration Status (value)   = up(1)
Operation Status (value)        = up(1)
Last Change                     = 105783429
                                                       Index                         = 1
Inbound Octets                  = 1703017292
Inbound Unicast Packets         = 330723               Alignment Errors              = 0
Inbound Non-Unicast Packets     = 2542161              FCS Errors                    = 0
Inbound Discards                = 0                    Single Collision Frames       = 0
Inbound Errors                  = 0                    Multiple Collision Frames     = 0
Inbound Unknown Protocols       = 894660               Deferred Transmissions        = 0
Outbound Octets                 = 35206501             Late Collisions               = 0
Outbound Unicast Packets        = 119383               Excessive Collisions          = 0
Outbound Non-Unicast Packets    = 1217                 Internal MAC Transmit Errors  = 0
Outbound Discards               = 0                    Carrier Sense Errors          = 0
Outbound Errors                 = 0                    Frames Too Long               = 0
Outbound Queue Length           = 0                    Internal MAC Receive Errors   = 0
Specific                        = 655367
```

Link Layer Tools - linkloop

- **The linkloop command uses IEEE 802.2 link-level test frames to check connectivity within a local area network (LAN).**
  - **A good sanity check of basic link level connectivity when dealing with IP connectivity problems within the same subnet or vlan.**
  - **Uses DLPI to talk to the interface card…no Streams/transport stack involved.**

First obtain the local interfaces instance # (also referred to as the PPA #) via *lanscan* and obtain the remote systems MAC address via ***lanscan*** on that system.

```
linkloop [-i PPA] [-n count] [-r rif] [-s size] [-t timeout]
           [-v] linkaddr ...

# linkloop  -n 4 -s 1400 -i 0  -v 0x08000962d46e
Link connectivity to LAN station: 0x08000962d46e
 -- OK
```



Link Layer Tools - nettl

- **The nettl tracing and logging subsystem**
  - **Kernel subsystems log at various levels of severity to /var/adm/nettl.LOGXXX *nettl –ss* to list them**
  - **At the link level it can be used to trace the packets at the driver level.**
  - **Significant driver events are also logged by default.**
  - **Traces to raw binary files which must be formatted using the *netfmt* command**
  - **Post filtering is done with *(-c filterfile)* option to netfmt**
  - **High speed links can overrun the trace buffer causing holes in the trace data.**

There are many kernel nettl subsystems. Execute ***nettl –ss*** to see them. The types of event logged and traced vary per subsystem. There are 4 levels of 'logging' …Informative, Warning, Error, and Disaster. Error and Disaster levels are enabled by default.

An extensive man page describes the use of nettl and netfmt to format the data. I'll spare you some reading. To enable tracing at the btlan driver level use:

```
nettl –tn pduin pduout –e btlan –tm 90000 –s 1024 –m 256 –f raw
```

or for IP level

```
nettl –tn pduin pduout –e ns_ls_ip –tm 90000 –s 1024 –m 256 –f raw
```

The –tm –m and –s options are important to avoid loosing data on a busy link.

On HPUX 11.0, the output is written to two output files with extensions.TRC00 and .TRC01. The .TRC00 file always has the most recent data and the files wrap. On 11.11+ there can be more than two files using the –n option. You just need lots of disc space.

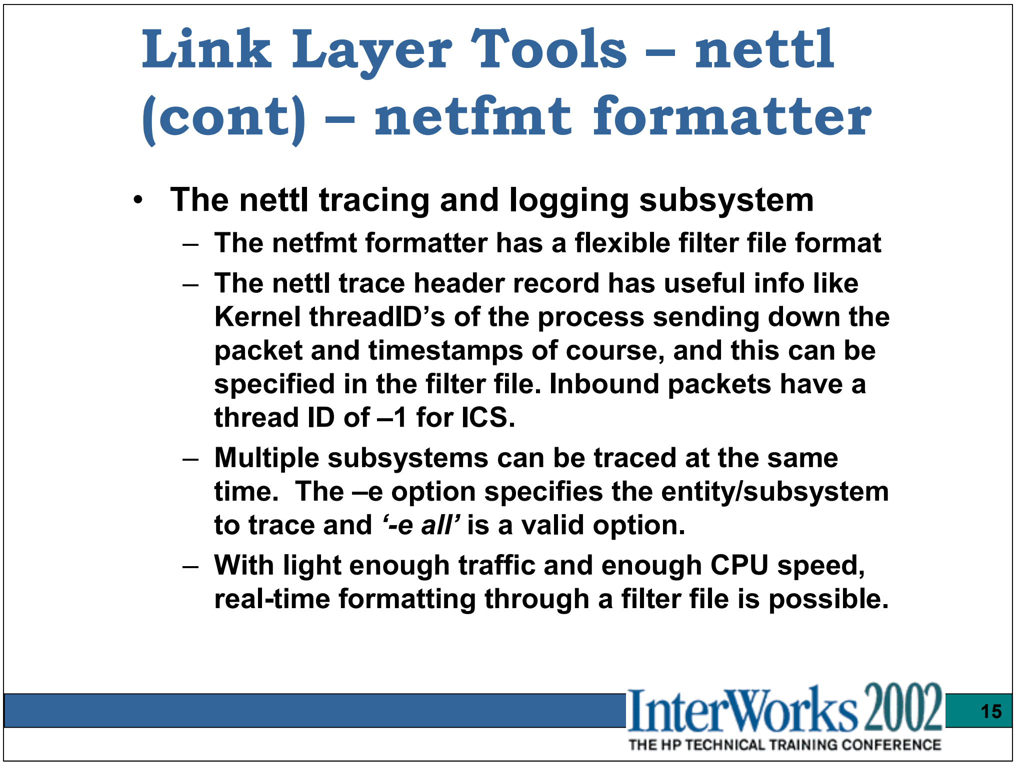
Use netfmt to format the .TRCXX files. The one-line-per-packet command:

```
netfmt –n –c filter –1T –f raw.TRC00 > terse.fmt
```

The fully formatted output is obtained using:

```
netfmt –n –c filter –lN –f raw.TRC00 > nice.fmt
```

***note the 'ell' above is not the number 1 as in the one-liner syntax.



Link Layer Tools – nettl (cont) – netfmt formatter

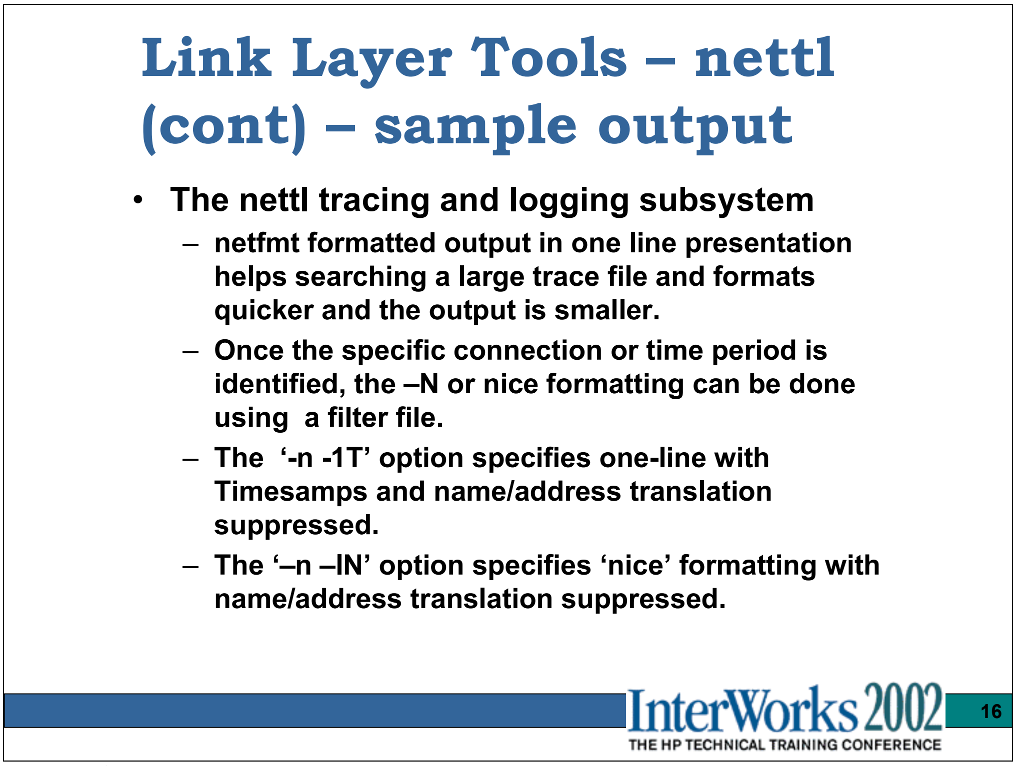
- **The nettl tracing and logging subsystem**
  - **The netfmt formatter has a flexible filter file format**
  - **The nettl trace header record has useful info like Kernel threadID's of the process sending down the packet and timestamps of course, and this can be specified in the filter file. Inbound packets have a thread ID of –1 for ICS.**
  - **Multiple subsystems can be traced at the same time. The –e option specifies the entity/subsystem to trace and '*-e all*' is a valid option.**
  - **With light enough traffic and enough CPU speed, real-time formatting through a filter file is possible.**

To avoid delays I typically use –n option to skip the number-to-name mapping that netfmt tries to do. You get raw IP's and port numbers instead, but it formats much more quickly.

The filter file has a layered approach…as in OSI-like layers, wherein filters at the same layer are OR'ed and AND'ed with subsequent upper layers.

A sample filter file with unused filters commented out:

```
formatter option suppress
formatter option !highlight
formatter mode nice
#filter dest 080009036A04
#filter source 080009036A04
filter ip_saddr 15.43.234.203
filter ip_daddr 15.43.234.203
#filter ip_saddr 15.19.80.72
#filter ip_daddr 15.19.80.72
#filter ip_saddr 127.0.0.1
#filter connection 15.43.225.56:1309 15.56.216.34:7000
#formatter filter time_from 15:11:22.000000 10/11/93
#formatter filter time_through 15:11:23.000000 10/11/93
#formatter filter subsystem NS_LS_LOOPBACK
formatter filter subsystem NS_LS_IP
#filter rpcdirection call
#filter rpcdirection reply
#filter tcp_sport 1201
#filter tcp_dport 53
#filter udp_sport 1200
#filter udp_dport 1201
#formatter filter Process ID 366153
#formatter filter kind pduout
```



Link Layer Tools – nettl (cont) – sample output

- The nettl tracing and logging subsystem
  - netfmt formatted output in one line presentation helps searching a large trace file and formats quicker and the output is smaller.
  - Once the specific connection or time period is identified, the –N or nice formatting can be done using a filter file.
  - The ‘-n -1T’ option specifies one-line with Timesamps and name/address translation suppressed.
  - The ‘–n –lN’ option specifies ‘nice’ formatting with name/address translation suppressed.

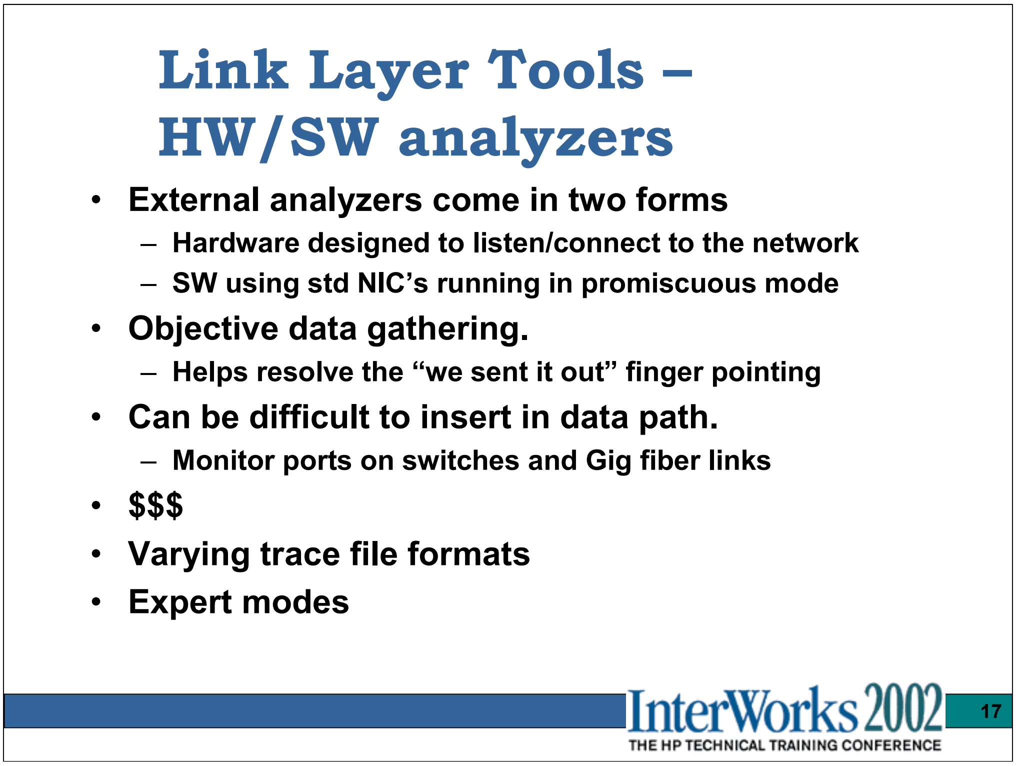
One-line per packet output:

```
07:34:03.083704 8b Unknown DSAP Type: 0xe0
07:34:03.678806 Eb arp request for: 15.24.40.87 from: 15.24.47.253
07:34:04.821804 TCP .23 > .1564: PA 361195ad:361195af(2) ack: 677695 win: 8000 telnet
07:34:04.821822 IP 15.24.46.28.23 > 15.24.46.33.1564: [DF] PA 361195ad:361195af(2) ack: 677695
win: 8000 telnet
07:34:04.821836 Ei 15.24.46.28.23 > 15.24.46.33.1564: [DF] PA 361195ad:361195af(2) ack: 677695
win: 8000 telnet
07:34:05.000810 Ei 15.24.46.33.1564 > 15.24.46.28.23: [DF] A ack: 361195af win: 2218
07:34:05.000820 IP 15.24.46.33.1564 > 15.24.46.28.23: [DF] A 677695:67769b(6) ack: 361195af win:
2218 telnet
```

Nice or fully formatted output:

```
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvARPA/9000 NETWORKINGvvvvvvvvvvvvvvvvvvvvvvvvvv@#%
  Timestamp            : Thu Jul 11 MDT 2002 07:34:04.821822
  Process ID           : 366153             Subsystem        : NS_LS_IP
  User ID ( UID )      : 0                  Trace Kind       : PDU OUT TRACE
  Device ID            : -1                 Path ID          : 0
  Connection ID        : 0
  Location             : 00123
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
================================ IP Header (outbound -- pid: 366153) =========
Source:  15.24.46.28(A) Dest:  15.24.46.33(A)
       len:  42      ttl: 64     proto: 6      cksum:  0x67da      id: 0x5887
     flags:  DF      tos: 0x0  hdrlen: 20     offset:  0x0     optlen: 0
-------------------------------- TCP Header ----------------------------------
sport:   23   -->   dport:  1564     flags: PUSH ACK
       seq: 0x361195ad  urp: 0x0      chksum: 0x494f   data len: 2
       ack: 0x677695    win: 0x8000   optlen: 0
-------------------------------- TELNET --------------------------------------
   0: 23 20 -- -- -- -- -- -- -- -- -- -- -- -- -- --  # ..............
```

Note – the header field labeled ‘Process ID’ is actually the kernel thread ID of the sending thread. The ***Glance*** performance tool has a menu to display the thread ID’s for all running processes.



Link Layer Tools – HW/SW analyzers

- External analyzers come in two forms
  - Hardware designed to listen/connect to the network
  - SW using std NIC's running in promiscuous mode
- Objective data gathering.
  - Helps resolve the "we sent it out" finger pointing
- Can be difficult to insert in data path.
  - Monitor ports on switches and Gig fiber links
- $$$
- Varying trace file formats
- Expert modes

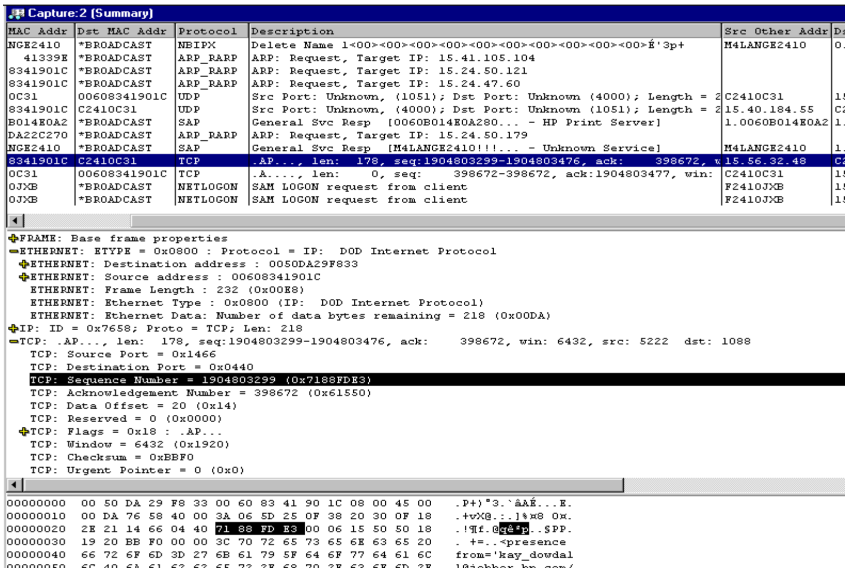
Popular HW analyzers for 100/1000BT/SX:

Network general Sniffer
Agilent 'Network Advisor'
…many more

Popular SW analyzers:

Microsoft Netmon
Network General Netxray
Network Instruments Observer 8.0
Ethereal
….many more

Link Layer Tools – SW analyzers

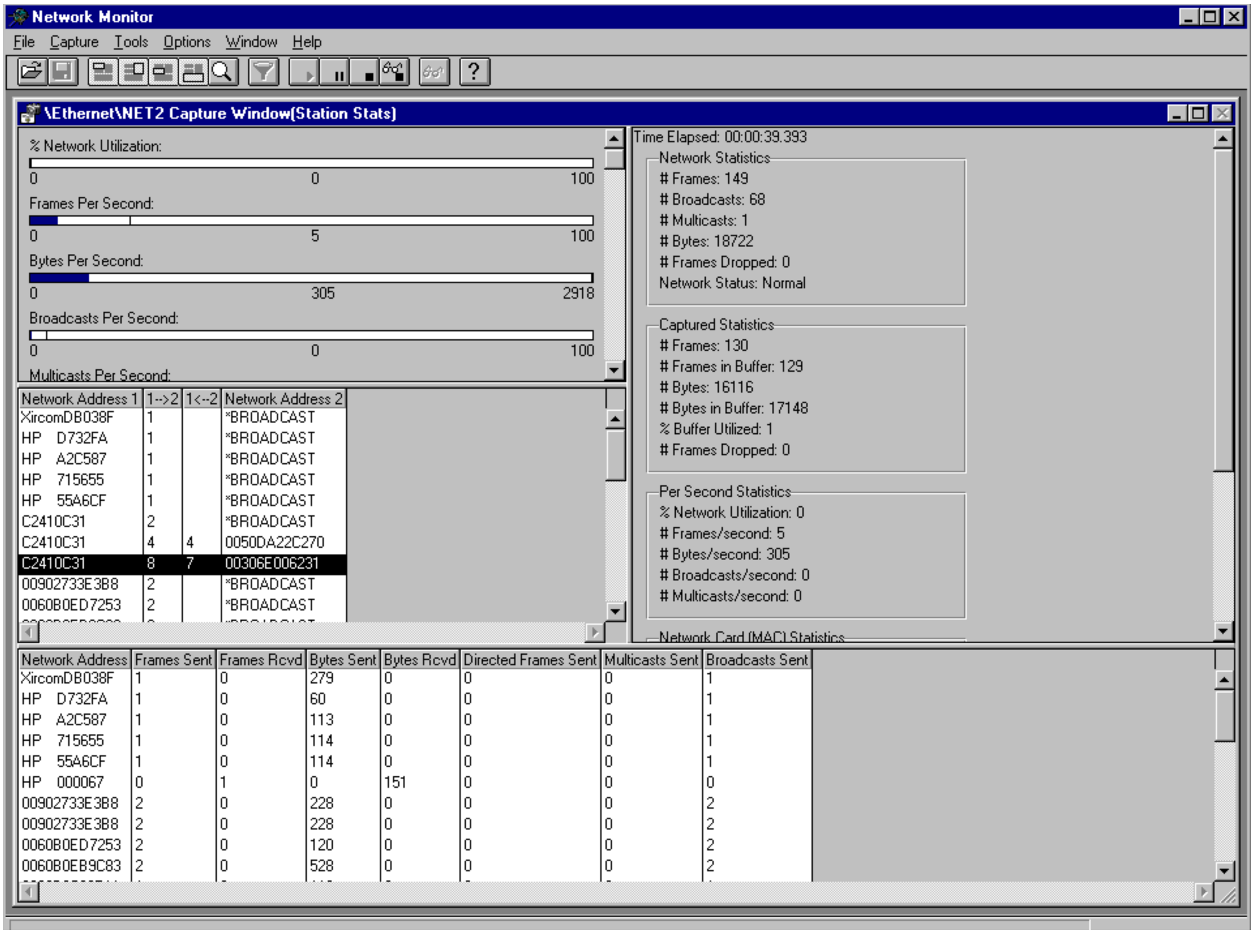


Capture:2 (Summary)

```
MAC Addr  Dst MAC Addr  Protocol  Description                                                         Src Other Addr  Ds
NGE2410   *BROADCAST    NBIPX     Delete Name 1<00><00><00><00><00><00><00><00><00><00>É'3p+          M4LANGE2410     0.
  41339E  *BROADCAST    ARP_RARP  ARP: Request, Target IP: 15.41.105.104
8341901C  *BROADCAST    ARP_RARP  ARP: Request, Target IP: 15.24.50.121
8341901C  *BROADCAST    ARP_RARP  ARP: Request, Target IP: 15.24.47.60
0C31      00608341901C  UDP       Src Port: Unknown, (1051); Dst Port: Unknown (4000); Length = 2     C2410C31        1
8341901C  C2410C31      UDP       Src Port: Unknown, (4000); Dst Port: Unknown (1051); Length = 2     15.40.184.55    C
B014E0A2  *BROADCAST    SAP       General Svc Resp  [0060B014E0A280... - HP Print Server]             1.0060B014E0A2  1.
DA22C270  *BROADCAST    ARP_RARP  ARP: Request, Target IP: 15.24.50.179
NGE2410   *BROADCAST    SAP       General Svc Resp  [M4LANGE2410!!!... - Unknown Service]             M4LANGE2410     1.
8341901C  C2410C31      TCP       .AP..., len:  178, seq:1904803299-1904803476, ack:    398672, w     15.56.32.48     C
0C31      00608341901C  TCP       .A...., len:    0, seq:    398672-398672, ack:1904803477, win:      C2410C31        1
0JXB      *BROADCAST    NETLOGON  SAM LOGON request from client                                       F2410JXB        1
0JXB      *BROADCAST    NETLOGON  SAM LOGON request from client                                       F2410JXB        1
```

```
+FRAME: Base frame properties
-ETHERNET: ETYPE = 0x0800 : Protocol = IP:  DOD Internet Protocol
  +ETHERNET: Destination address : 0050DA29F833
  +ETHERNET: Source address : 00608341901C
    ETHERNET: Frame Length : 232 (0x00E8)
    ETHERNET: Ethernet Type : 0x0800 (IP:  DOD Internet Protocol)
    ETHERNET: Ethernet Data: Number of data bytes remaining = 218 (0x00DA)
+IP: ID = 0x7658; Proto = TCP; Len: 218
-TCP: .AP..., len:  178, seq:1904803299-1904803476, ack:    398672, win: 6432, src: 5222  dst: 1088
    TCP: Source Port = 0x1466
    TCP: Destination Port = 0x0440
    TCP: Sequence Number = 1904803299 (0x7188FDE3)
    TCP: Acknowledgement Number = 398672 (0x61550)
    TCP: Data Offset = 20 (0x14)
    TCP: Reserved = 0 (0x0000)
  +TCP: Flags = 0x18 : .AP...
    TCP: Window = 6432 (0x1920)
    TCP: Checksum = 0xBBF0
    TCP: Urgent Pointer = 0 (0x0)
```

```
00000000  00 50 DA 29 F8 33 00 60 83 41 90 1C 08 00 45 00   .P+)°3.`âAÉ...E.
00000010  00 DA 76 58 40 00 3A 06 5D 25 0F 38 20 30 0F 18   .+vX@.:.]%¤8 0¤.
00000020  2E 21 14 66 04 40 71 88 FD E3 00 06 15 50 50 18   .!¶f.@qê²p..§PP.
00000030  19 20 BB F0 00 00 3C 70 72 65 73 65 6E 63 65 20   . +=..<presence 
00000040  66 72 6F 6D 3D 27 6B 61 79 5F 64 6F 77 64 61 6C   from='kay_dowdal
```



% Network Utilization:
0 0 100

Frames Per Second:
0 5 100

Bytes Per Second:
0 305 2918

Broadcasts Per Second:
0 0 100

Multicasts Per Second:

Time Elapsed: 00:00:39.393

Network Statistics
Frames: 149
Broadcasts: 68
Multicasts: 1
Bytes: 18722
Frames Dropped: 0
Network Status: Normal

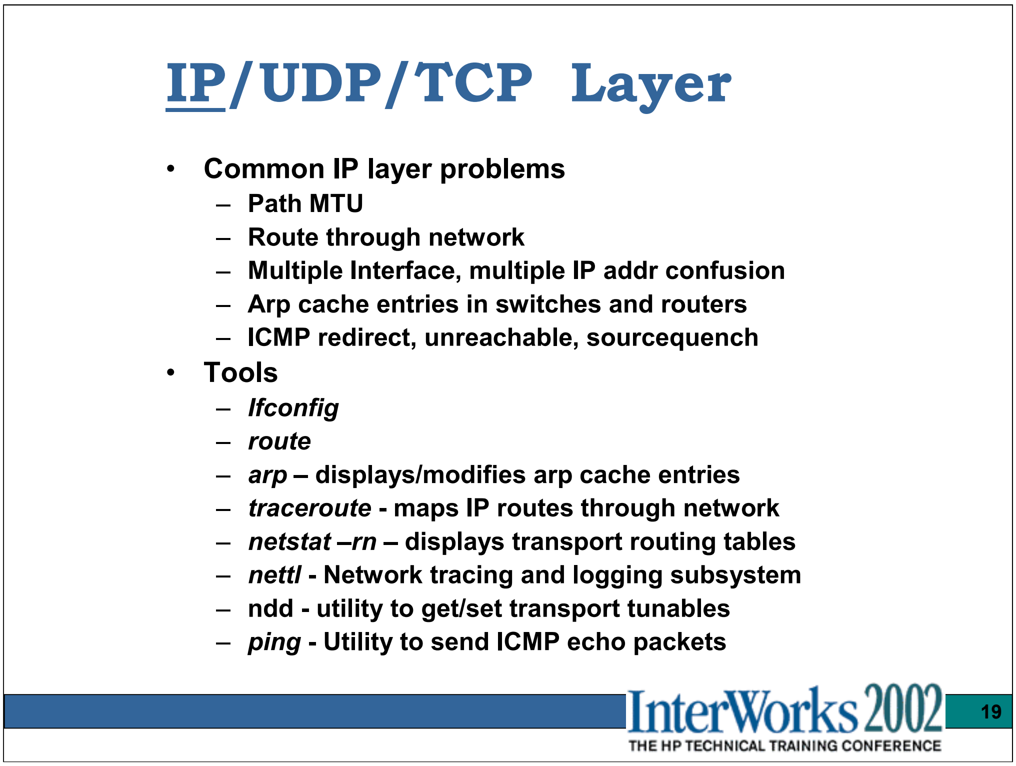
Captured Statistics
Frames: 130
Frames in Buffer: 129
Bytes: 16116
Bytes in Buffer: 17148
% Buffer Utilized: 1
# Frames Dropped: 0

Per Second Statistics
% Network Utilization: 0
# Frames/second: 5
# Bytes/second: 305
# Broadcasts/second: 0
# Multicasts/second: 0

Network Card (MAC) Statistics

Network Address 1	1-->2	1<--2	Network Address 2
XircomDB038F	1		*BROADCAST
HP D732FA	1		*BROADCAST
HP A2C587	1		*BROADCAST
HP 715655	1		*BROADCAST
HP 55A6CF	1		*BROADCAST
C2410C31	2		*BROADCAST
C2410C31	4	4	0050DA22C270
C2410C31	8	7	00306E006231
00902733E3B8	2		*BROADCAST
0060B0ED7253	2		*BROADCAST

Network Address	Frames Sent	Frames Rcvd	Bytes Sent	Bytes Rcvd	Directed Frames Sent	Multicasts Sent	Broadcasts Sent
XircomDB038F	1	0	279	0	0	0	1
HP D732FA	1	0	60	0	0	0	1
HP A2C587	1	0	113	0	0	0	1
HP 715655	1	0	114	0	0	0	1
HP 55A6CF	1	0	114	0	0	0	1
HP 000067	0	1	0	151	0	0	0
00902733E3B8	2	0	228	0	0	0	2
00902733E3B8	2	0	228	0	0	0	2
0060B0ED7253	2	0	120	0	0	0	2
0060B0EB9C83	2	0	528	0	0	0	2



IP/UDP/TCP Layer

- Common IP layer problems
  - Path MTU
  - Route through network
  - Multiple Interface, multiple IP addr confusion
  - Arp cache entries in switches and routers
  - ICMP redirect, unreachable, sourcequench
- Tools
  - *Ifconfig*
  - *route*
  - *arp* – displays/modifies arp cache entries
  - *traceroute* - maps IP routes through network
  - *netstat –rn* – displays transport routing tables
  - *nettl* - Network tracing and logging subsystem
  - ndd - utility to get/set transport tunables
  - *ping* - Utility to send ICMP echo packets

InterWorks2002
THE HP TECHNICAL TRAINING CONFERENCE

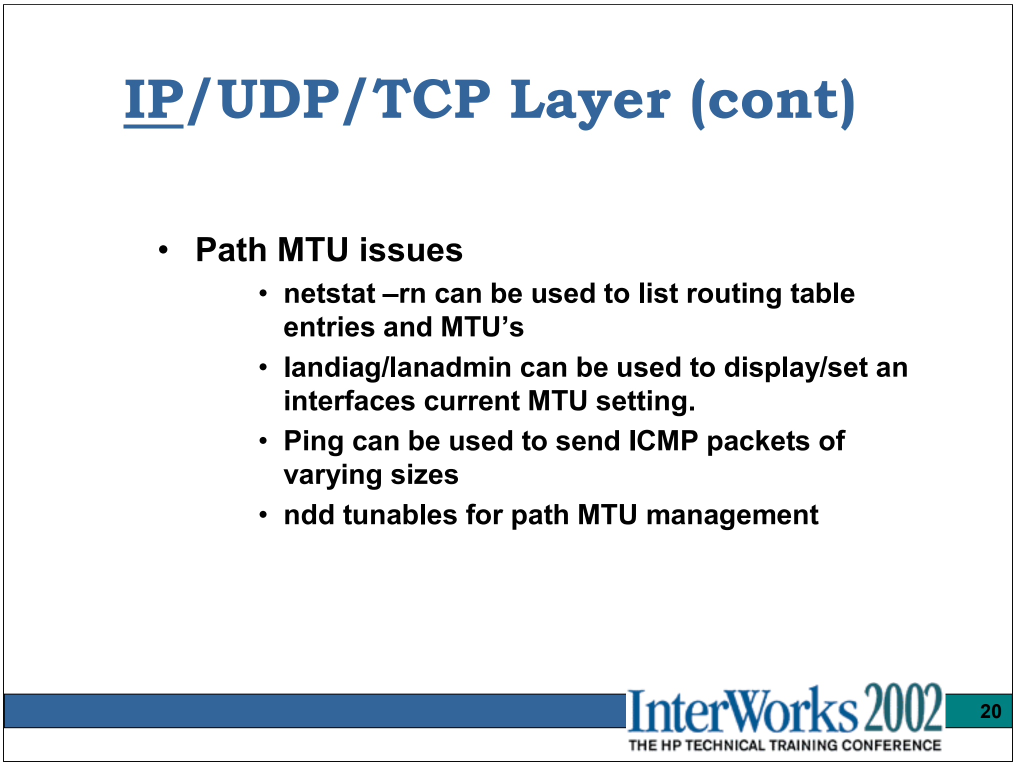
Path MTU issues frequently arise from concerns about mixed link level topologies such as FDDI, Token ring, ethernet, WAN’s.

IP routing through the network can affect application performance. Understand what route is being taken and why.

Multihomed systems (those with multiple IP interfaces) and systems with aliased (secondary) IP addresses on an interface can add to the confusion of which interfaces or paths through a network a connection will take. An accurate topology map with system NIC/IP addresses accurately labeled is a must.

The Arp caches in the switches and routers can on occasion become ‘confused’ about which MAC address is associated with which port. The HA or relocatable IP addresses have certain behaviors that _should_ keep the IP/MAC mappings updated correctly, but on occasion, a switch reboot or arp cache clear has been known to clear up erratic connectivity issues.

The ICMP messages (not true IP datagrams) of this type are not so much problems as indicators of problems. They also have a feature of responding with the UDP/TCP/IP header of the packet which generated the ICMP message…a big help in isolating the real cause of trouble.



IP/UDP/TCP Layer (cont)

- **Path MTU issues**
  - **netstat –rn can be used to list routing table entries and MTU's**
  - **landiag/lanadmin can be used to display/set an interfaces current MTU setting.**
  - **Ping can be used to send ICMP packets of varying sizes**
  - **ndd tunables for path MTU management**

If FDDI or token ring topologies are involved in the connection paths, there will likely be some network component (router or switch) doing IP fragmentation to manage the MTU differences. This is not so much a concern for TCP since it attempts to discover the proper path MTU as part of the connection setup phase for every TCP connection. UDP however has no such mechanism. It is not unheard of to discover IP fragmentation disabled or not supported on some old switch equipment.

The ***netstat –rn*** command displays the path MTU associated with each routing table entry.

The ***route*** command can be used to set route PMTU's for network and host routes, but only new connections will use the altered setting. Existing connections (in the case of TCP) use previous route MTU settings.

The ***lanadmin –m*** command can be used to check current MTU value for an interface. The ***lanadmin –M <mtu>*** command can be used to set the MTU for an interface which is often a handy way to temporarily play with MTU size while troubleshooting.

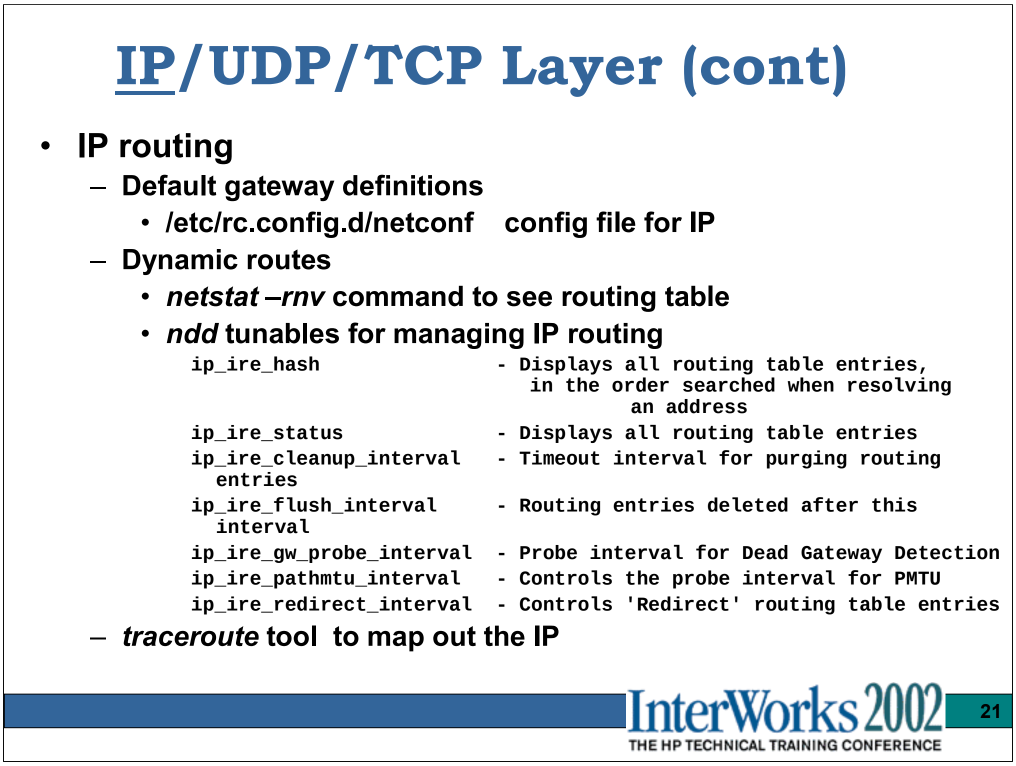
The ***ping -p <host> <packet-size>*** command can be used to probe path MTU issues. Packet sizes can vary from 64-4095 bytes.

The ***ndd*** command can control three MTU related variables:

ip_ire_pathmtu_interval - Controls the probe interval for PMTU

ip_pmtu_strategy - Controls the Path MTU Discovery strategy

tcp_ignore_path_mtu - Disable setting MSS from ICMP 'Frag Needed'



IP/UDP/TCP Layer (cont)

- IP routing
  - Default gateway definitions
    - /etc/rc.config.d/netconf   config file for IP
  - Dynamic routes
    - *netstat –rnv* command to see routing table
    - *ndd* tunables for managing IP routing

`ip_ire_hash`	- Displays all routing table entries, in the order searched when resolving an address
`ip_ire_status`	- Displays all routing table entries
`ip_ire_cleanup_interval`	- Timeout interval for purging routing entries
`ip_ire_flush_interval`	- Routing entries deleted after this interval
`ip_ire_gw_probe_interval`	- Probe interval for Dead Gateway Detection
`ip_ire_pathmtu_interval`	- Controls the probe interval for PMTU
`ip_ire_redirect_interval`	- Controls 'Redirect' routing table entries

  - *traceroute* tool to map out the IP

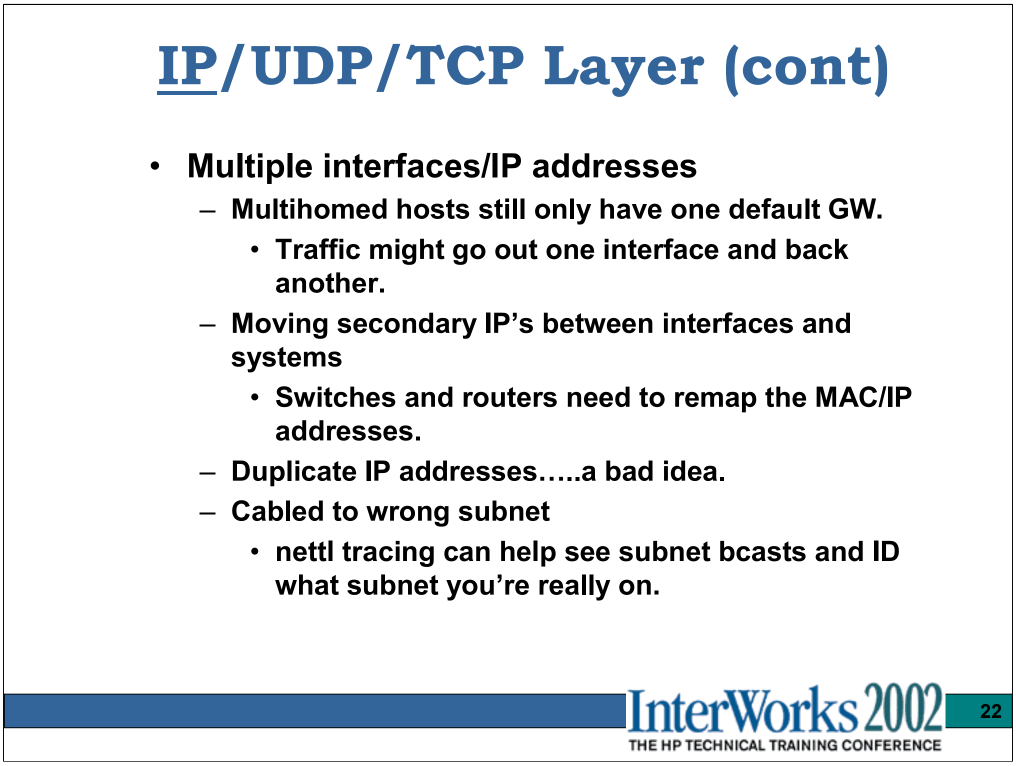
The netconf file in /etc/rc.config.d is the file which controls the configuration for the core networking subsystems on HP-UX. It assigns IP addresses to the LAN interfaces, sets the default gateway, determines if DHCP is used, and controls the enabling of gated.

The netstat –rnv command can be used to display the current routing tables active on the system:

```
Routing tables
Dest/Netmask                    Gateway            Flags    Refs Interface   Pmtu
127.0.0.1/255.255.255.255       127.0.0.1          UH         0  lo0         4136
10.10.30.28/255.255.255.255     10.10.30.28        UH         0  lan1        4136
15.24.46.28/255.255.255.255     15.24.46.28        UH         0  lan0        4136
15.24.40.0/255.255.248.0        15.24.46.28        U          2  lan0        1500
127.0.0.0/255.0.0.0             127.0.0.1          U          0  lo0            0
default/0.0.0.0                 15.24.47.253       UG         0  lan0           0
```

The traceroute command in /usr/contrib/bin can be used to map out the path an IP packet will take through the network. It also reports approximate response/latency times.

```
# ./traceroute hpatlse.atl.hp.com
traceroute to hpatlse.atl.hp.com (15.51.240.6), 30 hops max, 40 byte packets
 1  bel2410gw1.nsr.hp.com (15.24.55.253)  0.714 ms  0.611 ms  0.582 ms
 2  172.16.65.1 (172.16.65.1)  82.933 ms  82.511 ms  82.604 ms
 3  atlgwb03-leg148.cns.hp.com (15.24.240.58)  82.217 ms  82.179 ms  82.191 ms
 4  atlsite5.tio.atl.hp.com (15.41.16.215)  82.879 ms  82.647 ms  82.410 ms
 5  hpatlse.atl.hp.com (15.51.240.6)  82.966 ms  82.815 ms  82.368 ms
```



IP/UDP/TCP Layer (cont)

- Multiple interfaces/IP addresses
  - Multihomed hosts still only have one default GW.
    - Traffic might go out one interface and back another.
  - Moving secondary IP's between interfaces and systems
    - Switches and routers need to remap the MAC/IP addresses.
  - Duplicate IP addresses…..a bad idea.
  - Cabled to wrong subnet
    - nettl tracing can help see subnet bcasts and ID what subnet you're really on.

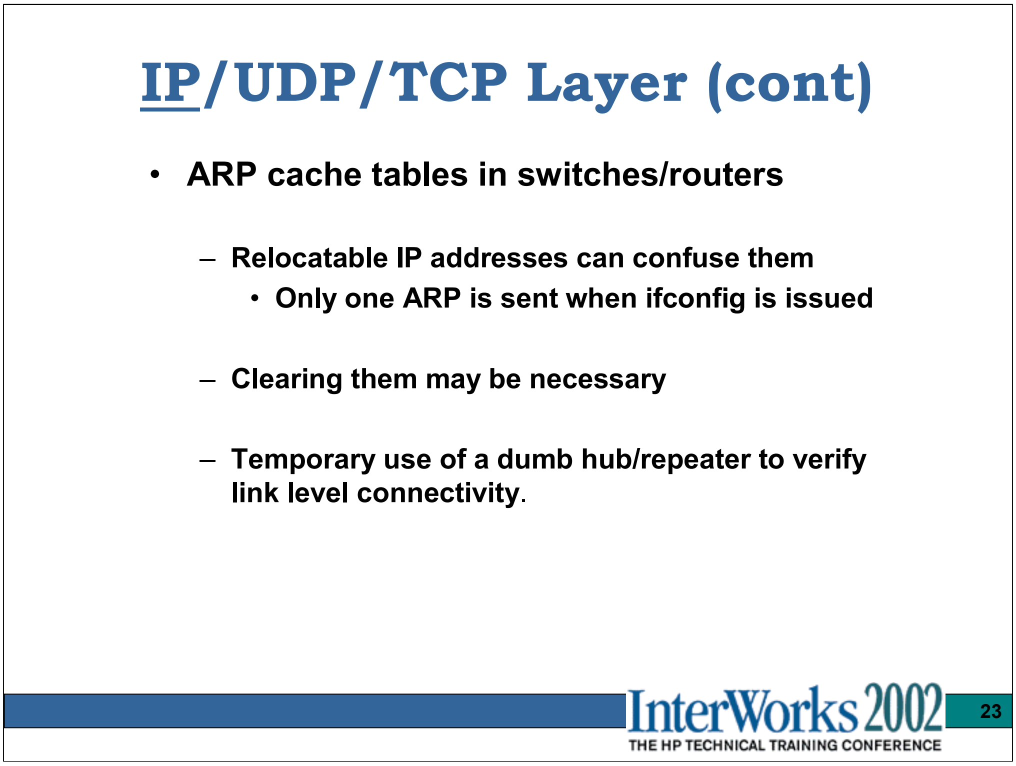
The IP addresses for interfaces are assigned/configured in the ***/etc/rc.config.d/netconf*** file.

netstat –in will show all interface IP assignments and current packet in/out counts.

ifconfig <lan name unit> displays the IP/subnetmask and other interface settings. When used to assign an IP (primary or secondary) to an interface, an unsolicited, self directed ARP packet is sent out to 'advertise' the new IP/MAC address mapping.

Duplicate IP addresses do still occur, so use another local subnet system's arp cache to see if his IP/MAC address mapping changes…then note the two MAC addresses. (The ***arp –an*** command)

If you're unsure whether the interface is on the correct subnet (ie. Can't ping what you think is another valid IP for the subnet) you can use ***nettl*** tracing to trace the inbound IP broadcast traffic….look at that traffic for source IP addresses.



IP/UDP/TCP Layer (cont)

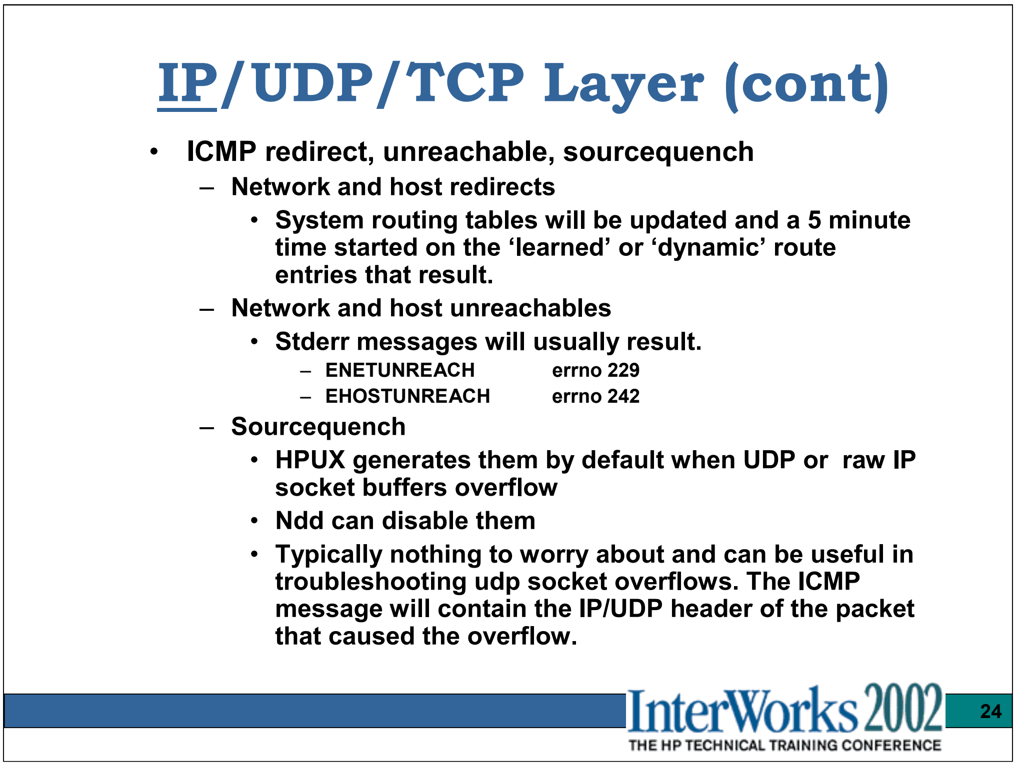
- **ARP cache tables in switches/routers**
  - **Relocatable IP addresses can confuse them**
    - **Only one ARP is sent when ifconfig is issued**
  - **Clearing them may be necessary**
  - **Temporary use of a dumb hub/repeater to verify link level connectivity.**

The current MAC address cached in the ARP cache for local subnet IP addresses can be seen using the ***arp –an*** command.

Be aware that local interface factory default MAC addresses can be overwritten at startup time by the "*_STATION_ADDRESS=" variable in the various interface config files in the ***/etc/rc.config.d*** directory.

Relocatable IP's require that switch/routers be notified of MAC/IP address mapping changes. When an IP address is assigned with the ***ifconfig*** command, only one unsolicited ARP is sent out.

The switch/router arp cache and port assignment tables can get 'confused' on occasion and need to be reset/cleared.



IP/UDP/TCP Layer (cont)

- **ICMP redirect, unreachable, sourcequench**
  - **Network and host redirects**
    - **System routing tables will be updated and a 5 minute time started on the ‘learned’ or ‘dynamic’ route entries that result.**
  - **Network and host unreachables**
    - **Stderr messages will usually result.**
      - **ENETUNREACH errno 229**
      - **EHOSTUNREACH errno 242**
  - **Sourcequench**
    - **HPUX generates them by default when UDP or raw IP socket buffers overflow**
    - **Ndd can disable them**
    - **Typically nothing to worry about and can be useful in troubleshooting udp socket overflows. The ICMP message will contain the IP/UDP header of the packet that caused the overflow.**

InterWorks 2002
THE HP TECHNICAL TRAINING CONFERENCE

At the IP layer, the ICMP network unreachable, host unreachable, network redirect, host redirect, and sourcequench messages are of interest. A summary of the ICMP messages received and sent by the transport can be obtained using the command ***netstat –sp icmp***.

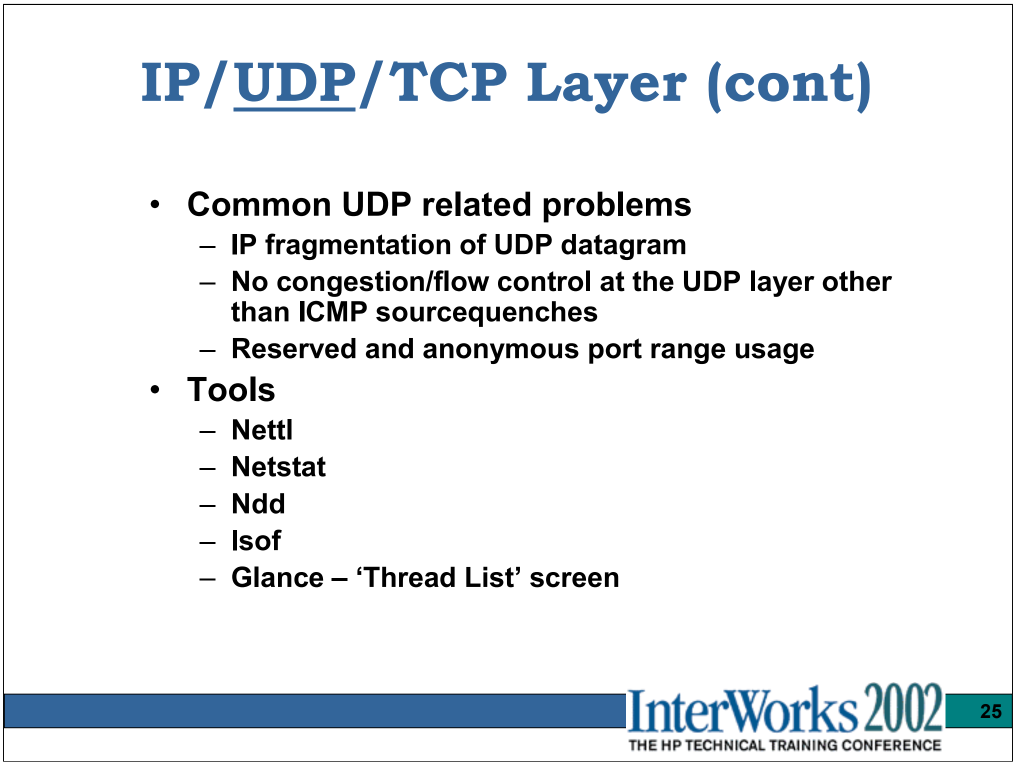
The ***ping –o*** command can be used to check basic IP connectivity and in most cases report the path through the network that the ICMP message traverses. The ***traceroute*** tool can also be used to map out the network path a packet traverses.

The ***netstat –rn*** command will show ‘flags’ for the routing entries and those with the “D” flag set are one learned/updated due to ICMP redirects. There are two ndd tunables that affect ICMP redirects:

```
ip_ire_redirect_interval - Controls 'Redirect' routing table entries
                           (5 minute default)
ip_send_redirects      -  Sends ICMP 'Redirect' packets
                           (enabled by default)
```

ICMP sourcequench messages are ‘advisory’ in nature, and how a particular transport responds to them will vary. HP-UX ignores them but does send them out (another ndd tunable enables/disables them) when a local UDP or RAWIP socket buffer overflows. The UDP stats given by ***netstat –sp udp*** do not tell you which UDP socket has overflowed, but another ndd command does (on 11.11 and above). ***ndd –get /dev/ip ip_udp_status*** dumps the UDP fanout table, and contained within that for each UDP port is the overflow count. This is only available for HP-UX 11.0 with ARPA transport patch ***PHNE_23456 or later*** installed.

See ***ndd –h ip_send_source_quench*** also.



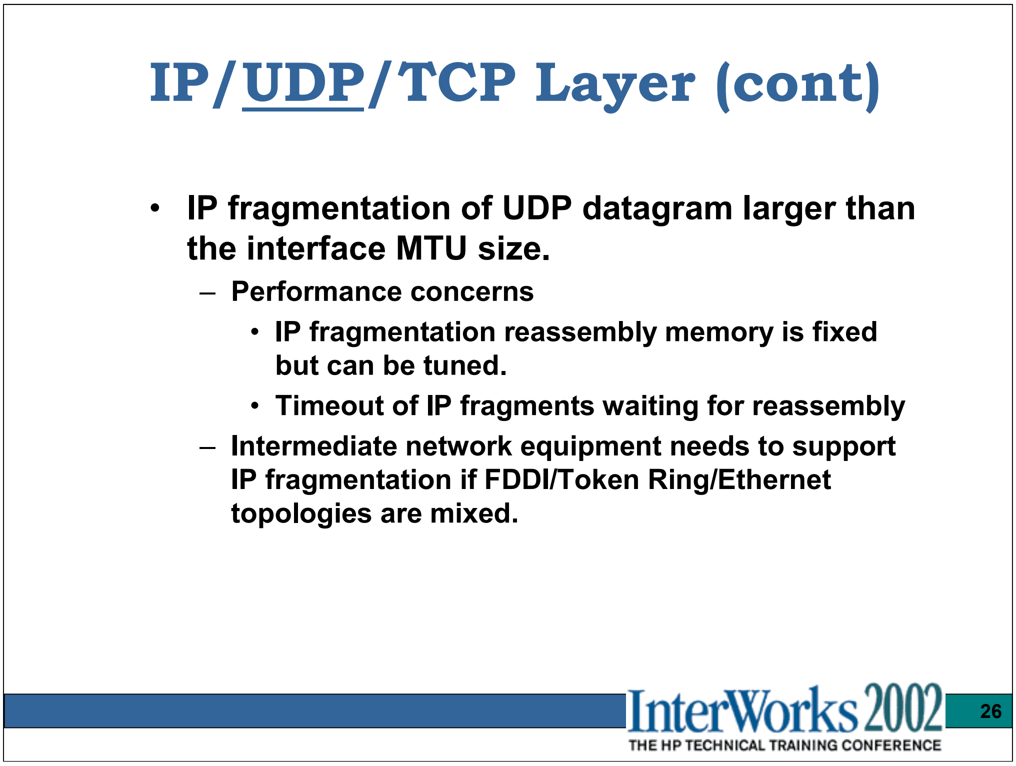
IP/UDP/TCP Layer (cont)

- Common UDP related problems
  - IP fragmentation of UDP datagram
  - No congestion/flow control at the UDP layer other than ICMP sourcequenches
  - Reserved and anonymous port range usage
- Tools
  - Nettl
  - Netstat
  - Ndd
  - lsof
  - Glance – ‘Thread List’ screen

Most frequent UDP abuser is NFS PV3 with 32k read/write sizes. The 32k read/write bursts are comprised of 21+ 1500 byte packets in a burst and can contribute to network congestion if the server is at Gigabit speeds and clients at 100bt.

Intermediate network equipment needs to support IP fragmentation if FDDI/Token Ring/Ethernet topologies are mixed.

Any UDP or raw IP socket buffer that overflows will cause an ICMP source quench packet to be sent out. Many network administrators get a bit concerned about seeing them, but in reality most network devices ignore them…but some may not. By default HP-UX does send them, but there is an ndd tunable to control them (*ndd –h ip_send_source_quench*). It is often difficult to determine which UDP or raw IP ports are overflowing. As mentioned in the previous slide, you can find the UDP .

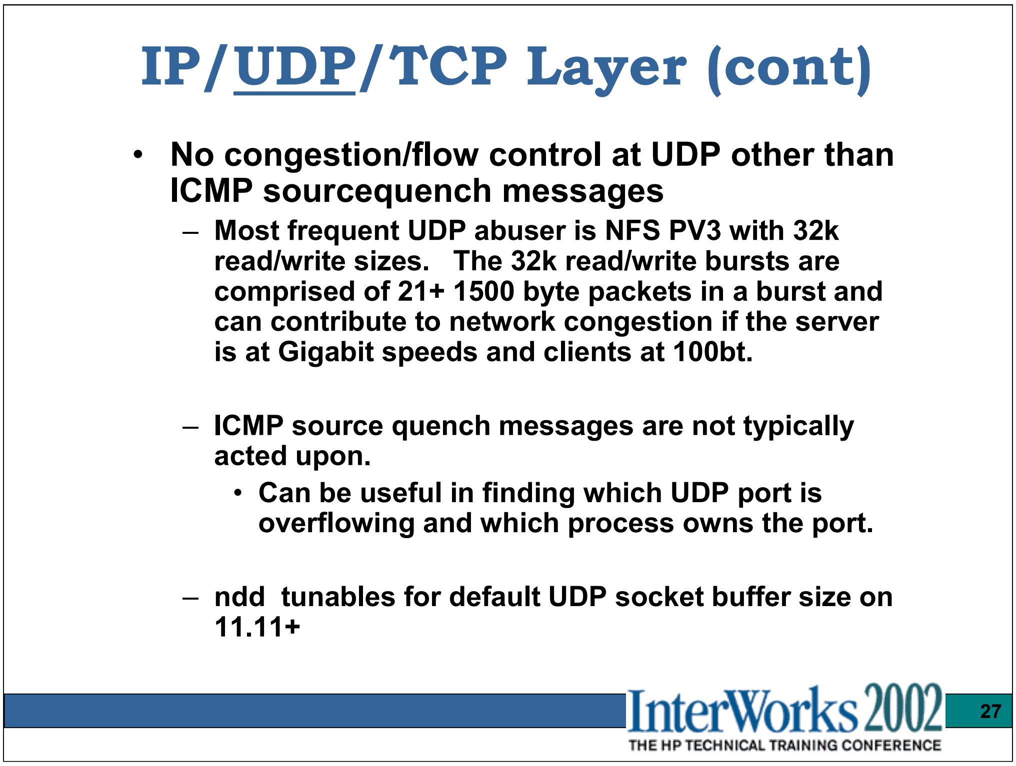


IP/UDP/TCP Layer (cont)

- IP fragmentation of UDP datagram larger than the interface MTU size.
  - Performance concerns
    - IP fragmentation reassembly memory is fixed but can be tuned.
    - Timeout of IP fragments waiting for reassembly
  - Intermediate network equipment needs to support IP fragmentation if FDDI/Token Ring/Ethernet topologies are mixed.

Two ndd tunables referring to IP reassembly (typically a UDP datagram being reassembled) are:

```
ip_reass_mem_limit     -  Maximum number of bytes for IP
                          reassembly (2 megs default)
ip_fragment_timeout    -  Controls how long IP fragments are
                          kept (60 secs default)
```



IP/UDP/TCP Layer (cont)

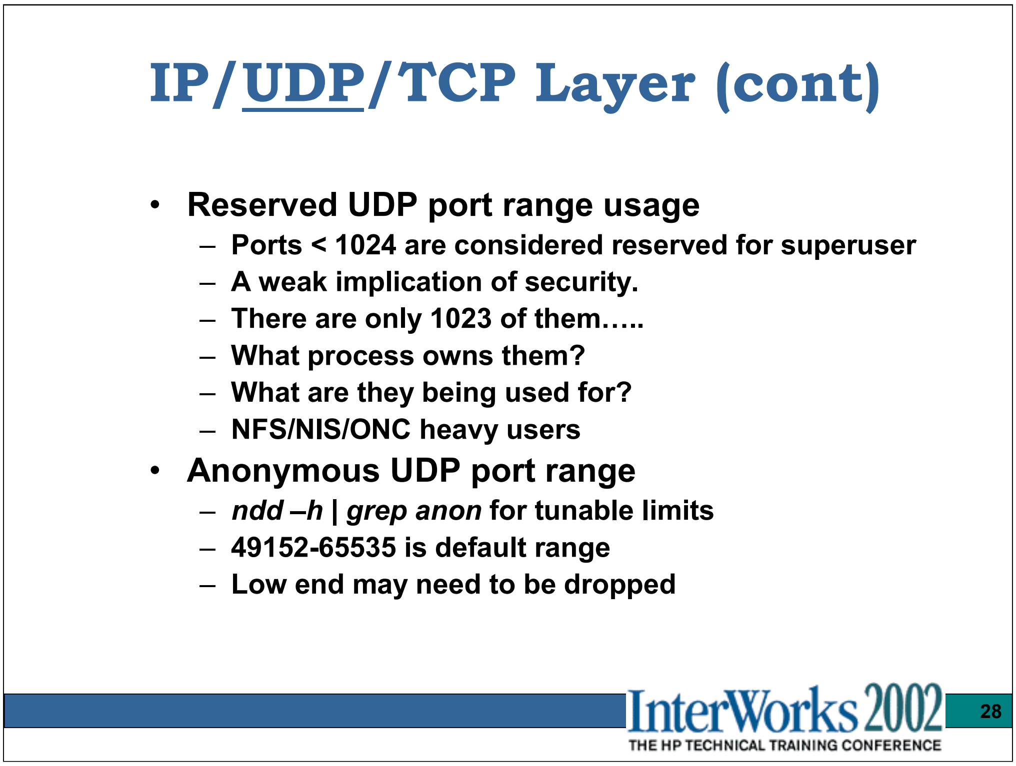
- No congestion/flow control at UDP other than ICMP sourcequench messages
  - Most frequent UDP abuser is NFS PV3 with 32k read/write sizes. The 32k read/write bursts are comprised of 21+ 1500 byte packets in a burst and can contribute to network congestion if the server is at Gigabit speeds and clients at 100bt.
  - ICMP source quench messages are not typically acted upon.
    - Can be useful in finding which UDP port is overflowing and which process owns the port.
  - ndd tunables for default UDP socket buffer size on 11.11+

Any UDP or raw IP socket buffer that overflows will cause an ICMP source quench packet to be sent out. Many network administrators get a bit concerned about seeing them, but in reality most network devices ignore them…but some may not. By default HP-UX does send them and ignores them when received, but there is an ndd tunable to control sending them (*ndd –h ip_send_source_quench*).

It is often difficult to determine which UDP or raw IP ports are overflowing and more importantly which process is asleep at the wheel…i.e. not reading from their socket in a timely manner.

As described earlier, you can use nettl tracing to catch the ICMP sourcequench message going out and look at the UDP header that it attaches from the original packet. You can also use ***ndd –get /dev/ip ip_udp_status*** to see which UDP sockets have overflows. Once you know the UDP port number, a tool like ***lsof*** (List Open Special Files…a very handy tool) which maps all the current running processes' open files, can be used to see which process own the particular UDP socket. You can also use nettl to trace outbound packets for that port and see what the kernel thread ID is of the sending process from the nettl trace record header. With the kernel thread ID and the use of the Glance performance monitoring tool, you can map the thread ID to a process ID.

Then you need the developer to tell you what his starchild process is doing when it's not reading from it's UDP socket.….let's hope he built in some logging features.



IP/UDP/TCP Layer (cont)

- Reserved UDP port range usage
  - Ports < 1024 are considered reserved for superuser
  - A weak implication of security.
  - There are only 1023 of them…..
  - What process owns them?
  - What are they being used for?
  - NFS/NIS/ONC heavy users
- Anonymous UDP port range
  - *ndd –h | grep anon* for tunable limits
  - 49152-65535 is default range
  - Low end may need to be dropped

netstat –an | grep udp will show the open UDP ports but not who owns them.

lsof –n | grep UDP will list all processes that have UDP ports open.

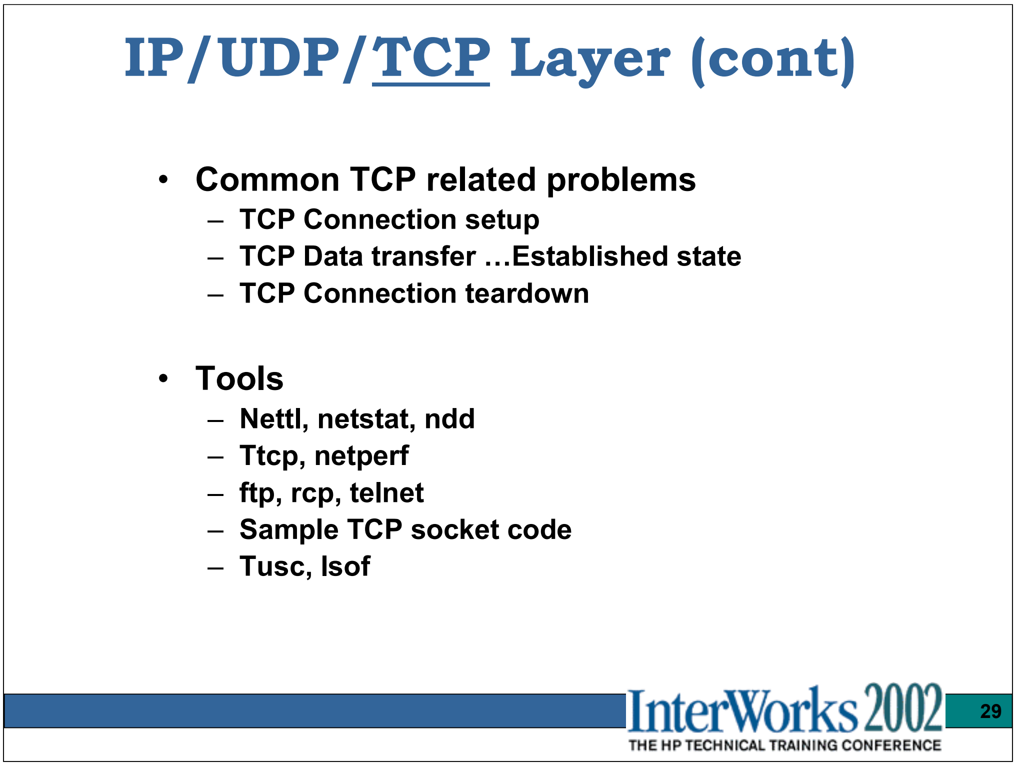
The kernel has a pool of UDP ports open for nfs client calls. The ***lsof*** tool will not show these ports since there is no user space process that owns them.

With the port number and/or the kernel thread ID (remember ***glance*** allows you to list the kernel thread ID's of processes) of the owner, you can do some ***nettl*** tracing to watch who and what the port is in use for…or just ***kill*** the owning process for getting between you and a UDP port.

The UDP ports that are allocated when a bind() call is made and a port is NOT explicitly requested are referred to as 'anonymous' or ephemeral port ranges.

More than the 24k default amount may be needed. Typically the bind() system call will fail with errno EADDRNOTAVAIL . The ndd command controls two tunables for UDP and a similar two tunables for TCP:

```
# ndd -h | grep anon
    tcp_largest_anon_port     - Largest anonymous port number to use
    udp_largest_anon_port     - Largest anonymous port number to use
    tcp_smallest_anon_port     - Smallest anonymous port number to use
    udp_smallest_anon_port     - Smallest anonymous port number to use
```



IP/UDP/TCP Layer (cont)

- **Common TCP related problems**
  - **TCP Connection setup**
  - **TCP Data transfer …Established state**
  - **TCP Connection teardown**

- **Tools**
  - **Nettl, netstat, ndd**
  - **Ttcp, netperf**
  - **ftp, rcp, telnet**
  - **Sample TCP socket code**
  - **Tusc, lsof**

The issues seen with the TCP protocol and the network management of a TCP connection will be discussed here separately from the system calls and application usage of TCP sockets. That comes later in the socket/application layer section.

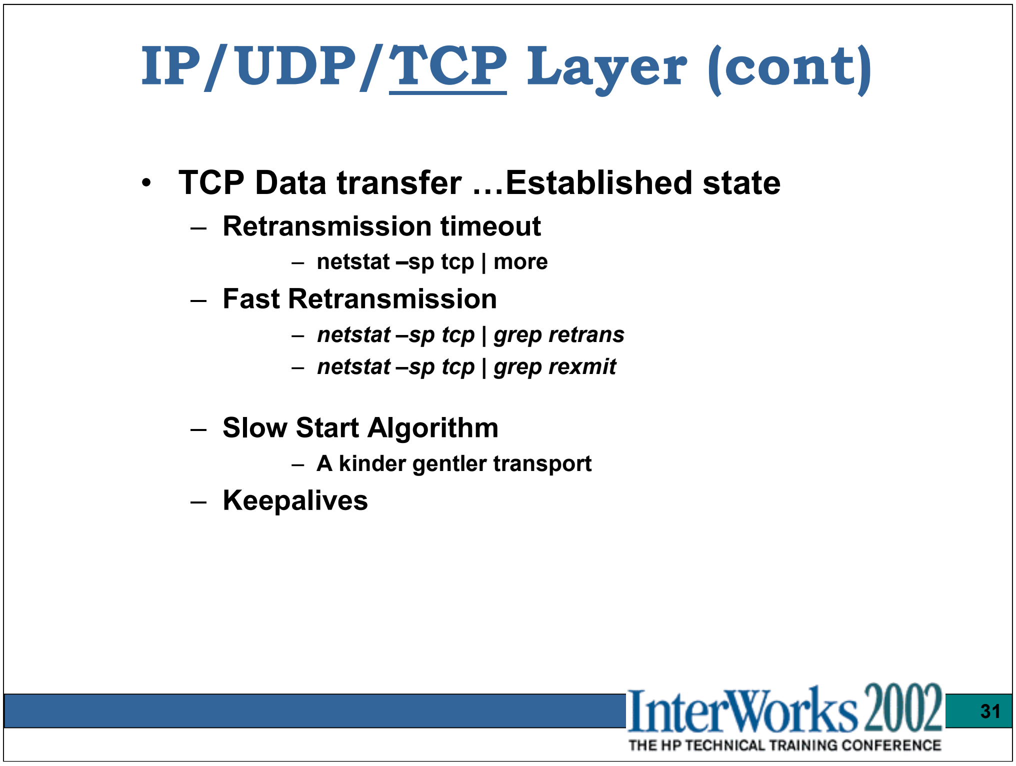
Again, many of the same tools used for link/IP/UDP layer investigation work here as well.

IP/UDP/TCP Layer (cont)

- **TCP Connection setup**
  - 3-way handshake
    - Kernel will put conn in EST before listener accept()s. The listen backlog queue size determines how many can be waiting. System default is 20 .
    - Take note of TCP options in SYN packets..MSS and window scaling options.
  - Connection timeouts
    - *ndd –h tcp_ip_abort_cinterval* 75 sec on HPUX 11.X
    - *netstat –an | grep SYN_SENT* is a clue
    - *ndd –h tcp_conn_grace_period*
  - Connection rejected
    - resets of failed connection attempts may tack on added text info to the reset packet
    - *netstat –sp tcp* to see drops due to queue full or no listener

Client side program			Server side program	
Conn state	System call	TCP packet	System call	Conn state
			lsd=socket()	
	sd=socket()		bind(lsd,..)	
			listen(lsd,backlog)	LISTEN
	connect(sd)	SYN →		
SYN_SENT				
		← SYN ACK		
				SYN_RCVD
	<connect returns>	ACK →		
				ESTABLISHED
ESTABLISHED			sd=accept(lsd,..)	
	send(sd,data)	DATA →	recv(sd,data,…)	

```
------------------------------- TCP Header ---------------------------------
sport:   55555   -->   dport:  49199     flags: RST ACK
       seq: 0x0          urp: 0x0      chksum: 0x1f     data len: 11
       ack: 0x306842ba  win: 0x0      optlen: 0
------------------------------- User Data ----------------------------------
   0: 4e 6f 20 6c 69 73 74 65 6e 65 72 -- -- -- -- --  No listener.....

0 connect requests dropped due to full queue
50 connect requests dropped due to no listener
```



IP/UDP/TCP Layer (cont)

- **TCP Data transfer …Established state**
  - **Retransmission timeout**
    - **netstat –sp tcp | more**
  - **Fast Retransmission**
    - ***netstat –sp tcp | grep retrans***
    - ***netstat –sp tcp | grep rexmit***
  - **Slow Start Algorithm**
    - **A kinder gentler transport**
  - **Keepalives**

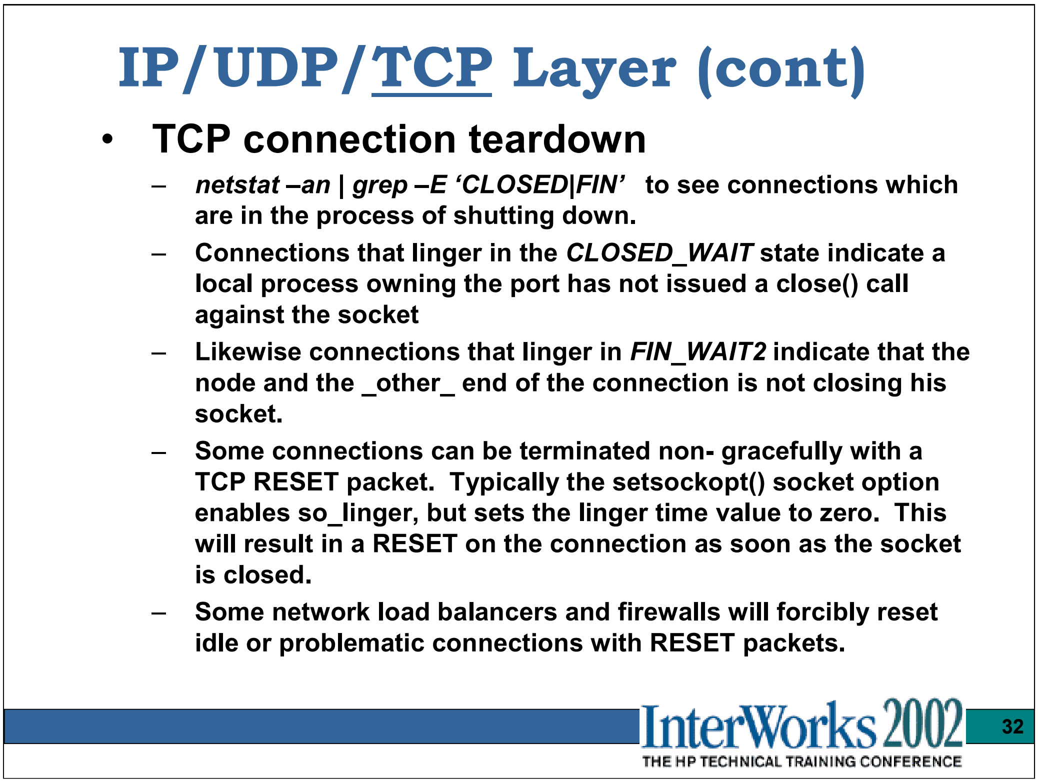
ndd tunables for TCP retransmit timers:

```
tcp_rexmit_interval_initial     - Initial value for round trip time-out
tcp_rexmit_interval_initial_lnp - tcp_rexmit_interval_initial for LNP
tcp_rexmit_interval_max   - Upper limit for computed round trip timeout
tcp_rexmit_interval_min   - Lower limit for computed round trip timeout
tcp_dupack_fast_retransmit - No. of ACKs needed to trigger a retransmit
```

TCP keepalives start after 2 hours of inactivity on a connection by default.

ndd –h tcp_keepalive_interval

Some network equipment (Load balancers, firewalls) may terminate an idle connection earlier than 2 hours….Usually just a TCP reset packet is received.



IP/UDP/TCP Layer (cont)

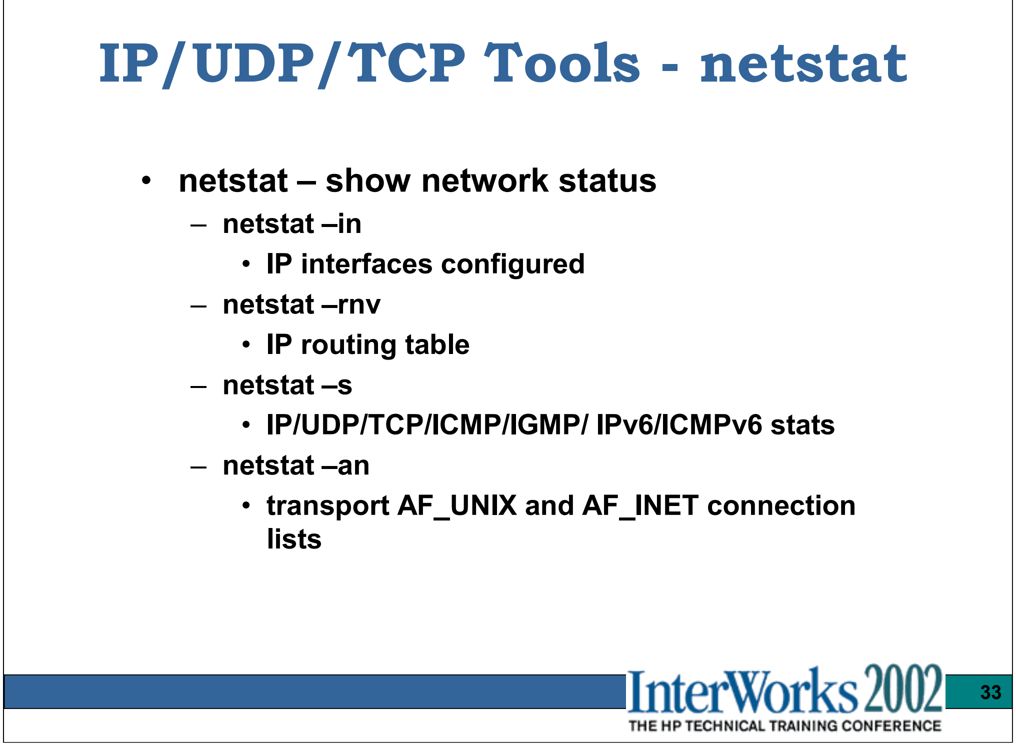
- **TCP connection teardown**
  - ***netstat –an | grep –E ‘CLOSED|FIN’*** **to see connections which are in the process of shutting down.**
  - **Connections that linger in the *CLOSED_WAIT* state indicate a local process owning the port has not issued a close() call against the socket**
  - **Likewise connections that linger in *FIN_WAIT2* indicate that the node and the _other_ end of the connection is not closing his socket.**
  - **Some connections can be terminated non- gracefully with a TCP RESET packet. Typically the setsockopt() socket option enables so_linger, but sets the linger time value to zero. This will result in a RESET on the connection as soon as the socket is closed.**
  - **Some network load balancers and firewalls will forcibly reset idle or problematic connections with RESET packets.**

Client side program			Server side program	
Conn state	System call	TCP packet	System call	Conn state
ESTABLISHED				ESTABLISHED
	close()	FIN →		
FIN_WAIT1			listen(lsd,backlog)	CLOSE_WAIT
		← ACK		
FIN_WAIT2				
		← FIN	close()	
TIME_WAIT				LAST_ACK
		ACK →		CLOSED

Important states to look for are CLOSE_WAIT on the server side and FIN_WAIT2 on the client side. If they persist it implies the server side process owning the TCP port is not closing it’s socket….the question is why?

Either side can initiate the connection shutdown. The side that does, I am labeling the ‘client’ in this example. The terms ‘client’ and ‘server’ are used in many different contexts.

The above connection diagram can differ depending on the use of the SO_LINGER socket option.



IP/UDP/TCP Tools - netstat

- netstat – show network status
  - netstat –in
    - IP interfaces configured
  - netstat –rnv
    - IP routing table
  - netstat –s
    - IP/UDP/TCP/ICMP/IGMP/ IPv6/ICMPv6 stats
  - netstat –an
    - transport AF_UNIX and AF_INET connection lists

netstat –in

```
Name          Mtu  Network          Address              Ipkts     Opkts
lan1*         1500 10.10.30.0       10.10.30.28         318005    105508
lan0          1500 15.24.40.0       15.24.46.28        1373097    327999
lo0           4136 127.0.0.0        127.0.0.1           177261    177262
```

netstat –rnv

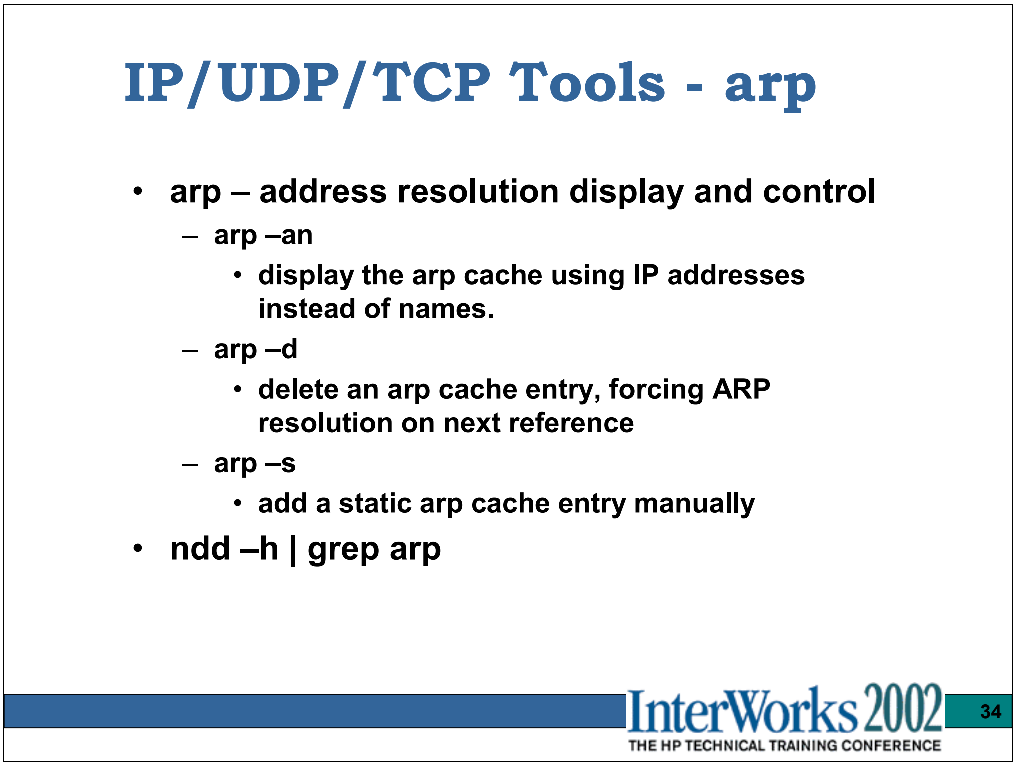
```
Dest/Netmask                    Gateway            Flags   Refs Interface  Pmtu
127.0.0.1/255.255.255.255       127.0.0.1          UH         0  lo0       4136
10.10.30.28/255.255.255.255     10.10.30.28        UH         0  lan1      4136
15.24.46.28/255.255.255.255     15.24.46.28        UH         0  lan0      4136
15.24.40.0/255.255.248.0        15.24.46.28        U          2  lan0      1500
127.0.0.0/255.0.0.0             127.0.0.1          U          0  lo0          0
default/0.0.0.0                 15.24.47.253       UG         0  lan0         0
```

netstat –s

```
tcp:
        381147 packets sent
                267989 data packets (239228277 bytes)
                48 data packets (2602 bytes) retransmitted
        …and on and on with remain tcp, udp.up.icmp, etc  stats
```

netstat -an

```
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address         (state)
tcp        0      0  127.0.0.1.49310        127.0.0.1.49226          ESTABLISHED
tcp        0      0  *.13                   *.*                      LISTEN
tcp        0      0  *.37                   *.*                      LISTEN
udp        0      0  *.13                   *.*
udp        0      0  *.518                  *.*
udp        0      0  *.514                  *.*
udp        0      0  *.*                    *.*
```



IP/UDP/TCP Tools - arp

- **arp – address resolution display and control**
  - arp –an
    - display the arp cache using IP addresses instead of names.
  - arp –d
    - delete an arp cache entry, forcing ARP resolution on next reference
  - arp –s
    - add a static arp cache entry manually
- **ndd –h | grep arp**

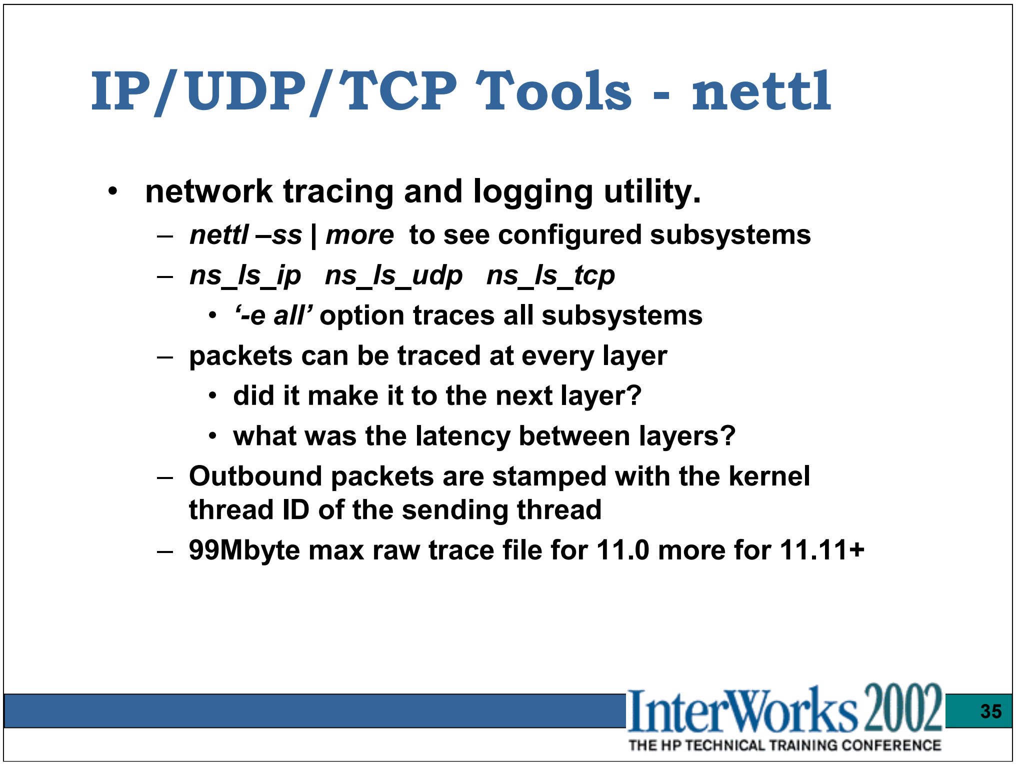
InterWorks 2002
THE HP TECHNICAL TRAINING CONFERENCE

arp –an

```
# arp -an
 (15.24.46.33) at 0:50:da:29:f8:33 ether
 (15.24.46.27) at 0:30:6e:6:15:d3 ether
 (15.24.47.253) at 0:60:83:41:90:1c ether
```

ndd –h | grep arp

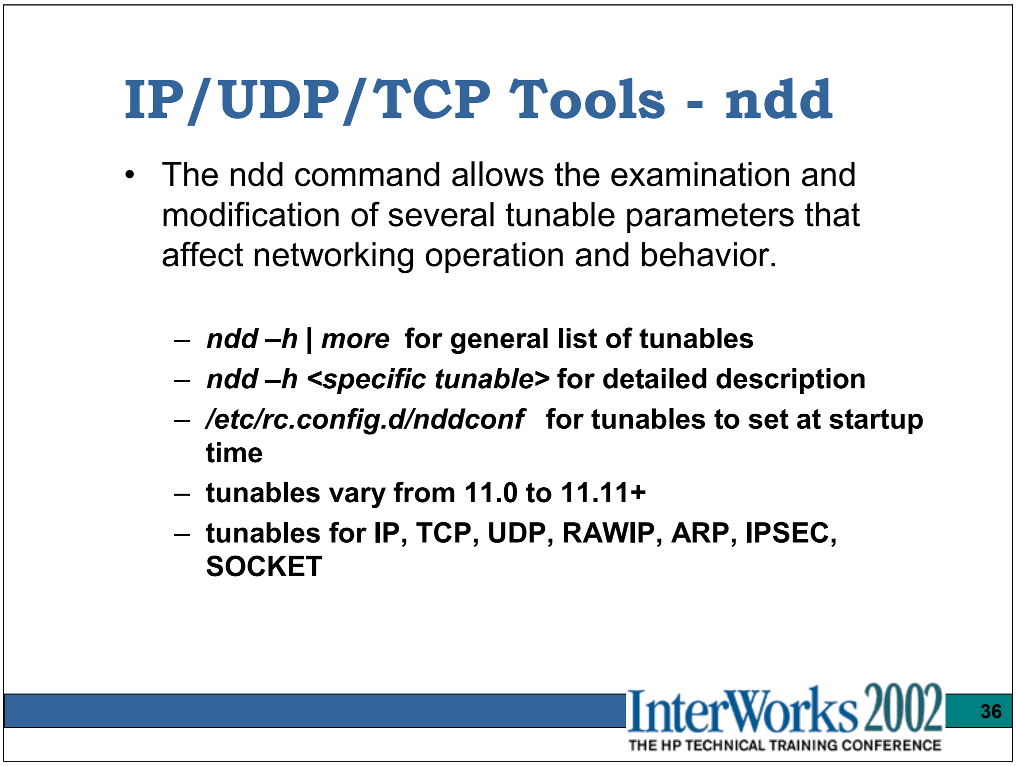
```
arp_cache_report          - Displays the ARP cache
arp_cleanup_interval      - Controls how long ARP entries stay in the
arp_defend_interval       - Seconds to wait before initially defending a
arp_redefend_interval     - Seconds to wait before defending a published
arp_resend_interval       - Number milliseconds between arp request
arp_announce_count         - Number of transmits used to announce a
arp_debug                  - Controls the level of ARP module debugging
arp_dl_sap                 - Set the SAP when ARP binds to a DLPI device
arp_dl_snap_sap            - The SAP to use for SNAP encapsulation
arp_probe_count            - Number of address resolution requests to make
```



IP/UDP/TCP Tools - nettl

- network tracing and logging utility.
  - *nettl –ss | more* to see configured subsystems
  - *ns_ls_ip ns_ls_udp ns_ls_tcp*
    - *'-e all'* option traces all subsystems
  - packets can be traced at every layer
    - did it make it to the next layer?
    - what was the latency between layers?
  - Outbound packets are stamped with the kernel thread ID of the sending thread
  - 99Mbyte max raw trace file for 11.0 more for 11.11+

For syntax/usage refer to the man page and the previous discussion in the Link Layer tools section.



IP/UDP/TCP Tools - ndd

- The ndd command allows the examination and modification of several tunable parameters that affect networking operation and behavior.
  - ***ndd –h | more*** **for general list of tunables**
  - ***ndd –h <specific tunable>*** **for detailed description**
  - ***/etc/rc.config.d/nddconf*** **for tunables to set at startup time**
  - **tunables vary from 11.0 to 11.11+**
  - **tunables for IP, TCP, UDP, RAWIP, ARP, IPSEC, SOCKET**

Some ndd commands (typically the ones labeled *_status) which dump large amounts of kernel information will have significant network performance impact. Global transport locks are held while key data structures are dumped. Most notably the ‘***ndd –get /dev/tcp tcp_status’*** command will halt all inbound traffic processing while the TCP fanout table is dumped….on systems with 2000+ TCP connections, this can mean a 5+ second network outage.

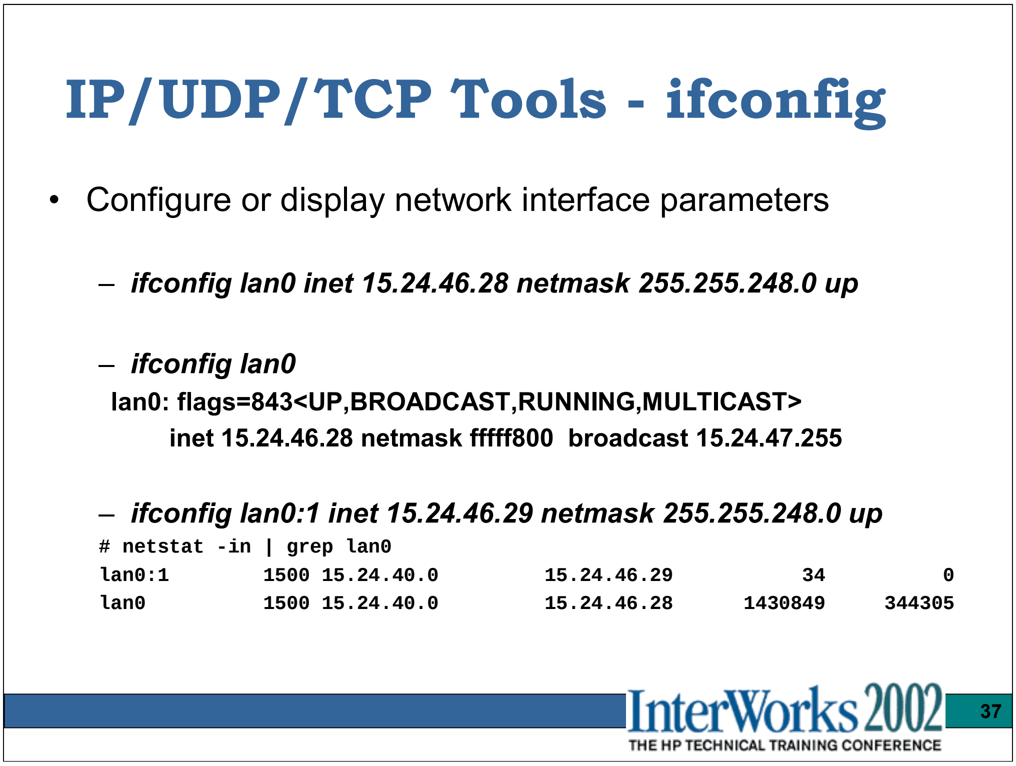
The ndd data returned is sent back via a single streams message. The system tunable STRMSGSZ (default is 64k) may need to be increased to allow all data to be seen.

The first field of the tcp_status output is a pointer to the tcp_t structure in the kernel. The q4 utility can be used to dump this structure to get detailed info about the state of a particular TCP connection…including the LISTEN sockets.

Here’s an example of looking at the TCP LISTEN socket for inetd’s telnet port 0x17 or 23 .

The field in the square brackets [xx,xx] are the local and remote port numbers.

```
# ndd -get /dev/tcp tcp_status | grep LISTEN | grep 17,
0000000100ea9068 000.000.000.000 045ce9d6 045ce9d5 00000000 00000000 00000000 00
000000 00000000 01500 00536 [17,0] TCP_LISTEN
# q4 /stand/vmunix /dev/mem
q4> load tcp_t from 0x0000000100ea9068
loaded 1 tcp_t as an array (stopped by max count)
q4> print -tx > tcp_telnet_listen_port          To see all kinds of TCP info or…
q4> print -tx tcp_conn_ind_cnt   tcp_conn_ind_max
tcp_conn_ind_cnt  0      ← How many conn’s we currently have waiting
tcp_conn_ind_max  0x14   ← max of 20 inbound connections pending.
```



IP/UDP/TCP Tools - ifconfig

- Configure or display network interface parameters
  - *ifconfig lan0 inet 15.24.46.28 netmask 255.255.248.0 up*
  - *ifconfig lan0*

lan0: flags=843<UP,BROADCAST,RUNNING,MULTICAST>
inet 15.24.46.28 netmask fffff800 broadcast 15.24.47.255

  - *ifconfig lan0:1 inet 15.24.46.29 netmask 255.255.248.0 up*

```
# netstat -in | grep lan0
lan0:1        1500 15.24.40.0        15.24.46.29         34         0
lan0          1500 15.24.40.0        15.24.46.28    1430849    344305
```

Basic command used to add/display IP addresses to interfaces.   The above shows an example of a primary Ip and a secondary IP being added.

To disable the secondary IP use:

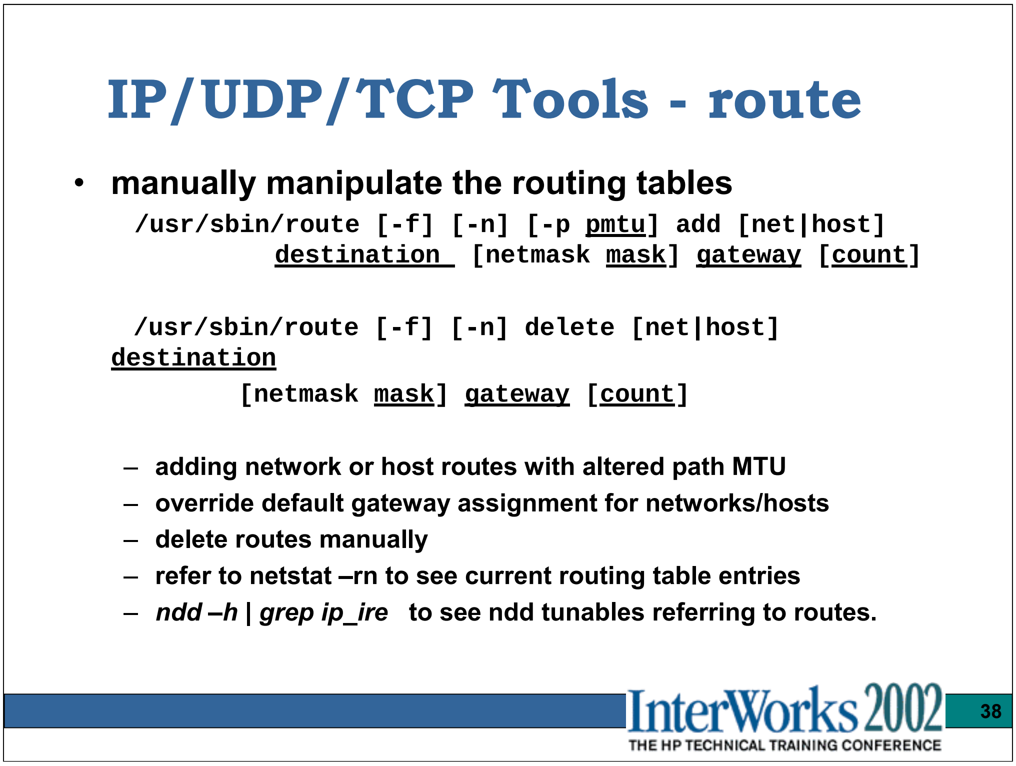
```
# ifconfig lan0:1 down
# netstat -in | grep lan0
lan0:1*       1500 15.24.40.0        15.24.46.29            125         0
lan0          1500 15.24.40.0        15.24.46.28        1430942    344386
```

To completely unplumb the IP address use:

```
# ifconfig lan0:1 0
# ifconfig lan0:1
lan0:1: flags=842<BROADCAST,RUNNING,MULTICAST>
        inet 0.0.0.0 netmask 0
# netstat -in | grep lan0
lan0:1*       1500 none              none                     0         0
lan0          1500 15.24.40.0        15.24.46.28        1431053    344455
```

Note the * asterisk denotes a down IP interface

.



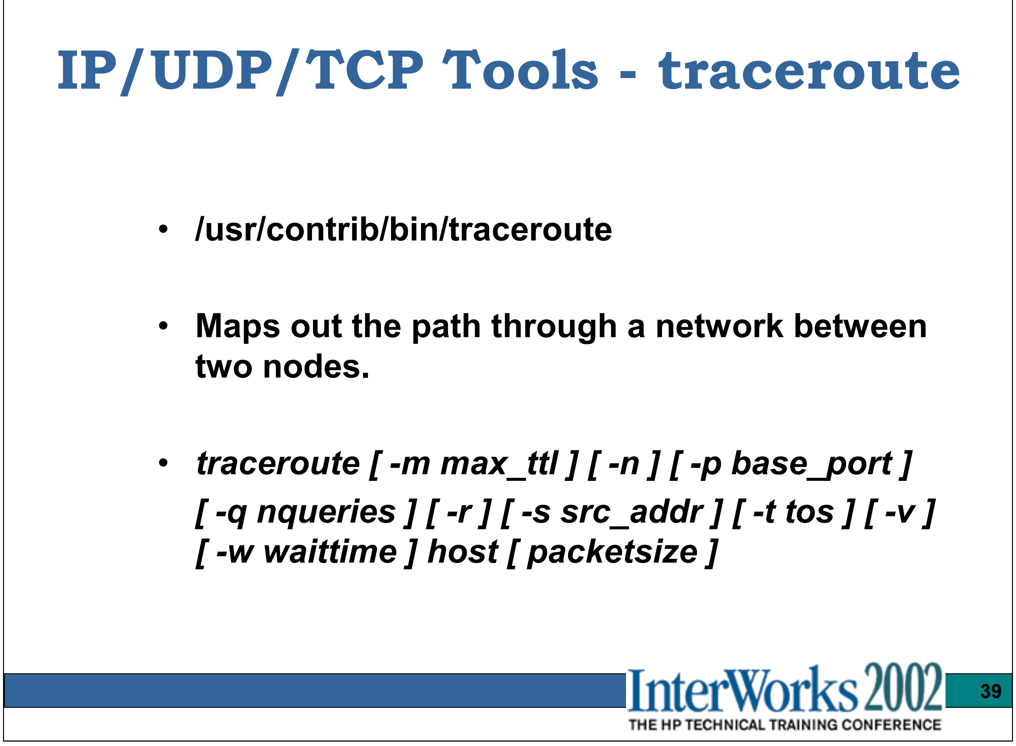
IP/UDP/TCP Tools - route

- manually manipulate the routing tables

```
  /usr/sbin/route [-f] [-n] [-p pmtu] add [net|host]
            destination  [netmask mask] gateway [count]

  /usr/sbin/route [-f] [-n] delete [net|host]
destination
          [netmask mask] gateway [count]
```

  - adding network or host routes with altered path MTU
  - override default gateway assignment for networks/hosts
  - delete routes manually
  - refer to netstat –rn to see current routing table entries
  - *ndd –h | grep ip_ire* to see ndd tunables referring to routes.

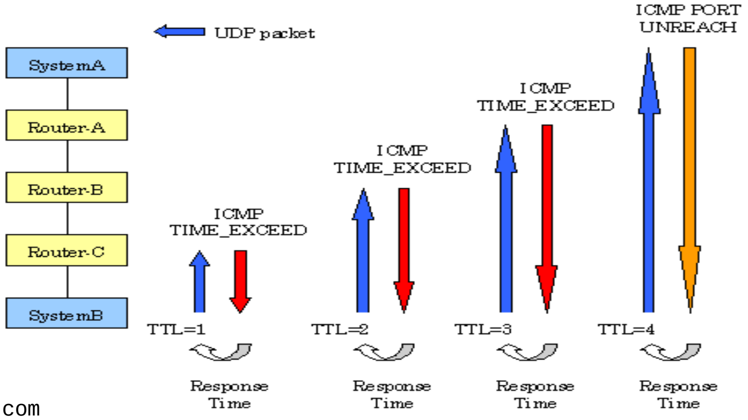
```
# ndd -h | grep ip_ire
    ip_ire_gw_probe            - Enable dead gateway probes
    ip_ire_hash                - Displays all routing table entries, in the
                                 order searched when resolving an IP address.
    ip_ire_status              - Displays all routing table entries
    ip_ire_cleanup_interval    - Timeout interval for purging unused
                                 routing entries
    ip_ire_flush_interval      - All routing entries deleted after this
                                 interval, even active routes.
    ip_ire_gw_probe_interval   - Probe interval for Dead Gateway Detection
    ip_ire_pathmtu_interval    - Controls the probe interval for PMTU
    ip_ire_redirect_interval   - Controls 'Redirect' routing table entries
```



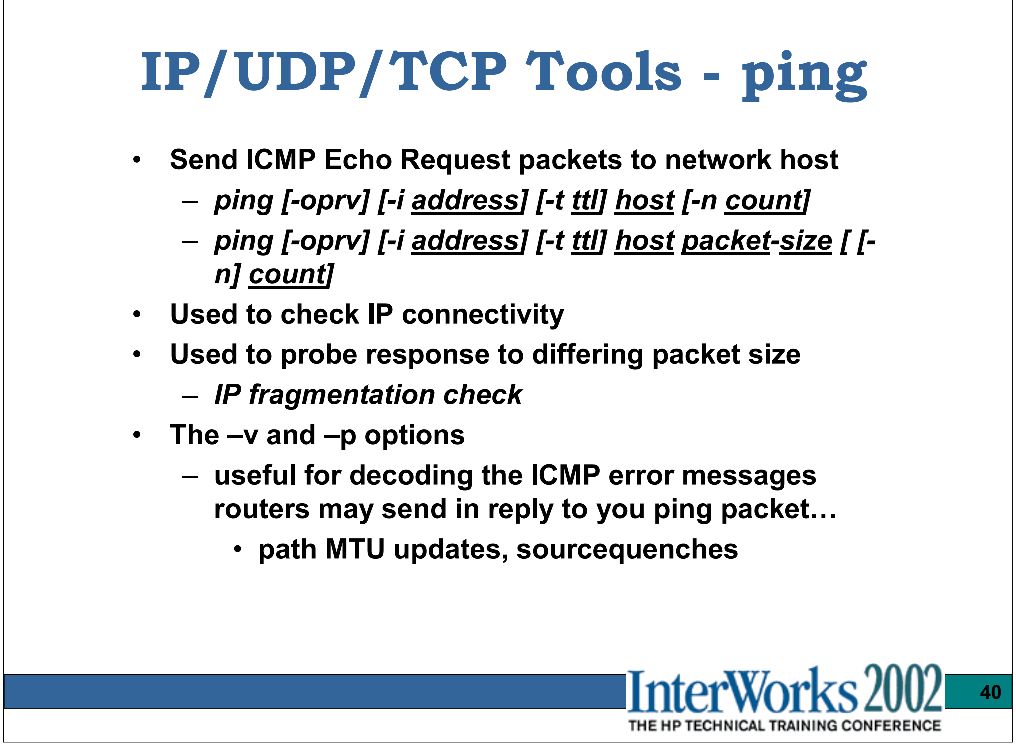
IP/UDP/TCP Tools - traceroute

- **/usr/contrib/bin/traceroute**
- **Maps out the path through a network between two nodes.**
- ***traceroute [-m max_ttl] [-n] [-p base_port]***
  [-q nqueries] [-r] [-s src_addr] [-t tos] [-v]
  [-w waittime] host [packetsize]

Traceroute command attempts to send a UDP packet to the remote host by setting IP header's TTL(time to live) field to 1. Then, the adjacent router will generate ICMP TIME_EXCEED message. Traceroute can measure the response time of this 1st hop router with this ICMP message. It sends the same packet for 3 times and prints each time's response. Next, it sends the same UDP packet setting TTL to 2. Then, the next hop router will generate ICMP TIME_EXCEED message. Using this message, traceroute can measure the response time of the 2nd hop router. It repeats it for 3 times for the 2nd router again. Traceroute will increment TTL and measure each router's response time in the same way. Then, if the UDP packet reaches the final destination, it can generate ICMP PORT_UNREACH message since traceroute uses port numbers which are not likely to be used (33434, 33435, 33436. ..). The diagram below describes this behavior.



```
# ./traceroute wtc.cup.hp.com

traceroute to wtec.cup.hp.com (15.XX.XX.XX), 30 hops max, 40 byte packets

 1  bl241g1.nsr.hp.com (15.XX.XX.XX)  0.697 ms  1.170 ms  0.588 ms

 2  172.XX.XX.XX (172.XX.XX.XX)  33.579 ms  34.013 ms  34.226 ms

 3  pagb02-legh2.americas.hp.net (15.XX.XX.XX)  34.872 ms  36.888 ms  35.660 ms

 4  cugb01-p9-1-0.americas.hp.net (15.XX.XX.XX)  35.212 ms  34.513 ms  34.777 ms

 5  cp4-gw2.cup.hp.com (15.XX.XX.XX)  34.469 ms  33.857 ms  34.446 ms

 6  cp5-gw.cup.hp.com (15.XX.XX.XX)  34.424 ms  34.144 ms  34.204 ms

 7  wtc.cup.hp.com (15.XX.XX.XX)  34.103 ms  34.622 ms  34.419 ms
```



IP/UDP/TCP Tools - ping

- Send ICMP Echo Request packets to network host
  - *ping [-oprv] [-i address] [-t ttl] host [-n count]*
  - *ping [-oprv] [-i address] [-t ttl] host packet-size [[-n] count]*
- Used to check IP connectivity
- Used to probe response to differing packet size
  - *IP fragmentation check*
- The –v and –p options
  - useful for decoding the ICMP error messages routers may send in reply to you ping packet…
    - path MTU updates, sourcequenches

```
# ping -v hpujrlz
PING hpujrlz.jpn.hp.com: 64 byte packets
92 bytes from 15.74.172.191: icmp_type=4 (Source Quench)
x00: x4500005c
x04: x00000000
x08: xff015260
x0c: x0f4aacbf
x10: x0f4aacc0
x14: x0400fbff
x18: x00000000
x1c: x45000054
x20: xaf204000
x24: xff015474
x28: x0f4aacc0
x2c: x0f4aacbf
x30: x08005f54
x34: x13a40000
icmp_code=0
64 bytes from 15.74.172.191: icmp_seq=0. time=3. ms
92 bytes from 15.74.172.191: icmp_type=4 (Source Quench)
x00: x4500005c
x04: x00000000 : : : :
```

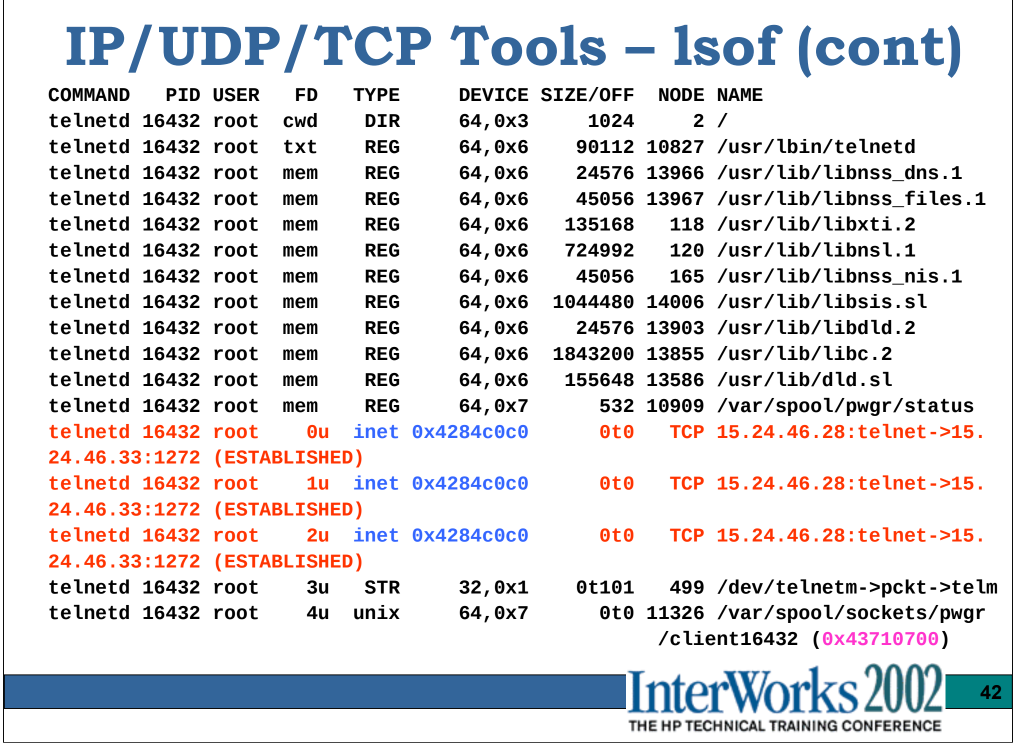
```
# ping -vp 5.5.5.1 1500
PING 5.5.5.1: 1500 byte packets
92 bytes from 15.74.172.191: icmp_type=3 (Dest
Unreachable)
x00: x4500005c
x04: xc8414000
x08: xff013914
x0c: x0f4aacbf
x10: x0f4aaee3
x14: x03046b37 | Type = 3 | Code = 4 | Checksum |
x18: x00000200 | unused = 0 | Next-Hop MTU |
x1c: x450005dc
x20: xe4116000
x24: xfe01aadb
x28: x0f4aaee3
x2c: x05050501
 x30: x08006757
x34: x0fd00000 icmp_code=4
new Path MTU = 512
1500 bytes from 5.5.5.1: icmp_seq=1. time=4. ms
----5.5.5.1 PING Statistics----
2 packets transmitted, 1 packets received, 50% packet loss
round-trip (ms) min/avg/max = 4/4/4
```

IP/UDP/TCP Tools - lsof

- **List Open Special Files utility.**
  - **Lists open files for every running process and where possible tries to map it to a meaningful file name/path or type**
  - **Lists local and remote IP and UDP/TCP port numbers**
  - **provides socket address pointers for AF_INET and AF_UNIX sockets**
  - **shows what shared libs are mmap'ed in.**
  - **cwd for the process**
  - **Does not show kernel owned sockets…i.e. NFS**

The latest distribution of lsof is available via anonymous ftp from the host vic.cc.purdue.edu. You'll find the lsof distribution in the pub/tools/unix/lsof directory. You can also use this **http://ftp.cerias.purdue.edu/pub/tools/unix/sysutils/lsof**

```
# ./lsof -n -p 16432
COMMAND   PID USER   FD   TYPE     DEVICE SIZE/OFF  NODE NAME
telnetd 16432 root  cwd    DIR     64,0x3     1024     2 /
telnetd 16432 root  txt    REG     64,0x6    90112 10827 /usr/lbin/telnetd
telnetd 16432 root  mem    REG     64,0x6    24576 13966 /usr/lib/libnss_dns.1
telnetd 16432 root  mem    REG     64,0x6    45056 13967 /usr/lib/libnss_files.1
telnetd 16432 root  mem    REG     64,0x6   135168   118 /usr/lib/libxti.2
telnetd 16432 root  mem    REG     64,0x6   724992   120 /usr/lib/libnsl.1
telnetd 16432 root  mem    REG     64,0x6    45056   165 /usr/lib/libnss_nis.1
telnetd 16432 root  mem    REG     64,0x6  1044480 14006 /usr/lib/libsis.sl
telnetd 16432 root  mem    REG     64,0x6    24576 13903 /usr/lib/libdld.2
telnetd 16432 root  mem    REG     64,0x6  1843200 13855 /usr/lib/libc.2
telnetd 16432 root  mem    REG     64,0x6   155648 13586 /usr/lib/dld.sl
telnetd 16432 root  mem    REG     64,0x7      532 10909 /var/spool/pwgr/status
telnetd 16432 root    0u  inet 0x4284c0c0      0t0   TCP 15.24.46.28:telnet->15.
24.46.33:1272 (ESTABLISHED)
telnetd 16432 root    1u  inet 0x4284c0c0      0t0   TCP 15.24.46.28:telnet->15.
24.46.33:1272 (ESTABLISHED)
telnetd 16432 root    2u  inet 0x4284c0c0      0t0   TCP 15.24.46.28:telnet->15.
24.46.33:1272 (ESTABLISHED)
telnetd 16432 root    3u   STR     32,0x1    0t101   499 /dev/telnetm->pckt->telm
telnetd 16432 root    4u  unix     64,0x7      0t0 11326 /var/spool/sockets/pwgr
/client16432 (0x43710700)
```



IP/UDP/TCP Tools – lsof (cont)

```
COMMAND   PID USER   FD   TYPE     DEVICE SIZE/OFF   NODE NAME
telnetd 16432 root  cwd    DIR      64,0x3     1024      2 /
telnetd 16432 root  txt    REG      64,0x6    90112  10827 /usr/lbin/telnetd
telnetd 16432 root  mem    REG      64,0x6    24576  13966 /usr/lib/libnss_dns.1
telnetd 16432 root  mem    REG      64,0x6    45056  13967 /usr/lib/libnss_files.1
telnetd 16432 root  mem    REG      64,0x6   135168    118 /usr/lib/libxti.2
telnetd 16432 root  mem    REG      64,0x6   724992    120 /usr/lib/libnsl.1
telnetd 16432 root  mem    REG      64,0x6    45056    165 /usr/lib/libnss_nis.1
telnetd 16432 root  mem    REG      64,0x6  1044480  14006 /usr/lib/libsis.sl
telnetd 16432 root  mem    REG      64,0x6    24576  13903 /usr/lib/libdld.2
telnetd 16432 root  mem    REG      64,0x6  1843200  13855 /usr/lib/libc.2
telnetd 16432 root  mem    REG      64,0x6   155648  13586 /usr/lib/dld.sl
telnetd 16432 root  mem    REG      64,0x7      532  10909 /var/spool/pwgr/status
telnetd 16432 root    0u  inet 0x4284c0c0       0t0    TCP 15.24.46.28:telnet->15.
24.46.33:1272 (ESTABLISHED)
telnetd 16432 root    1u  inet 0x4284c0c0       0t0    TCP 15.24.46.28:telnet->15.
24.46.33:1272 (ESTABLISHED)
telnetd 16432 root    2u  inet 0x4284c0c0       0t0    TCP 15.24.46.28:telnet->15.
24.46.33:1272 (ESTABLISHED)
telnetd 16432 root    3u   STR      32,0x1    0t101    499 /dev/telnetm->pckt->telm
telnetd 16432 root    4u  unix      64,0x7      0t0  11326 /var/spool/sockets/pwgr
                                                    /client16432 (0x43710700)
```

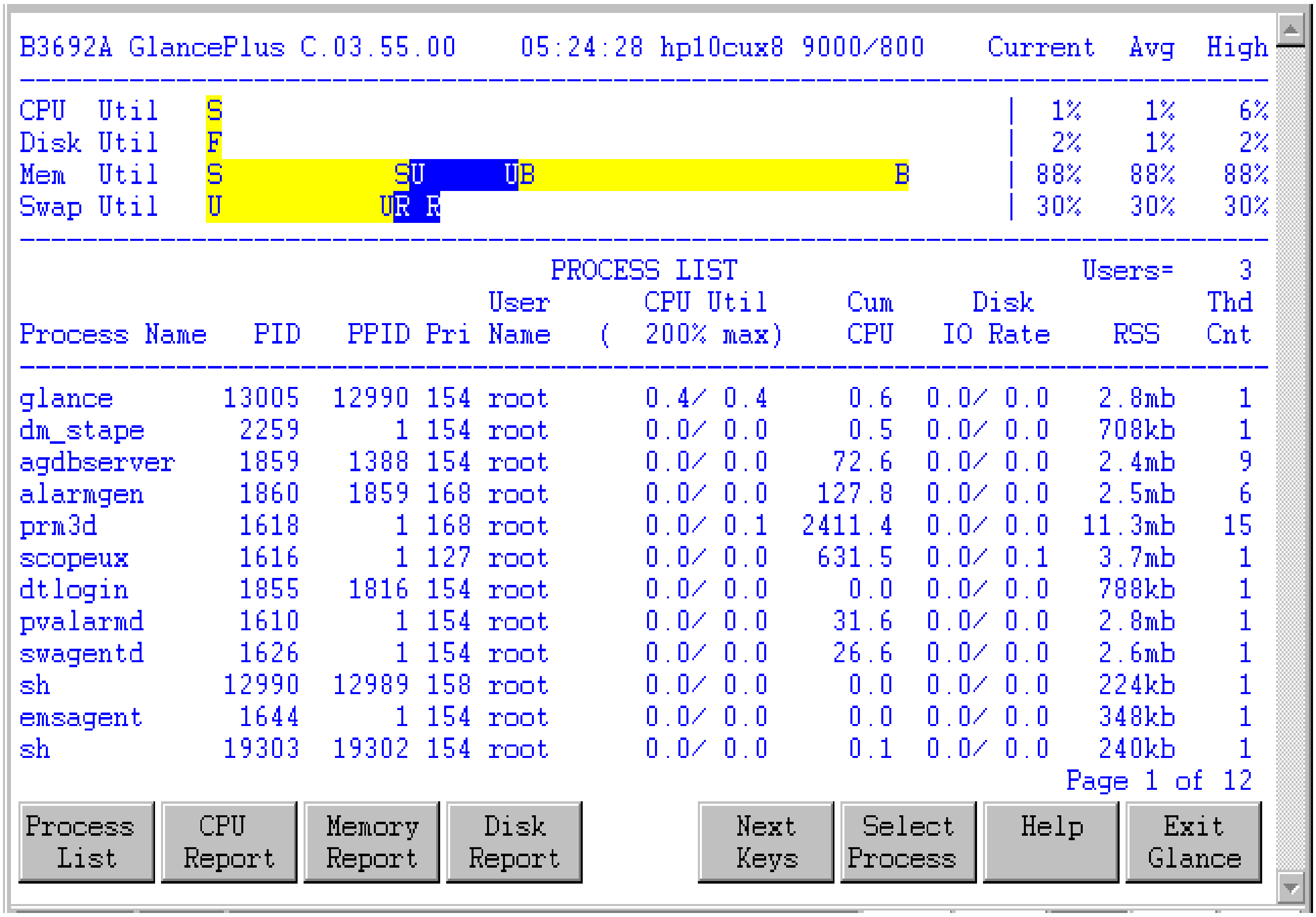
q4 /stand/vmunix /dev/mem

```
q4> load struct socket from 0x4284c0c0
q4> print -tx | so_type so_options so_linger so_state
            so_type  0x1
       so_options  0xc
        so_linger  0
         so_state  0x82
q4> print -tx | grep so_q
            so_q0  0
             so_q  0
         so_q0len  0
          so_qlen  0
        so_qlimit  0
q4> print -tx | grep buf
        so_sndbuf  0xffff
        so_rcvbuf  0xffff
```

See **/usr/include/sys/socket.h** for definition of types, flags and options fields.

IP/UDP/TCP Tools - Glance

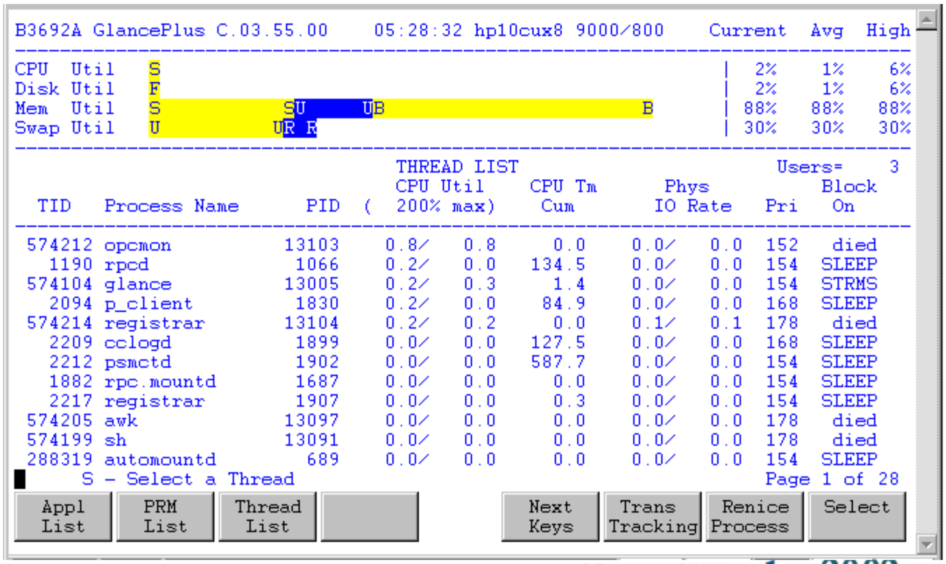
- GlancePlus system performance monitor for HP-UX
- Useful screens…..
  - **Thread List**
  - **Process syscalls**
  - **Open files**
    - **shows offset within files and file types/names**
  - **Network by interface**
  - **NFS global and by system**
    - **stats plus read/write rates for client and server**
  - **Memory report**
    - **buffercache size and pagein/out rates**



```
B3692A GlancePlus C.03.55.00     05:24:28 hp10cux8 9000/800    Current  Avg  High
--------------------------------------------------------------------------------
CPU  Util   S                                                 |  1%     1%    6%
Disk Util   F                                                 |  2%     1%    2%
Mem  Util   S           SU     UB                       B     | 88%    88%   88%
Swap Util   U          UR R                                   | 30%    30%   30%
--------------------------------------------------------------------------------
                                  PROCESS LIST                    Users=     3
                                User       CPU Util     Cum      Disk          Thd
Process Name   PID    PPID Pri Name    ( 200% max)     CPU    IO Rate     RSS   Cnt
--------------------------------------------------------------------------------
glance        13005  12990 154 root      0.4/ 0.4      0.6   0.0/ 0.0    2.8mb   1
dm_stape       2259      1 154 root      0.0/ 0.0      0.5   0.0/ 0.0    708kb   1
agdbserver     1859   1388 154 root      0.0/ 0.0     72.6   0.0/ 0.0    2.4mb   9
alarmgen       1860   1859 168 root      0.0/ 0.0    127.8   0.0/ 0.0    2.5mb   6
prm3d          1618      1 168 root      0.0/ 0.1   2411.4   0.0/ 0.0   11.3mb  15
scopeux        1616      1 127 root      0.0/ 0.0    631.5   0.0/ 0.1    3.7mb   1
dtlogin        1855   1816 154 root      0.0/ 0.0      0.0   0.0/ 0.0    788kb   1
pvalarmd       1610      1 154 root      0.0/ 0.0     31.6   0.0/ 0.0    2.8mb   1
swagentd       1626      1 154 root      0.0/ 0.0     26.6   0.0/ 0.0    2.6mb   1
sh            12990  12989 158 root      0.0/ 0.0      0.0   0.0/ 0.0    224kb   1
emsagent       1644      1 154 root      0.0/ 0.0      0.0   0.0/ 0.0    348kb   1
sh            19303  19302 154 root      0.0/ 0.0      0.1   0.0/ 0.0    240kb   1
                                                                   Page 1 of 12
```

IP/UDP/TCP Tools - Glance

- Thread List screen

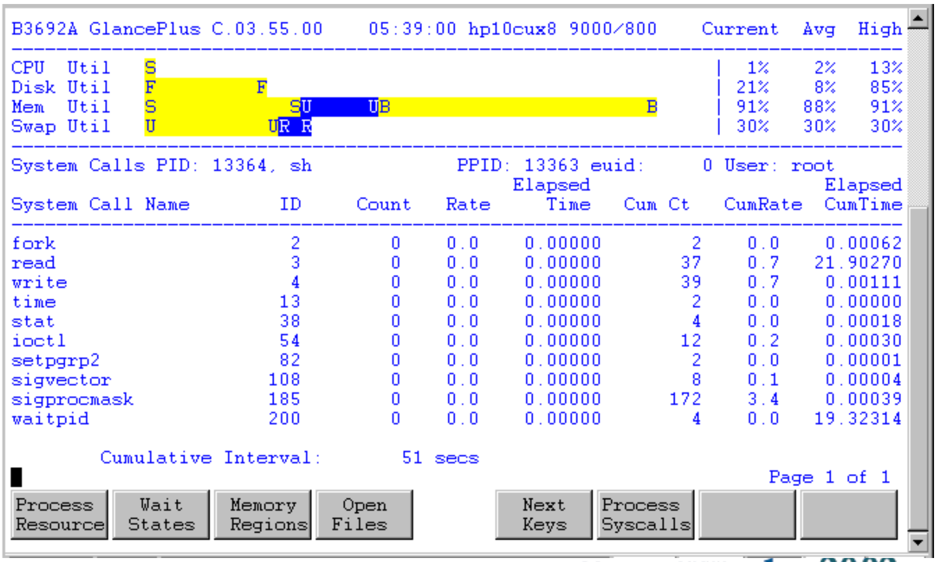


```
B3692A GlancePlus C.03.55.00    05:28:32 hp10cux8 9000/800    Current  Avg  High
--------------------------------------------------------------------------------
CPU  Util   S                                                  |  2%    1%    6%
Disk Util   F                                                  |  2%    1%    6%
Mem  Util   S           SU     UB                       B      | 88%   88%   88%
Swap Util   U          UR R                                    | 30%   30%   30%
--------------------------------------------------------------------------------
                                  THREAD LIST                       Users=    3
                                  CPU Util    CPU Tm      Phys          Block
  TID   Process Name      PID  (  200% max)    Cum       IO Rate   Pri   On
--------------------------------------------------------------------------------
 574212 opcmon          13103    0.8/   0.8      0.0    0.0/   0.0  152   died
   1190 rpcd             1066    0.2/   0.0    134.5    0.0/   0.0  154  SLEEP
 574104 glance          13005    0.2/   0.3      1.4    0.0/   0.0  154  STRMS
   2094 p_client         1830    0.2/   0.0     84.9    0.0/   0.0  168  SLEEP
 574214 registrar       13104    0.2/   0.2      0.0    0.1/   0.1  178   died
   2209 cclogd           1899    0.0/   0.0    127.5    0.0/   0.0  168  SLEEP
   2212 psmctd           1902    0.0/   0.0    587.7    0.0/   0.0  154  SLEEP
   1882 rpc.mountd       1687    0.0/   0.0      0.0    0.0/   0.0  154  SLEEP
   2217 registrar        1907    0.0/   0.0      0.3    0.0/   0.0  154  SLEEP
 574205 awk             13097    0.0/   0.0      0.0    0.0/   0.0  178   died
 574199 sh              13091    0.0/   0.0      0.0    0.0/   0.0  178   died
 288319 automountd        689    0.0/   0.0      0.0    0.0/   0.0  154  SLEEP
█     S - Select a Thread                                          Page 1 of 28
```

Recall that nettl stamps all outbound packet headers with the Kernel Thread ID (TID above) and you can specify this TID in the filter file used by the netfmt command to format the raw trace file.

IP/UDP/TCP Tools - Glance

- Process syscalls



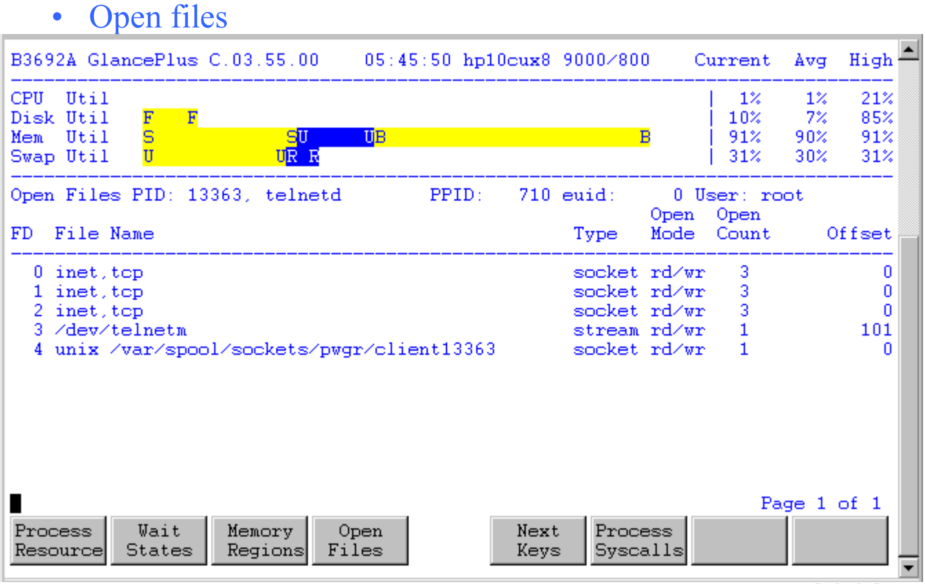
```
B3692A GlancePlus C.03.55.00    05:39:00 hp10cux8 9000/800    Current   Avg   High
----------------------------------------------------------------------------------
CPU  Util   S                                                   |  1%    2%    13%
Disk Util   F         F                                         | 21%    8%    85%
Mem  Util   S            SU     UB                          B   | 91%   88%    91%
Swap Util   U          UR R                                     | 30%   30%    30%
----------------------------------------------------------------------------------
System Calls PID: 13364, sh              PPID: 13363 euid:     0 User: root
                                              Elapsed                      Elapsed
System Call Name        ID      Count    Rate     Time   Cum Ct   CumRate  CumTime
----------------------------------------------------------------------------------
fork                     2          0     0.0  0.00000        2       0.0  0.00062
read                     3          0     0.0  0.00000       37       0.7 21.90270
write                    4          0     0.0  0.00000       39       0.7  0.00111
time                    13          0     0.0  0.00000        2       0.0  0.00000
stat                    38          0     0.0  0.00000        4       0.0  0.00018
ioctl                   54          0     0.0  0.00000       12       0.2  0.00030
setpgrp2                82          0     0.0  0.00000        2       0.0  0.00001
sigvector              108          0     0.0  0.00000        8       0.1  0.00004
sigprocmask            185          0     0.0  0.00000      172       3.4  0.00039
waitpid                200          0     0.0  0.00000        4       0.0 19.32314

        Cumulative Interval:       51 secs
█                                                                      Page 1 of 1
```

On the main global screen there is a softkey for ‘Select Process’ . Once you’ve selected a process you can look at the process specific screens.

This screen is useful for spotting unusually high rates of a particular syscall or a syscall that is accumulating a lot of CPU time. This data is typically used in conjunction with a tusc trace or application logfile to make sense of what the high call rate is due to.

IP/UDP/TCP Tools - Glance

- Open files

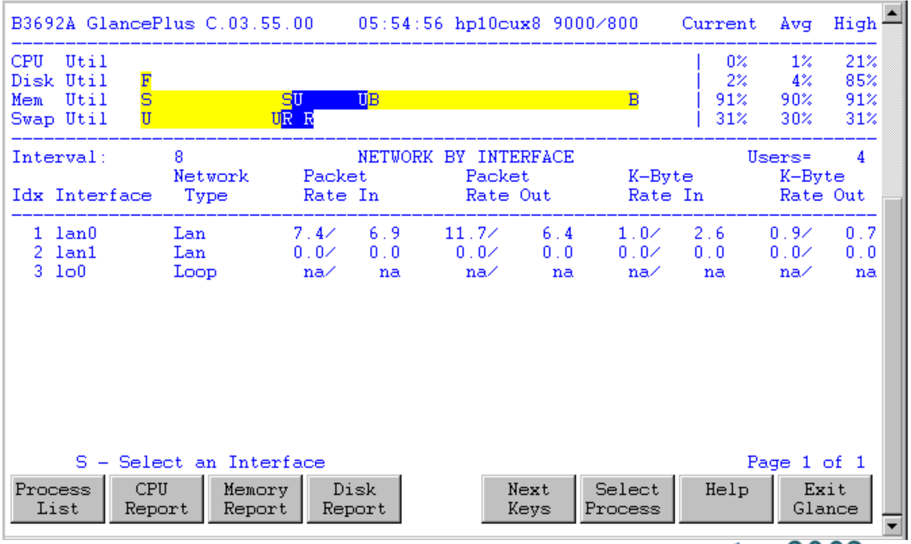


```
B3692A GlancePlus C.03.55.00    05:45:50 hp10cux8 9000/800    Current  Avg  High
--------------------------------------------------------------------------------
CPU  Util                                                     |  1%     1%   21%
Disk Util   F   F                                             | 10%     7%   85%
Mem  Util   S            SU     UB                        B   | 91%    90%   91%
Swap Util   U           UR R                                  | 31%    30%   31%
--------------------------------------------------------------------------------
Open Files PID: 13363, telnetd        PPID:   710 euid:     0 User: root
                                                   Open   Open
FD  File Name                                Type  Mode   Count      Offset
--------------------------------------------------------------------------------
  0 inet,tcp                                 socket rd/wr   3              0
  1 inet,tcp                                 socket rd/wr   3              0
  2 inet,tcp                                 socket rd/wr   3              0
  3 /dev/telnetm                             stream rd/wr   1            101
  4 unix /var/spool/sockets/pwgr/client13363 socket rd/wr   1              0


                                                                  Page 1 of 1
```

If the lsof tool is not available, this screen can be used to map a processes File Descriptors to files/sockets. It does not provide details of the type of socket or the IP/ports associated with it. The tusc tool used with the verbose option will display the IP/port number information as well.

IP/UDP/TCP Tools - Glance

- Network by interface

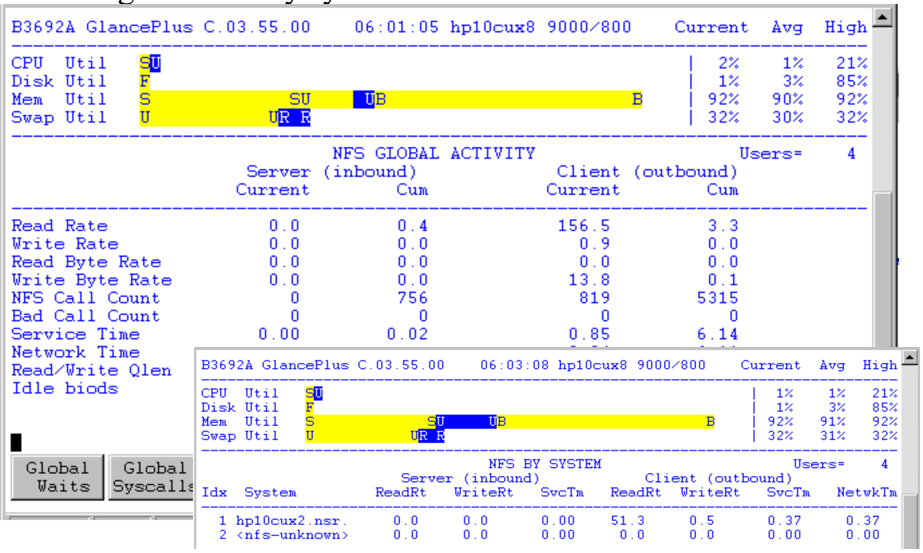


```
B3692A GlancePlus C.03.55.00    05:54:56 hp10cux8 9000/800    Current  Avg  High
--------------------------------------------------------------------------------
CPU  Util                                                    |   0%     1%   21%
Disk Util   F                                                |   2%     4%   85%
Mem  Util   S            SU      UB                        B | 91%    90%   91%
Swap Util   U           UR R                                 | 31%    30%   31%
--------------------------------------------------------------------------------
Interval:      8                NETWORK BY INTERFACE             Users=    4
               Network     Packet          Packet         K-Byte         K-Byte
Idx Interface   Type       Rate In         Rate Out       Rate In        Rate Out
--------------------------------------------------------------------------------
  1 lan0       Lan        7.4/   6.9    11.7/   6.4    1.0/   2.6    0.9/   0.7
  2 lan1       Lan        0.0/   0.0     0.0/   0.0    0.0/   0.0    0.0/   0.0
  3 lo0        Loop        na/    na      na/    na     na/    na     na/    na

      S - Select an Interface                                      Page 1 of 1
```

Unless you really want to use the netstat –in command and do the packet rate calculations manually, this is very useful for overall link throughput/utilization info.

IP/UDP/TCP Tools - Glance

- NFS global and by system



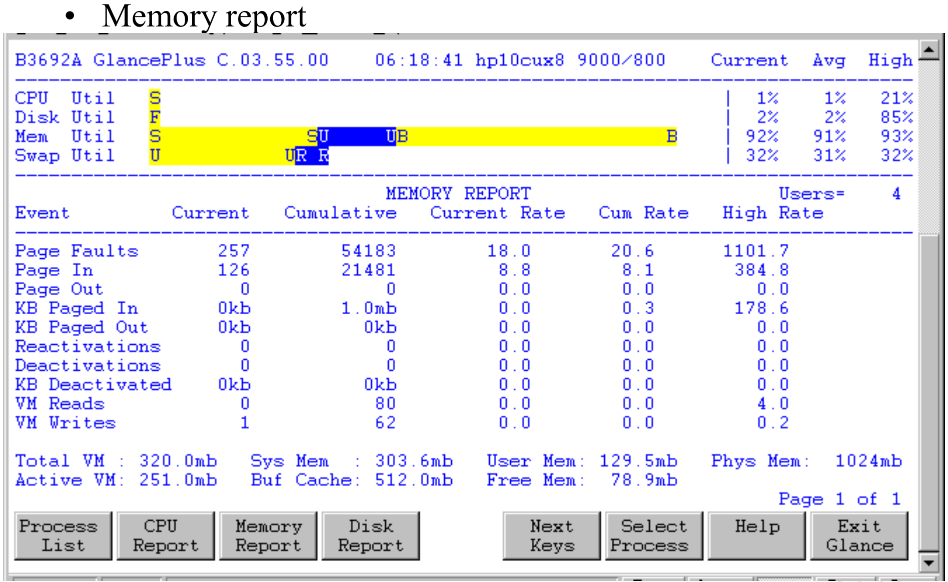
```
B3692A GlancePlus C.03.55.00     06:01:05 hp10cux8 9000/800     Current  Avg  High
----------------------------------------------------------------------------------
CPU  Util    SU                                                  |  2%    1%   21%
Disk Util    F                                                   |  1%    3%   85%
Mem  Util    S             SU    UB                       B      | 92%   90%   92%
Swap Util    U           UR R                                    | 32%   30%   32%
----------------------------------------------------------------------------------
                              NFS GLOBAL ACTIVITY                Users=     4
                      Server (inbound)            Client (outbound)
                     Current        Cum          Current         Cum
----------------------------------------------------------------------------------
Read Rate               0.0         0.4            156.5         3.3
Write Rate              0.0         0.0              0.9         0.0
Read Byte Rate          0.0         0.0              0.0         0.0
Write Byte Rate         0.0         0.0             13.8         0.1
NFS Call Count            0         756              819        5315
Bad Call Count            0           0                0           0
Service Time           0.00        0.02             0.85        6.14
Network Time
Read/Write Qlen
Idle biods
```

```
B3692A GlancePlus C.03.55.00     06:03:08 hp10cux8 9000/800     Current  Avg  High
----------------------------------------------------------------------------------
CPU  Util    SU                                                  |  1%    1%   21%
Disk Util    F                                                   |  1%    3%   85%
Mem  Util    S             SU     UB                      B      | 92%   91%   92%
Swap Util    U           UR R                                    | 32%   31%   32%
----------------------------------------------------------------------------------
                              NFS BY SYSTEM                      Users=     4
                      Server (inbound)             Client (outbound)
Idx  System          ReadRt   WriteRt   SvcTm   ReadRt  WriteRt   SvcTm   NetwkTm
----------------------------------------------------------------------------------
  1 hp10cux2.nsr.      0.0      0.0      0.00     51.3     0.5      0.37     0.37
  2 <nfs-unknown>      0.0      0.0      0.00      0.0     0.0      0.00     0.00
```

A good client or server wide view of NFS traffic rates and client side service times. It does not break it down into individual file systems mounted.

IP/UDP/TCP Tools - Glance

- Memory report



```
B3692A GlancePlus C.03.55.00    06:18:41 hp10cux8 9000/800    Current  Avg  High
--------------------------------------------------------------------------------
CPU  Util   S                                                  |  1%    1%   21%
Disk Util   F                                                  |  2%    2%   85%
Mem  Util   S             SU      UB                      B    | 92%   91%   93%
Swap Util   U           UR R                                   | 32%   31%   32%
--------------------------------------------------------------------------------
                                 MEMORY REPORT                      Users=    4
Event             Current   Cumulative   Current Rate   Cum Rate   High Rate
--------------------------------------------------------------------------------
Page Faults           257        54183         18.0        20.6       1101.7
Page In               126        21481          8.8         8.1        384.8
Page Out                0            0          0.0         0.0          0.0
KB Paged In           0kb        1.0mb          0.0         0.3        178.6
KB Paged Out          0kb          0kb          0.0         0.0          0.0
Reactivations           0            0          0.0         0.0          0.0
Deactivations           0            0          0.0         0.0          0.0
KB Deactivated        0kb          0kb          0.0         0.0          0.0
VM Reads                0           80          0.0         0.0          4.0
VM Writes               1           62          0.0         0.0          0.2

Total VM : 320.0mb   Sys Mem   : 303.6mb   User Mem: 129.5mb   Phys Mem:  1024mb
Active VM: 251.0mb   Buf Cache: 512.0mb   Free Mem:  78.9mb
                                                                    Page 1 of 1
```

Since buffercache can be fixed or dynamic, this is a quick way to see the exact usage. NFS clients are limited to only using 25% of buffercache for NFS mounted file access. As a result, any NFS copy/performance testing using large files needs to take this into consideration. NFS client performance begins to drop off if the 25% limit is constantly being hit.

The Page Out rates and the Deactivations indicate memory pressure.

IP/UDP/TCP Tools - ttcp

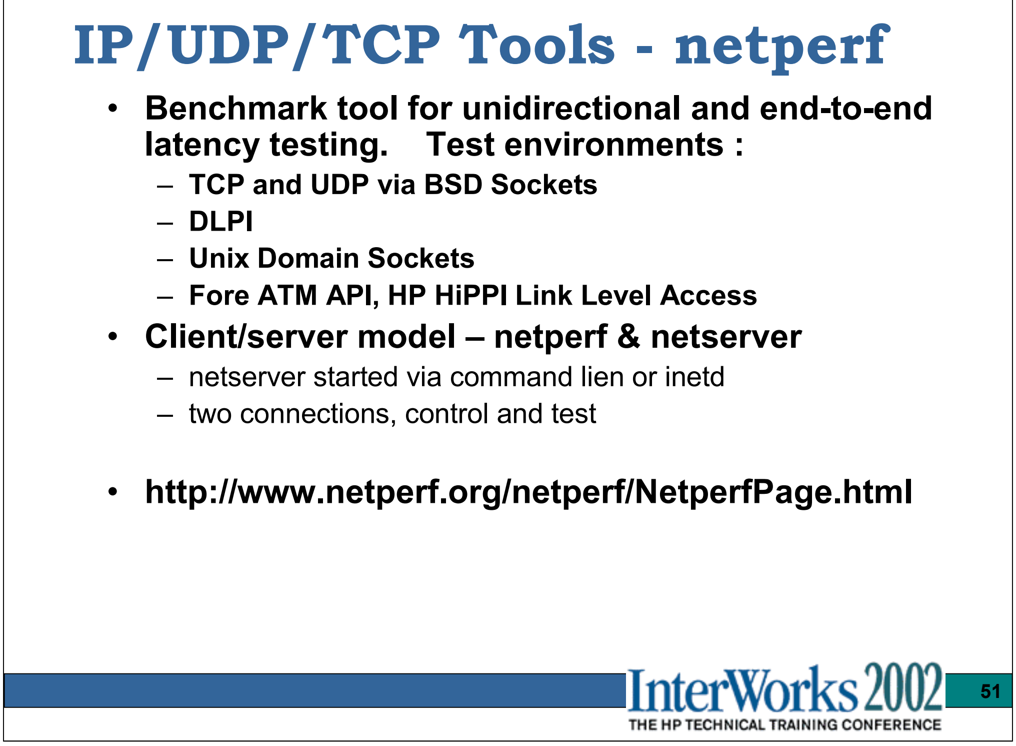
- Test TCP (TTCP) is a command-line sockets-based benchmarking tool for measuring TCP and UDP performance between two systems.
- simpler than netperf, UDP/TCP only
- public domain and copies avail at a number of anon ftp sites.
- typical usage:

  ttcp –stp9 <host>

  - sends 2048 8k buffers to TCP port 9 (inetd’s discard port) on target host.
  - can be run in server mode if no discard port available
- No use of file system buffercache to skew results

```
Usage: ttcp -t [-options] host [ < in ]
       ttcp -r [-options > out]
Common options:
        -l ##   length of bufs read from or written to network (default 8192)
        -u      use UDP instead of TCP
        -p ##   port number to send to or listen at (default 5001)
        -s      -t: source a pattern to network
                -r: sink (discard) all data from network
        -A      align the start of buffers to this modulus (default 16384)
        -O      start buffers at this offset from the modulus (default 0)
        -v      verbose: print more statistics
        -d      set SO_DEBUG socket option
        -b ##   set socket buffer size (if supported)
        -f X    format for rate: k,K = kilo{bit,byte}; m,M = mega; g,G = giga
Options specific to -t:
        -n##    number of source bufs written to network (default 2048)
        -D      don't buffer TCP writes (sets TCP_NODELAY socket option)
Options specific to -r:
        -B      for -s, only output full blocks as specified by -l (for TAR)
        -T      "touch": access each byte as it's read
```

```
# ./ttcp -stp9 15.24.46.27
ttcp-t: buflen=8192, nbuf=2048, align=16384/0, port=9  tcp  -> 15.24.46.27
ttcp-t: socket
ttcp-t: connect
ttcp-t: 16777216 bytes in 14.18 real seconds = 1155.46 KB/sec +++
ttcp-t: 2048 I/O calls, msec/call = 7.09, calls/sec = 144.43
ttcp-t: 0.0user 0.0sys 0:14real 0% 0i+47d 25maxrss 0+1pf 512+7csw
```



IP/UDP/TCP Tools - netperf

- **Benchmark tool for unidirectional and end-to-end latency testing. Test environments :**
  - **TCP and UDP via BSD Sockets**
  - **DLPI**
  - **Unix Domain Sockets**
  - **Fore ATM API, HP HiPPI Link Level Access**
- **Client/server model – netperf & netserver**
  - netserver started via command lien or inetd
  - two connections, control and test
- **http://www.netperf.org/netperf/NetperfPage.html**

Running server side as a standalone daemon:

netserver -p portnum Listen for connect requests on portnum.

Or to have inetd invoke add this line to the /etc/services file:

```
netperf 12865/tcp
```

Then add this line to the /etc/inetd.conf file:

```
netperf stream tcp nowait root /opt/netperf/netserver netserver
```

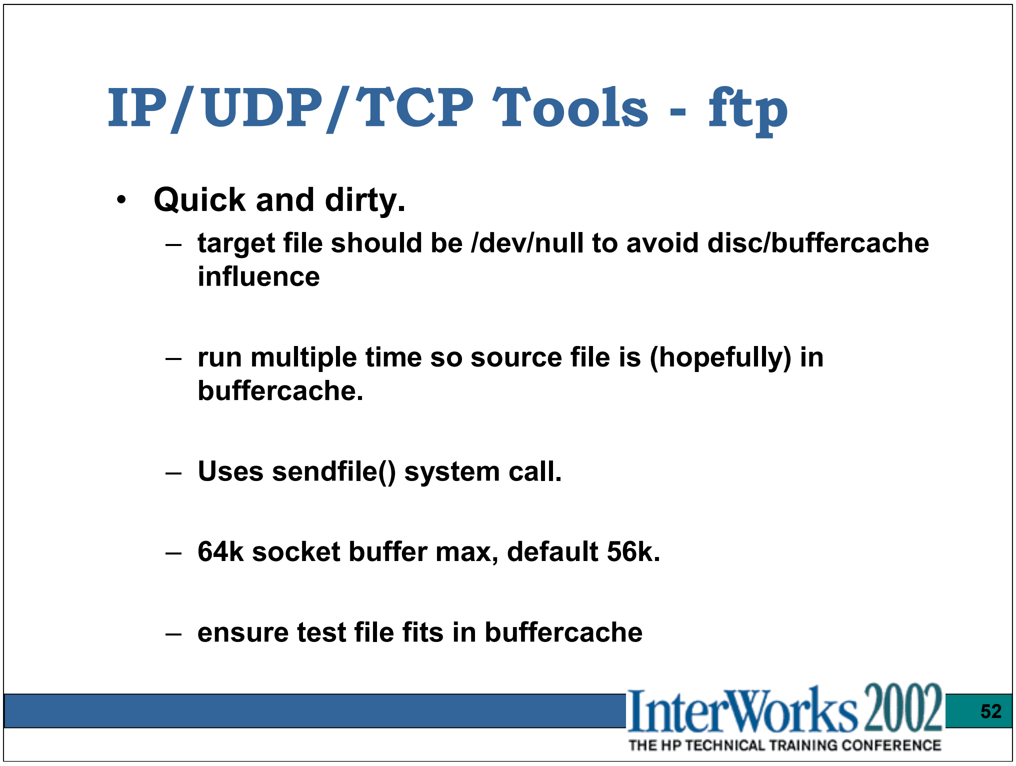
netperf sample using defaults

Default test is 10 seconds. The socket buffers on ether end will be sized at system defaults. All TCP options (e.g. TCP_NODELAY) will be at defaults. The simple test is performed by entering the following commands:
on server system: (or have inetd configured) # ./netserver &
Starting netserver at port 12865
on client system:
./netperf -H 10.123.123.6 .

```
TCP STREAM TEST to 10.123.123.6

Recv      Send      Send
Socket    Socket    Message   Elapsed
Size      Size      Size      Time      Throughput
bytes     bytes     bytes     secs      10^6bits/sec

32768     32768     32768     10.01     69.31
```

•If the test purpose is to investigate problems that arise from high latency of transport, driver, and card combinations, then NETPERF is the appropriate tool to use, especially the Request/Response group of Netperf tests, are suitable for this purpose; because such latency problems would be masked by large socket buffers.

•Moreover, Netperf's DLPI tests are ideal for stressing the driver at a lower level, thus they can be used for testing Ethernet (or LAN Emulation).

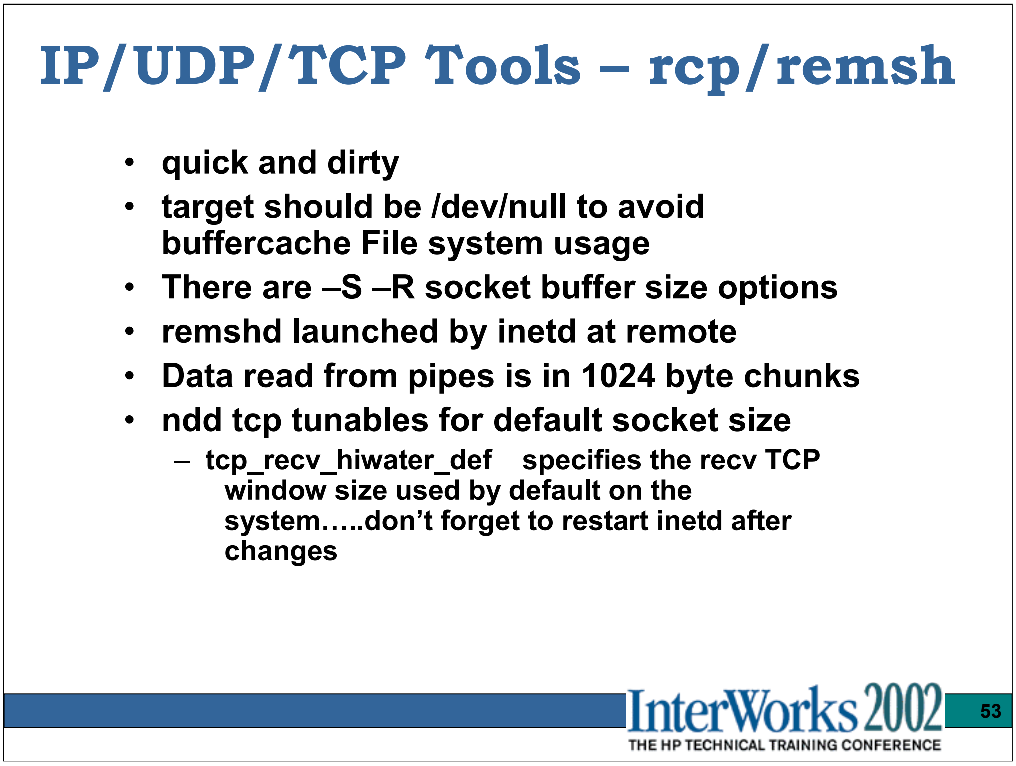


IP/UDP/TCP Tools - ftp

- Quick and dirty.
  - target file should be /dev/null to avoid disc/buffercache influence
  - run multiple time so source file is (hopefully) in buffercache.
  - Uses sendfile() system call.
  - 64k socket buffer max, default 56k.
  - ensure test file fits in buffercache

Guaranteed to be on every HP system plus most other vendors with real operating systems.

Hash marks to observe packet retransmits etc.

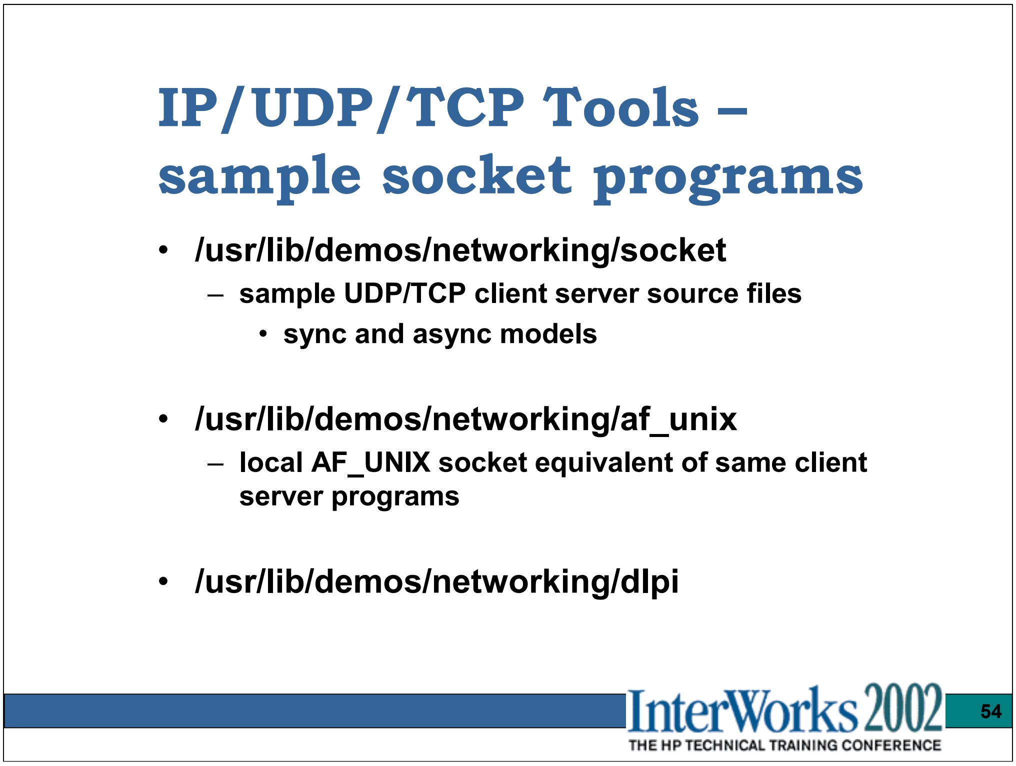


IP/UDP/TCP Tools – rcp/remsh

- quick and dirty
- target should be /dev/null to avoid buffercache File system usage
- There are –S –R socket buffer size options
- remshd launched by inetd at remote
- Data read from pipes is in 1024 byte chunks
- ndd tcp tunables for default socket size
  - tcp_recv_hiwater_def specifies the recv TCP window size used by default on the system…..don't forget to restart inetd after changes

InterWorks 2002
THE HP TECHNICAL TRAINING CONFERENCE

More buffer size flexibility than ftp, but if you're going to bother with the –S –R options just go get ttcp.

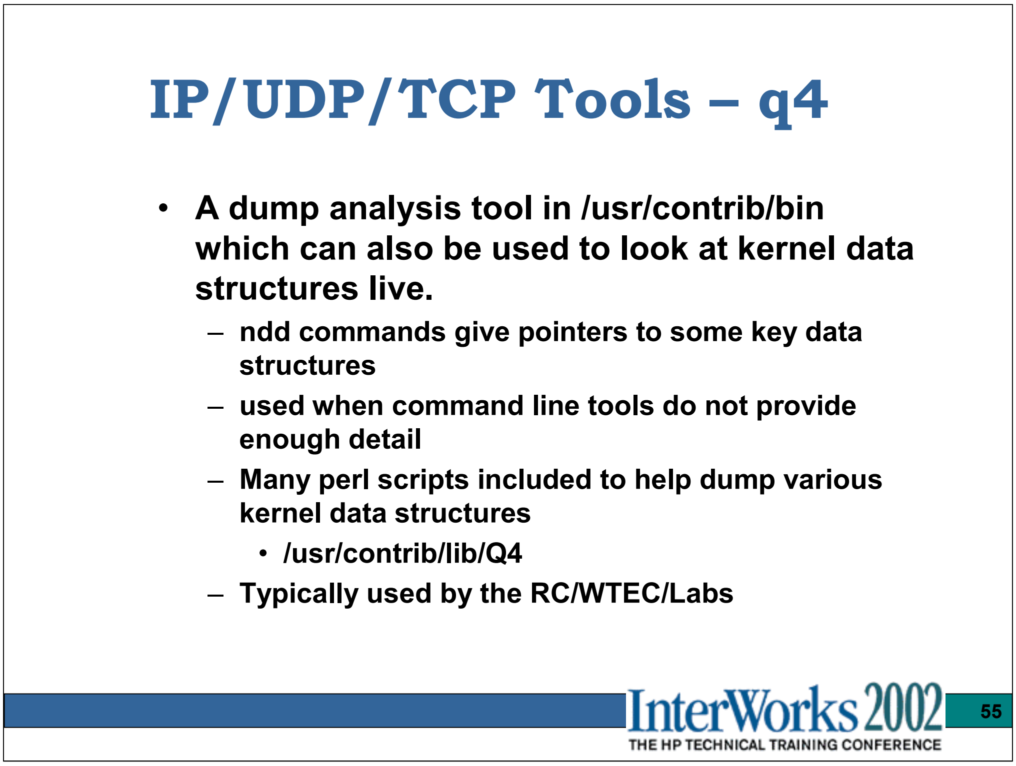


IP/UDP/TCP Tools – sample socket programs

- /usr/lib/demos/networking/socket
  - sample UDP/TCP client server source files
    - sync and async models
- /usr/lib/demos/networking/af_unix
  - local AF_UNIX socket equivalent of same client server programs
- /usr/lib/demos/networking/dlpi

Great for playing with various TCP/UDP socket options, TCP behaviors, hostname lookups,and investigating all kinds of ‘how does this really work’ scenarios.

Typically requires adding the ‘example’ service name to the /etc/services file.



IP/UDP/TCP Tools – q4

- A dump analysis tool in /usr/contrib/bin which can also be used to look at kernel data structures live.
  - ndd commands give pointers to some key data structures
  - used when command line tools do not provide enough detail
  - Many perl scripts included to help dump various kernel data structures
    - /usr/contrib/lib/Q4
  - Typically used by the RC/WTEC/Labs

Q4 is intended primarily as a dump analysis tool, but can be used on a live system. A few simple tasks that provide useful info:

Get a look at a the kernel stack trace for processes:

```
q4> load struct proc from proc_list next p_factp max 3000
    loaded 139 struct procs as a linked list (stopped by null pointer)
q4> print –tx > proc_structures.out
q4> trace –u pile > proc_stacktrace_with_args.out
q4> trace –v pile > proc_stacktrace_with_stkptrs.out
q4> trace pile  > proc_stacktrace_plain.out
```

Select one process to look at in more detail:

```
q4> keep p_pid == 740
kept 1 of 139 struct proc's, discarded 138
q4> load struct kthread from p_firstthreadp next kt_nextp max 40
loaded 3 struct kthreads as a linked list (stopped by null pointer)
q4> print -tx | grep last
        kt_lastrun_time  0x6d64fd
        kt_lastrun_time  0x6d6503
        kt_lastrun_time  0x6d64ff
q4> ticks_since_boot
033745712       7326666 0x6fcbca        tick = 0.010 seconds usually
```

NFS client/server

- typical performance related issues
  - Biod tuning
  - number of nfsds
  - PV3 vs. PV2 TCP vs UDP
  - Automount (legacy vs. autofs)
    - autofs and LOFS
  - Buffercache
  - HPUX 11.0 vs. 11.11 and beyond
- Tools
  - nfsstat, nettl, rpcinfo, deamon logging AND….
  - Session Number 001 - NFS Performance Tuning for HP-UX 11.0 and 11i Systems by Dave Olker
  - Optimizing NFS Performance: Tuning and Troubleshooting NFS on HP-UX Systems by David Olker

InterWorks 2002
THE HP TECHNICAL TRAINING CONFERENCE

The biods are the processes that handle the client read/write requests and mange the read-ahead function. How many to use and how efficient/fair they are in serving client requests is completely dependant on the client read/write usage profile.

The number of nfsds is less critical…just have enough of them There are cases where adding more will just end up with more waiting on the same resource, adding to contention for that resource…i.e., all nfsds accessing files in a huge directory waiting for the directory inode lock. the ps –elf | grep nfsd will show a common wait channel.

Be aware of PV2 and PV3 read/write size differences, async vs sync differences, and the transports used.

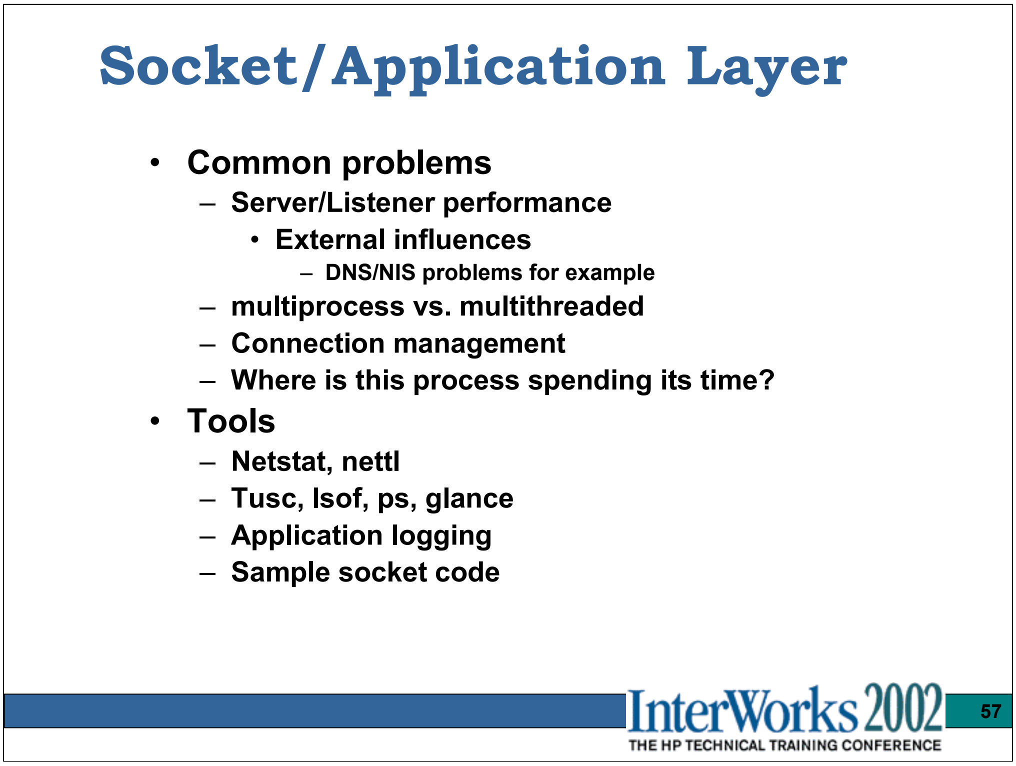
Automount maps…direct=good, indirect=ok, hierarchical=easy/lazy/bad. Inactivity unmount timer setting should be raised from 5minute default unless maps change frequently.

Buffercache is a concern for HPUX client code primarily. For a pure NFS server, more buffercache is better. the NFS client code will limit itself to 25% of buffercache. Due to this, the tendency is to increase buffercache size for the client-sides sake, there is significant overhead incurred in some of the client-side buffercache management routines. It's a try-and-see iterative balancing act.

HPUX 11.0 NFS client code makes extensive use of the File System Alpha semaphore and can be a performance/scalability bottleneck. HPUX 11.11+ has replaced this with multiple spinlocks. This has greatly improved client performance and scalability on HPUX 11.11+.

http://h21007.www2.hp.com/dspp/files/unprotected/devresource/Docs/Presentations/NFSperf.pdf

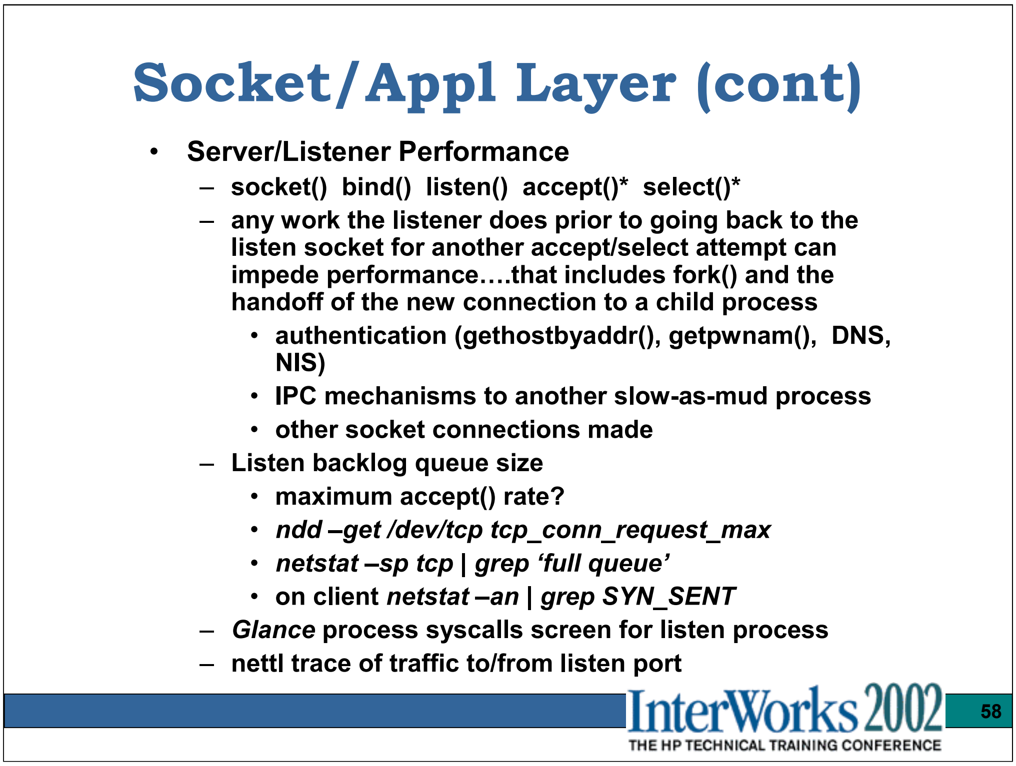
An excellent reference for this vast subject.



Socket/Application Layer

- Common problems
  - Server/Listener performance
    - External influences
      - DNS/NIS problems for example
  - multiprocess vs. multithreaded
  - Connection management
  - Where is this process spending its time?
- Tools
  - Netstat, nettl
  - Tusc, lsof, ps, glance
  - Application logging
  - Sample socket code

Some of these tools have been previously discussed, but they will be mentioned here again with specific reference to features that can be useful in the socket/application layer troubleshooting.



Socket/Appl Layer (cont)

- **Server/Listener Performance**
  - **socket() bind() listen() accept()* select()***
  - **any work the listener does prior to going back to the listen socket for another accept/select attempt can impede performance….that includes fork() and the handoff of the new connection to a child process**
    - **authentication (gethostbyaddr(), getpwnam(), DNS, NIS)**
    - **IPC mechanisms to another slow-as-mud process**
    - **other socket connections made**
  - **Listen backlog queue size**
    - **maximum accept() rate?**
    - ***ndd –get /dev/tcp tcp_conn_request_max***
    - ***netstat –sp tcp | grep ‘full queue’***
    - **on client *netstat –an | grep SYN_SENT***
  - ***Glance* process syscalls screen for listen process**
  - **nettl trace of traffic to/from listen port**

Some of these tools have been previously discussed, but they will be mentioned here again with specific reference to features that can be useful in the socket/application layer troubleshooting.

Typically a multiprocess model will do non-blocking accept()s with select() to check for a readable listen socket…a new connection. It forks a child process to do the accept(), and returns to the accept()/select() polling loop.

See sample server.tcp.c code in ***/usr/lib/demos/networking/socket.***

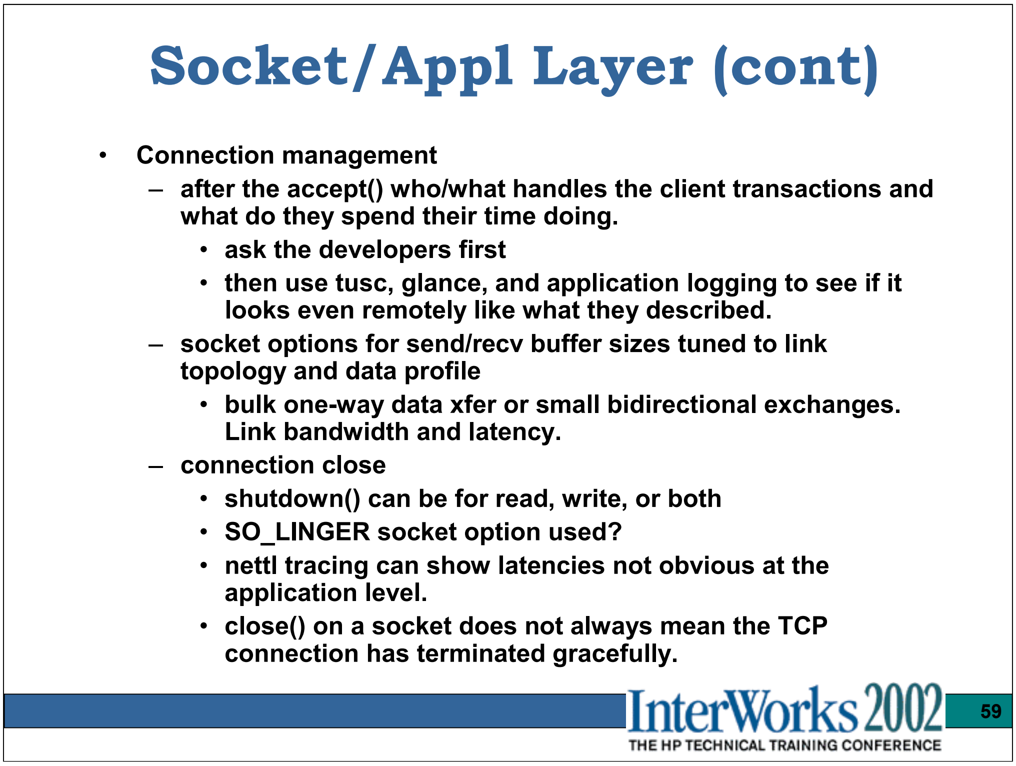
System tunable maxfiles and maxfiles_lim can be hit when large number of connections are being created

Tools for observing the listen process to see what it’s up to:

tusc, glance, nettl, and the listen processes own ***application level logfile*** which had better be there or you need a rebate from the developer for the incomplete job he/she/it did for you.

tusc – Trace Unix System Call. A wonderful utility to trace process system calls with arguments, call timing, return values, etc. Invaluable.

Debugging is almost always best done at the highest layer possible.



Socket/Appl Layer (cont)

- Connection management
  - after the accept() who/what handles the client transactions and what do they spend their time doing.
    - ask the developers first
    - then use tusc, glance, and application logging to see if it looks even remotely like what they described.
  - socket options for send/recv buffer sizes tuned to link topology and data profile
    - bulk one-way data xfer or small bidirectional exchanges. Link bandwidth and latency.
  - connection close
    - shutdown() can be for read, write, or both
    - SO_LINGER socket option used?
    - nettl tracing can show latencies not obvious at the application level.
    - close() on a socket does not always mean the TCP connection has terminated gracefully.

The goal here is to identify what the application/processes are **suppose** to be doing, then use tools to trace/time/profile where the time is being spent.

If a critical area of behavior is suspect, it is sometimes possible to use the sample socket code in **/usr/lib/demos/networking/socket** to simulate the key mechanisms and simulate the problem outside a production environment.

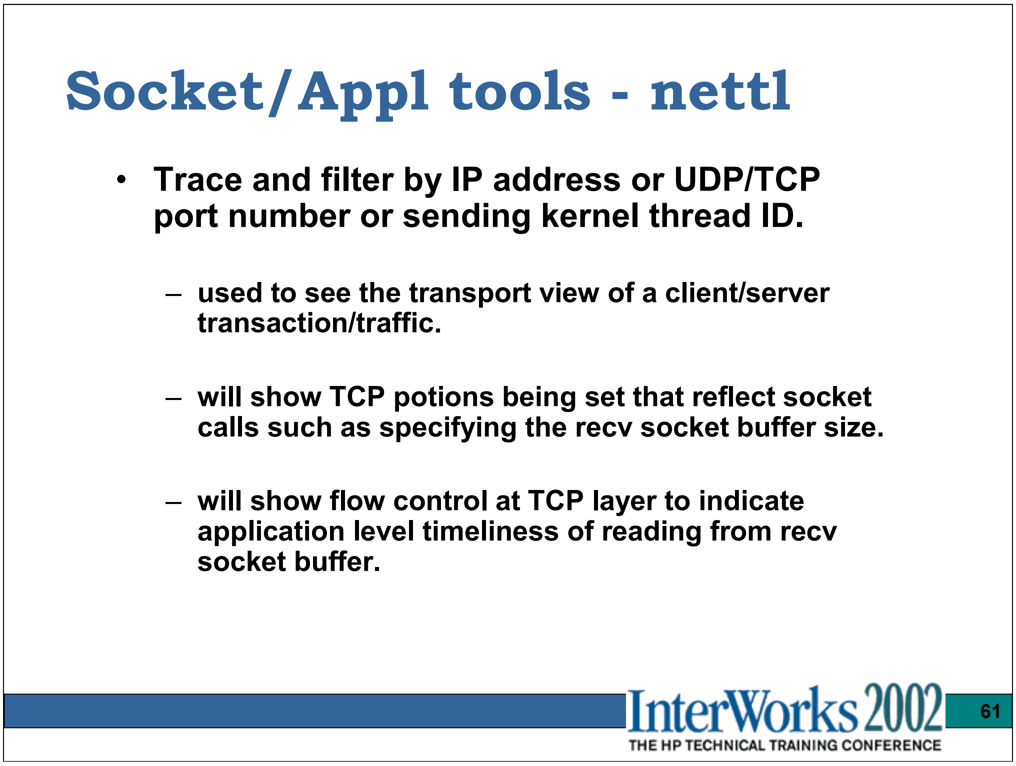
Socket/Appl tools - netstat

- Connection states via *netstat -an*
  - SYN_SENT
    - trying to contact remote host….either host is not up or the listen queue is full….otherwise you would have seen a RESET packet
  - ESTABLISHED
    - normal state for active TCP connection
  - CLOSE_WAIT
    - FIN received from remote end and waiting for local process owning the port to issue a close.
  - FIN_WAIT2
    - FIN sent to remote, and ACK'ed, but remote process has not closed his end of connection…the partner state to CLOSE_WAIT on the remote host.
  - TIME_WAIT
    - 60 second state after both ends close gracefully

Used in conjunction with lsof to find processes owning connections of interest.

Once local process owning port is ID'ed tusc or glance can be used to see what it is doing.

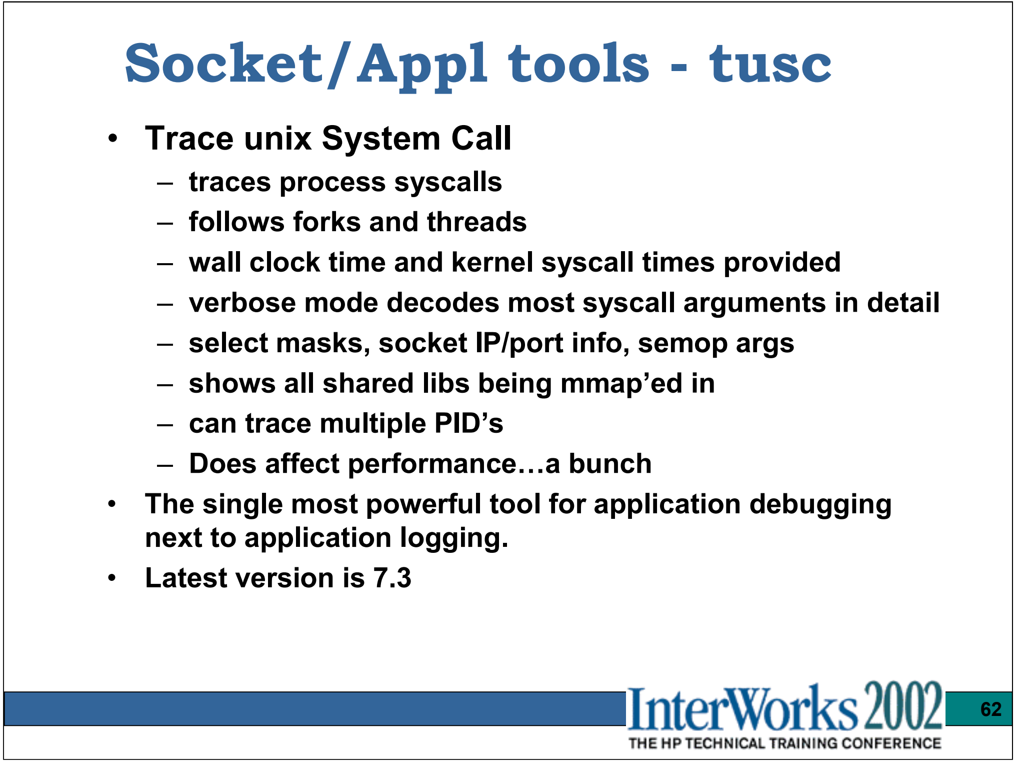
Then the application developer can tell you why it is doing it.



Socket/Appl tools - nettl

- Trace and filter by IP address or UDP/TCP port number or sending kernel thread ID.
  - used to see the transport view of a client/server transaction/traffic.
  - will show TCP potions being set that reflect socket calls such as specifying the recv socket buffer size.
  - will show flow control at TCP layer to indicate application level timeliness of reading from recv socket buffer.

A prime example of nettl offering a little different view of things is the case of the client SYN packet being accepted immediately by the transport, and the client starting to send data….this can happen immediately yet it does not mean the listener has done the accept() yet. A listener side logging mechanism may not show any latency in such a connection, but to the client, it sees the entire delay.



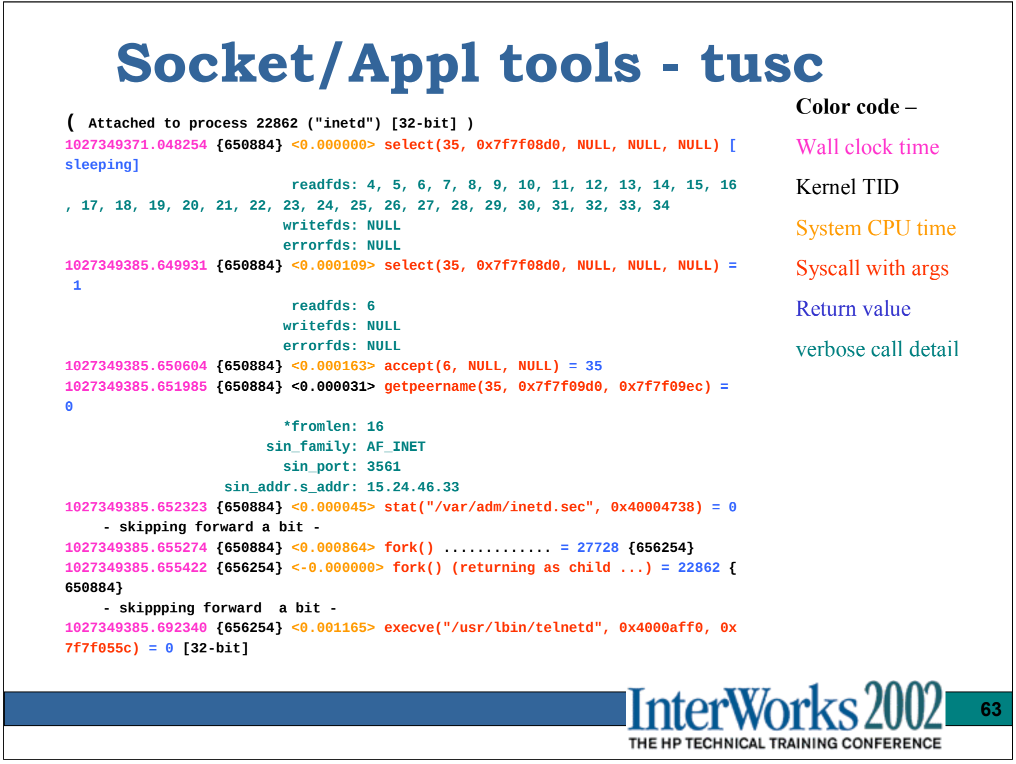
Socket/Appl tools - tusc

- Trace unix System Call
  - traces process syscalls
  - follows forks and threads
  - wall clock time and kernel syscall times provided
  - verbose mode decodes most syscall arguments in detail
  - select masks, socket IP/port info, semop args
  - shows all shared libs being mmap’ed in
  - can trace multiple PID’s
  - Does affect performance…a bunch
- The single most powerful tool for application debugging next to application logging.
- Latest version is 7.3

http://ftp.au.freebsd.org/pub/hpux/Sysadmin/tusc-7.3 or any freeBSD.org mirror ftp site

Since many services (telnet, remshd, etc) are forked from inetd (the listener) to trace such processes you will need to trace the inetd process with the ‘-f’ option which follows the process forks.

```
ps -ef | grep inetd
    root 27708 22348  1 08:47:22 pts/tb    0:00 grep inetd
    root 22862     1  0 04:05:19 ?         0:00 inetd
# /tmp/tusc -flv -ccc -T "" -o /tmp/tusc.out 22862
<cntl-C>
( Detaching from process 27729 ("login -h c2410c33.nsr.hp.com -p") )
( Detaching from process 27728 ("telnetd") )
( Detaching from process 22862 ("inetd") )
#
```



Socket/Appl tools - tusc

```
( Attached to process 22862 ("inetd") [32-bit] )
1027349371.048254 {650884} <0.000000> select(35, 0x7f7f08d0, NULL, NULL, NULL) [
sleeping]
                           readfds: 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16
, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34
                          writefds: NULL
                          errorfds: NULL
1027349385.649931 {650884} <0.000109> select(35, 0x7f7f08d0, NULL, NULL, NULL) =
 1
                           readfds: 6
                          writefds: NULL
                          errorfds: NULL
1027349385.650604 {650884} <0.000163> accept(6, NULL, NULL) = 35
1027349385.651985 {650884} <0.000031> getpeername(35, 0x7f7f09d0, 0x7f7f09ec) =
0
                          *fromlen: 16
                        sin_family: AF_INET
                          sin_port: 3561
                   sin_addr.s_addr: 15.24.46.33
1027349385.652323 {650884} <0.000045> stat("/var/adm/inetd.sec", 0x40004738) = 0
    - skipping forward a bit -
1027349385.655274 {650884} <0.000864> fork() ............ = 27728 {656254}
1027349385.655422 {656254} <-0.000000> fork() (returning as child ...) = 22862 {
650884}
    - skippping forward  a bit -
1027349385.692340 {656254} <0.001165> execve("/usr/lbin/telnetd", 0x4000aff0, 0x
7f7f055c) = 0 [32-bit]
```

Color code –

Wall clock time

Kernel TID

System CPU time

Syscall with args

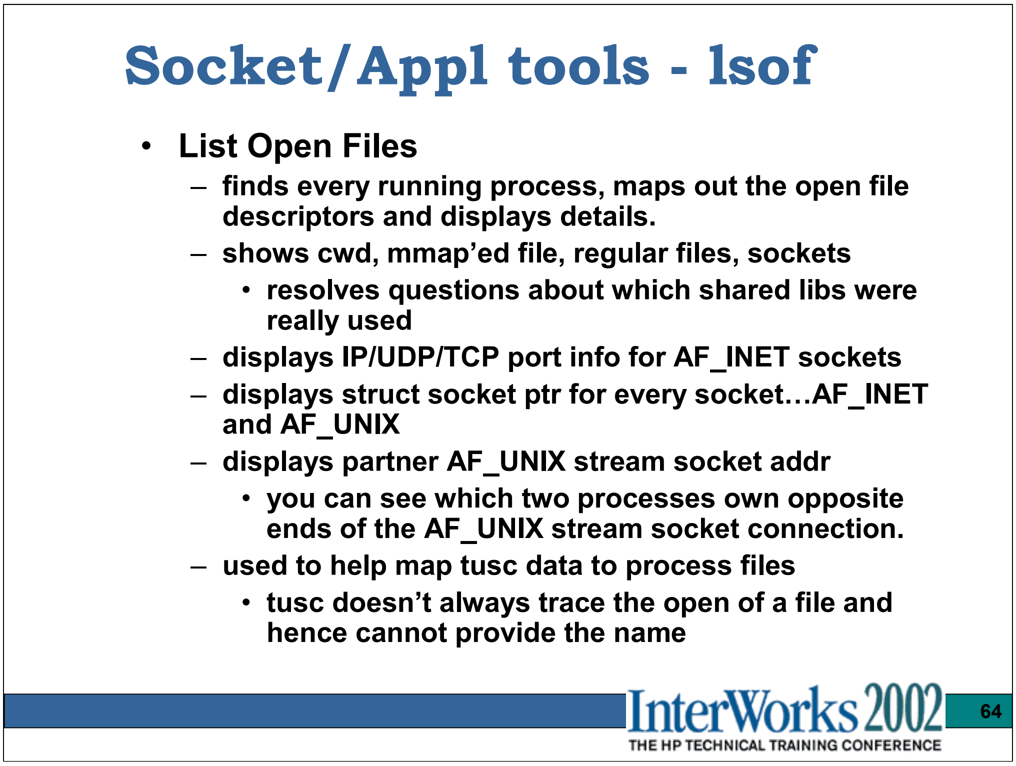
Return value

verbose call detail

Syscall trace entries are written to the output file at syscall completion time. the ‘ –E’ option will log a trace file entry upon syscall entry and exit. Without the –E option, it is hard to know what wall clock time was due to user space CPU time vs. system call duration. For non-blocking system calls, the system cpu time provided will be the bulk of the wall clock time for that syscall.

Application logging is often the only way to know for sure what is being done in user space without attaching with a real debugger live and trying to extract a stack trace from the process.

Having source code for the process being traced is a _big_ help in understanding exactly what’s going on.



Socket/Appl tools - lsof

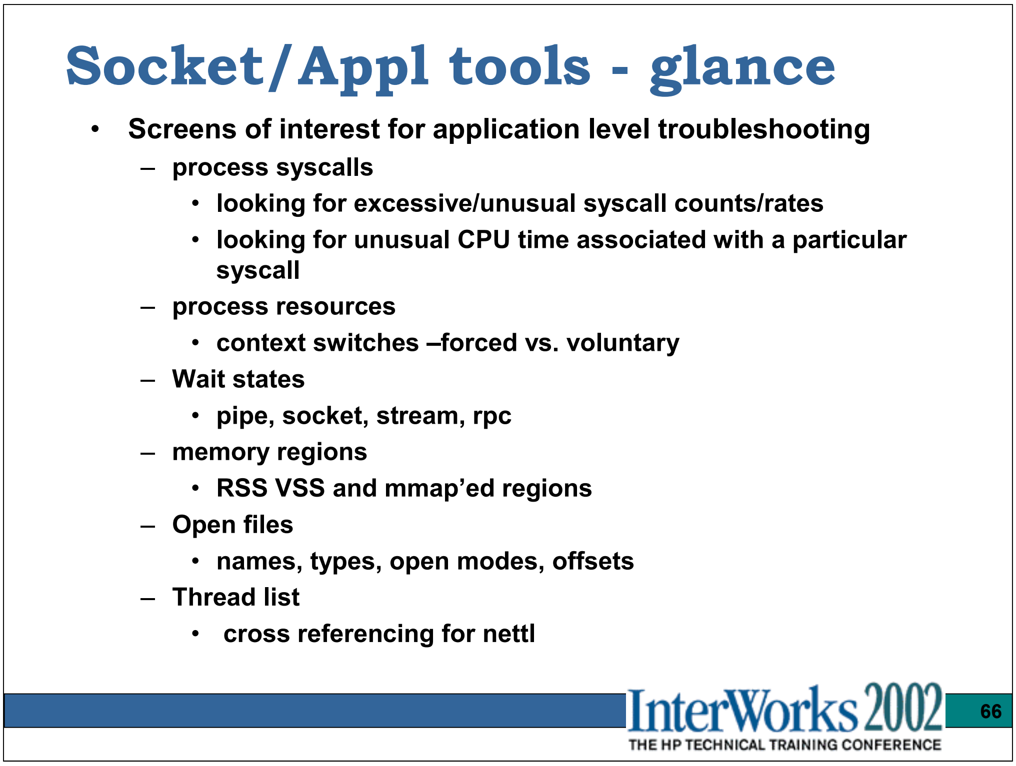
- List Open Files
  - finds every running process, maps out the open file descriptors and displays details.
  - shows cwd, mmap’ed file, regular files, sockets
    - resolves questions about which shared libs were really used
  - displays IP/UDP/TCP port info for AF_INET sockets
  - displays struct socket ptr for every socket…AF_INET and AF_UNIX
  - displays partner AF_UNIX stream socket addr
    - you can see which two processes own opposite ends of the AF_UNIX stream socket connection.
  - used to help map tusc data to process files
    - tusc doesn’t always trace the open of a file and hence cannot provide the name

```
# ./lsof -n -p 22862 | more
COMMAND    PID USER   FD   TYPE       DEVICE SIZE/OFF  NODE NAME
inetd    22862 root  cwd    DIR       64,0x5     3072     2 /tmp
inetd    22862 root  txt    REG       64,0x6    65536 22253 /usr/sbin/inetd
inetd    22862 root  mem    REG       64,0x6    16384 14017 /usr/lib/libstraddr.1
inetd    22862 root  mem    REG       64,0x6    45056 13967 /usr/lib/libnss_files.
1
inetd    22862 root  mem    REG       64,0x6    45056   165 /usr/lib/libnss_nis.1
inetd    22862 root  mem    REG       64,0x6   135168   118 /usr/lib/libxti.2
inetd    22862 root  mem    REG       64,0x6   724992   120 /usr/lib/libnsl.1
inetd    22862 root  mem    REG       64,0x6   282624 13948 /usr/lib/libm.2
inetd    22862 root  mem    REG       64,0x6   151552 14001 /usr/lib/libsec.2
inetd    22862 root  mem    REG       64,0x6    24576 13903 /usr/lib/libdld.2
inetd    22862 root  mem    REG       64,0x6  1843200 13855 /usr/lib/libc.2
inetd    22862 root  mem    REG       64,0x6   155648 13586 /usr/lib/dld.sl
inetd    22862 root    0r   DIR       64,0x3     1024     2 /
inetd    22862 root    1r   DIR       64,0x3     1024     2 /
inetd    22862 root    2r   DIR       64,0x3     1024     2 /
inetd    22862 root    3u  FIFO   0x41d5d048      0t0   201
inetd    22862 root    4u  inet   0x426d2280      0t0   UDP *:50474 (Idle)
inetd    22862 root    5u  inet   0x41f5d4c0      0t0   TCP *:ftp (LISTEN)
inetd    22862 root    6u  inet   0x41f5d640      0t0   TCP *:telnet (LISTEN)
```

Socket/Appl tools – ps -elf

- Maps process names to pids for cross reference with other tools
- The wait channel
  - address/token passed to sleep()/sleep_one()
- The proc structure address
- incore memory image size
  - data, text, stack
- total CPU execution time

It’s handy to have a ps –elf listing(s) when other tools are used since process names are not always available and the processes themselves are transitory.



Socket/Appl tools - glance

- Screens of interest for application level troubleshooting
  - process syscalls
    - looking for excessive/unusual syscall counts/rates
    - looking for unusual CPU time associated with a particular syscall
  - process resources
    - context switches –forced vs. voluntary
  - Wait states
    - pipe, socket, stream, rpc
  - memory regions
    - RSS VSS and mmap'ed regions
  - Open files
    - names, types, open modes, offsets
  - Thread list
    - cross referencing for nettl

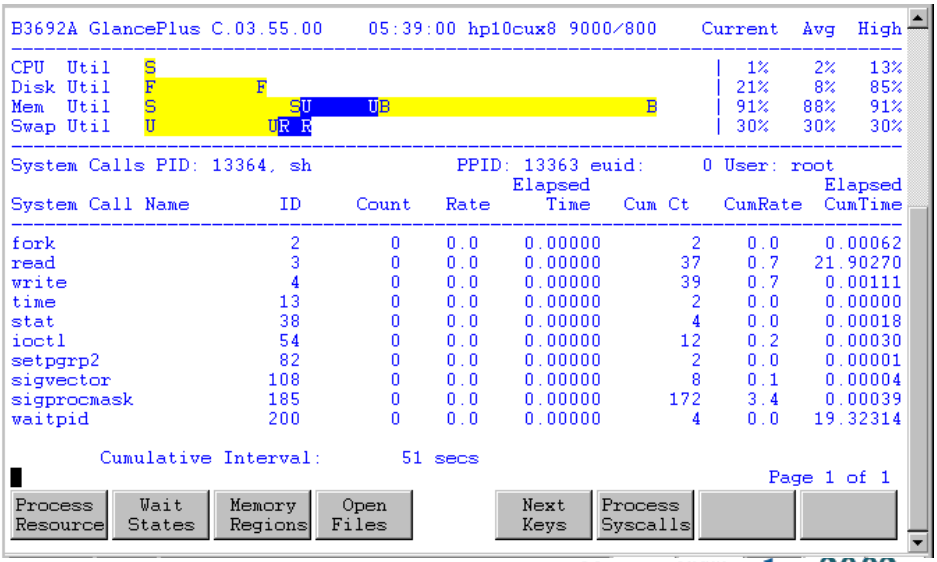
If the process syscalls indicates an unusual rate or count of a particular syscall you can always use tusc to get more detailed information.

Memory allocation requests can be tracked by looking at the 'brk( )' system call with tusc. The brk( ) syscall is used to extend the process data stack area.

Open files screen is the next best thing to lsof.

Socket/Appl Tools - Glance

- Process syscalls



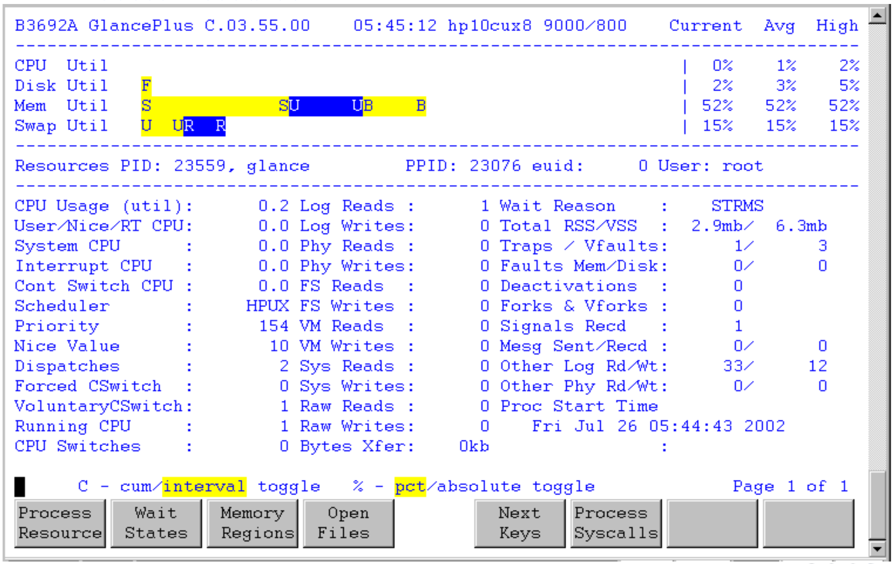
```
B3692A GlancePlus C.03.55.00     05:39:00 hp10cux8 9000/800     Current  Avg  High
--------------------------------------------------------------------------------
CPU  Util   S                                                  |  1%    2%   13%
Disk Util   F         F                                        | 21%    8%   85%
Mem  Util   S           SU     UB                        B     | 91%   88%   91%
Swap Util   U           UR R                                   | 30%   30%   30%
--------------------------------------------------------------------------------
System Calls PID: 13364, sh             PPID: 13363 euid:     0 User: root
                                             Elapsed                      Elapsed
System Call Name         ID     Count    Rate     Time   Cum Ct   CumRate  CumTime
--------------------------------------------------------------------------------
fork                      2         0     0.0  0.00000        2       0.0  0.00062
read                      3         0     0.0  0.00000       37       0.7 21.90270
write                     4         0     0.0  0.00000       39       0.7  0.00111
time                     13         0     0.0  0.00000        2       0.0  0.00000
stat                     38         0     0.0  0.00000        4       0.0  0.00018
ioctl                    54         0     0.0  0.00000       12       0.2  0.00030
setpgrp2                 82         0     0.0  0.00000        2       0.0  0.00001
sigvector               108         0     0.0  0.00000        8       0.1  0.00004
sigprocmask             185         0     0.0  0.00000      172       3.4  0.00039
waitpid                 200         0     0.0  0.00000        4       0.0 19.32314

        Cumulative Interval:      51 secs
                                                                    Page 1 of 1
```

On the main global screen there is a softkey for 'Select Process' . Once you've selected a process you can look at the process specific screens.

This screen is useful for spotting unusually high rates of a particular syscall or a syscall that is accumulating a lot of CPU time. This data is typically used in conjunction with a tusc trace or application logfile to make sense of what the high call rate is due to.

Socket/Appl Tools - Glance

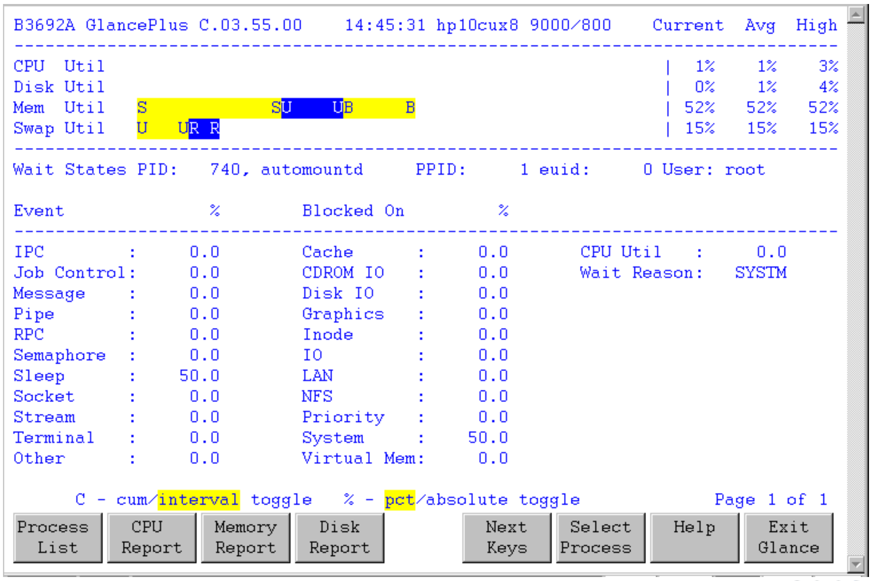
- Process Resource



```
B3692A GlancePlus C.03.55.00    05:45:12 hp10cux8 9000/800    Current  Avg  High
--------------------------------------------------------------------------------
CPU  Util                                                   |  0%    1%    2%
Disk Util   F                                               |  2%    3%    5%
Mem  Util   S            SU     UB    B                     | 52%   52%   52%
Swap Util   U  UR  R                                        | 15%   15%   15%
--------------------------------------------------------------------------------
Resources PID: 23559, glance         PPID: 23076 euid:     0 User: root
--------------------------------------------------------------------------------
CPU Usage (util):      0.2 Log Reads :       1 Wait Reason    :    STRMS
User/Nice/RT CPU:      0.0 Log Writes:       0 Total RSS/VSS  :  2.9mb/  6.3mb
System CPU      :      0.0 Phy Reads :       0 Traps / Vfaults:      1/      3
Interrupt CPU   :      0.0 Phy Writes:       0 Faults Mem/Disk:      0/      0
Cont Switch CPU :      0.0 FS Reads  :       0 Deactivations  :      0
Scheduler       :     HPUX FS Writes :       0 Forks & Vforks :      0
Priority        :      154 VM Reads  :       0 Signals Recd   :      1
Nice Value      :       10 VM Writes :       0 Mesg Sent/Recd :      0/      0
Dispatches      :        2 Sys Reads :       0 Other Log Rd/Wt:     33/     12
Forced CSwitch  :        0 Sys Writes:       0 Other Phy Rd/Wt:      0/      0
VoluntaryCSwitch:        1 Raw Reads :       0 Proc Start Time
Running CPU     :        1 Raw Writes:       0    Fri Jul 26 05:44:43 2002
CPU Switches    :        0 Bytes Xfer:     0kb                :

█     C - cum/interval toggle   % - pct/absolute toggle            Page 1 of 1
```

The items of interest here are the CPU utilization, Context switching, RSS/VSS, logical/physical read counts. Knowing what is ‘good’ or ‘bad’ depends largely on the application, but in general forced context switches mean you’re being a cpu hog and consuming your 10ms time slice, or are returning from a system call and another higher priority process is waiting.

Socket/Appl Tools - Glance



```
B3692A GlancePlus C.03.55.00    14:45:31 hp10cux8 9000/800    Current  Avg  High
--------------------------------------------------------------------------------
CPU  Util                                                      |  1%    1%    3%
Disk Util                                                      |  0%    1%    4%
Mem  Util   S            SU    UB      B                       | 52%   52%   52%
Swap Util   U   UR R                                           | 15%   15%   15%
--------------------------------------------------------------------------------
Wait States PID:   740, automountd     PPID:     1 euid:     0 User: root

Event                %            Blocked On           %
--------------------------------------------------------------------------------
IPC          :      0.0          Cache        :      0.0         CPU Util    :     0.0
Job Control:        0.0          CDROM IO     :      0.0         Wait Reason:    SYSTM
Message      :      0.0          Disk IO      :      0.0
Pipe         :      0.0          Graphics     :      0.0
RPC          :      0.0          Inode        :      0.0
Semaphore    :      0.0          IO           :      0.0
Sleep        :     50.0          LAN          :      0.0
Socket       :      0.0          NFS          :      0.0
Stream       :      0.0          Priority     :      0.0
Terminal     :      0.0          System       :     50.0
Other        :      0.0          Virtual Mem:        0.0

      C - cum/interval toggle    % - pct/absolute toggle              Page 1 of 1
```

Network related wait states are:

Pipe – Pipes (Non-streams based) are 8k in size and can block if data is being piped to a network connected process. The command '**dd if=/dev/rdsk/bigdisc bs=64k | remsh target node dd of=/dev/rmt/0m bs=64k**' will read from the raw disc in 64k read() calls, write it to the pipe, the stdin for the remsh reads in 1k increments, and sends it over the TCP connection to the target node's stdin for the dd process. A tape problem on the target node, would likely flow control off (TCP window of zero) the TCP connection. The remsh process would show blocked on socket, while the sending dd process would be blocked on pipe.

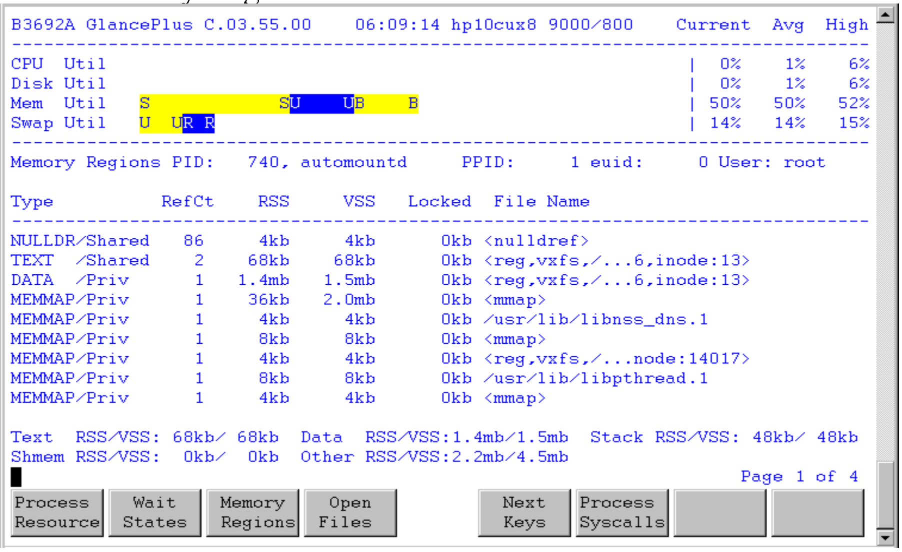
RPC – blocked waiting for a reply from an RPC program

Socket – blocked on a read or write of a socket FD.

Stream – blocked on getmsg() or putmsg() call on a Streams connection. The libxti library routines open /dev/udp and /dev/tcp streams based driver device files directly. Plus many other places in the kernel will be blocked on some portion of a stream connection.

Socket/Appl Tools - Glance

- Memory Regions

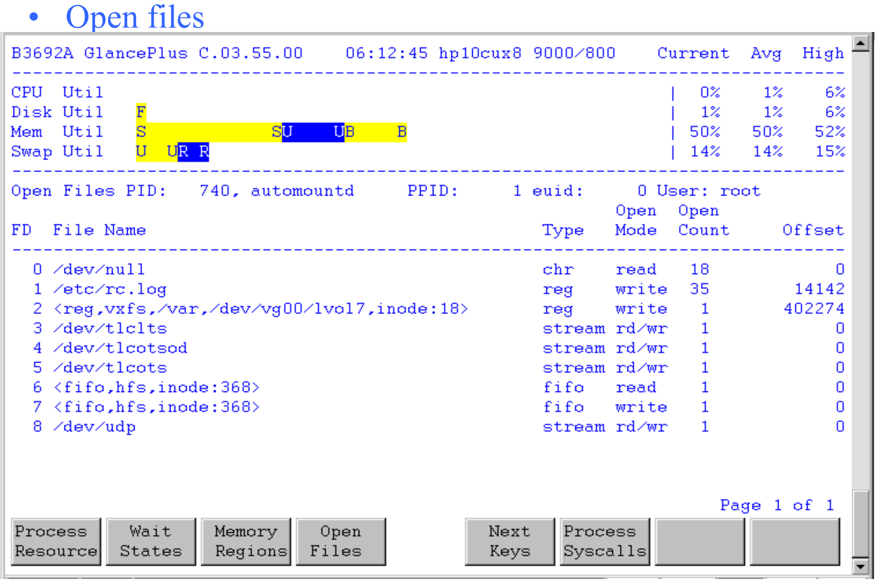


```
B3692A GlancePlus C.03.55.00    06:09:14 hp10cux8 9000/800    Current  Avg  High
--------------------------------------------------------------------------------
CPU  Util                                                        |  0%    1%    6%
Disk Util                                                        |  0%    1%    6%
Mem  Util   S            SU     UB     B                         | 50%   50%   52%
Swap Util   U  UR R                                              | 14%   14%   15%
--------------------------------------------------------------------------------
Memory Regions PID:   740, automountd     PPID:     1 euid:     0 User: root

Type          RefCt    RSS     VSS    Locked  File Name
--------------------------------------------------------------------------------
NULLDR/Shared   86      4kb     4kb      0kb <nulldref>
TEXT  /Shared    2     68kb    68kb      0kb <reg,vxfs,/...6,inode:13>
DATA  /Priv      1    1.4mb   1.5mb      0kb <reg,vxfs,/...6,inode:13>
MEMMAP/Priv      1     36kb   2.0mb      0kb <mmap>
MEMMAP/Priv      1      4kb     4kb      0kb /usr/lib/libnss_dns.1
MEMMAP/Priv      1      8kb     8kb      0kb <mmap>
MEMMAP/Priv      1      4kb     4kb      0kb <reg,vxfs,/...node:14017>
MEMMAP/Priv      1      8kb     8kb      0kb /usr/lib/libpthread.1
MEMMAP/Priv      1      4kb     4kb      0kb <mmap>

Text  RSS/VSS: 68kb/ 68kb  Data  RSS/VSS:1.4mb/1.5mb  Stack RSS/VSS: 48kb/ 48kb
Shmem RSS/VSS:  0kb/  0kb  Other RSS/VSS:2.2mb/4.5mb
                                                                   Page  1 of  4
```

A more detailed view of a processes memory resource usage…if DATA memory size is 1.5 Gigs and growing, you might have a memory leak :0

Socket/Appl Tools - Glance

- Open files



```
B3692A GlancePlus C.03.55.00    06:12:45 hp10cux8 9000/800    Current  Avg  High
--------------------------------------------------------------------------------
CPU  Util                                                    |   0%     1%     6%
Disk Util    F                                               |   1%     1%     6%
Mem  Util    S            SU    UB     B                     |  50%    50%    52%
Swap Util    U  UR R                                         |  14%    14%    15%
--------------------------------------------------------------------------------
Open Files PID:   740, automountd     PPID:     1 euid:     0 User: root
                                                      Open  Open
FD  File Name                                   Type   Mode  Count      Offset
--------------------------------------------------------------------------------
  0 /dev/null                                   chr    read   18             0
  1 /etc/rc.log                                 reg    write  35         14142
  2 <reg,vxfs,/var,/dev/vg00/lvol7,inode:18>    reg    write   1        402274
  3 /dev/tlclts                                 stream rd/wr   1             0
  4 /dev/tlcotsod                               stream rd/wr   1             0
  5 /dev/tlcots                                 stream rd/wr   1             0
  6 <fifo,hfs,inode:368>                        fifo   read    1             0
  7 <fifo,hfs,inode:368>                        fifo   write   1             0
  8 /dev/udp                                    stream rd/wr   1             0



                                                                   Page 1 of 1
```

If the lsof tool is not available this is the next best thing. In the example above there are no AF_INET or AF_UNIX sockets open, but you do see the /dev/udp and /dev/tlc* files open…these are network connections opened via calls to the libxti.2 shared library. The Glance openfiles screen does not label all the mmap'ed files, but the lsof tools does:

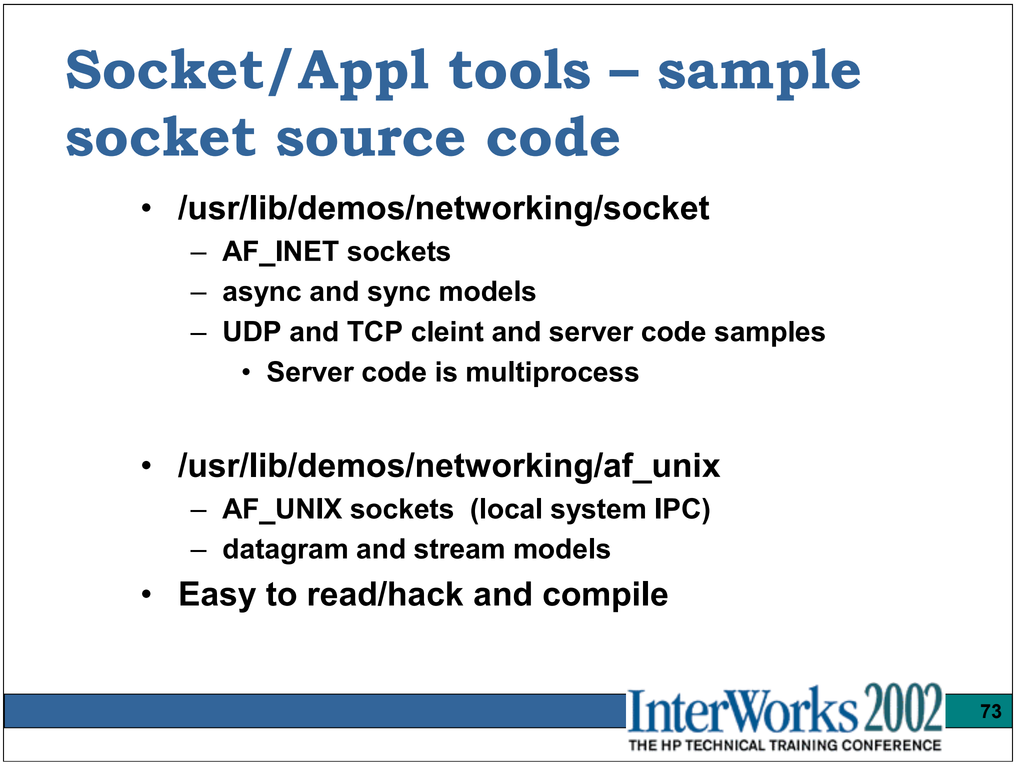
```
#lsof -p 740 | grep mem
automount 740 root   mem    REG      64,0x6    24576 13966
/usr/lib/libnss_dns.1
automount 740 root   mem    REG      64,0x6    16384 14017 /usr
(/dev/vg00/lvol6)

automount 740 root   mem    REG      64,0x6   147456   111
/usr/lib/libpthread.1
automount 740 root   mem    REG      64,0x6   135168   118 /usr/lib/libxti.2
automount 740 root   mem    REG      64,0x6   724992   120 /usr/lib/libnsl.1
automount 740 root   mem    REG      64,0x6    36864 13991 /usr
(/dev/vg00/lvol6)

automount 740 root   mem    REG      64,0x6  1843200 13855 /usr/lib/libc.2
automount 740 root   mem    REG      64,0x6    24576 13903 /usr/lib/libdld.2
automount 740 root   mem    REG      64,0x6   155648 13586 /usr/lib/dld.sl
```

Socket/Appl tools – logging

- Debugging is almost always best done at the highest layer possible.
  - analyzing a network trace is often akin to looking at footprints in the sand and trying to tell what color hat the guy was wearing…
- Multiple levels of detail
  - assuming more detail means a larger impact on appl performance
- If you’re going to bother to log a message, make it meaningful.
  - where in the code
  - timestamp in sufficient granularity
  - system errno and appl error together
- Dynamic enabling of logging
  - implemented as a signal handler

Many of the NFS RPC daemons use SIGUSER2 signal (kill –17) to toggle verbose logging with default logging locations….very handy.

Don’t print a message that says ‘recv on pipe’ if you are really reading data via shared mem, and semops



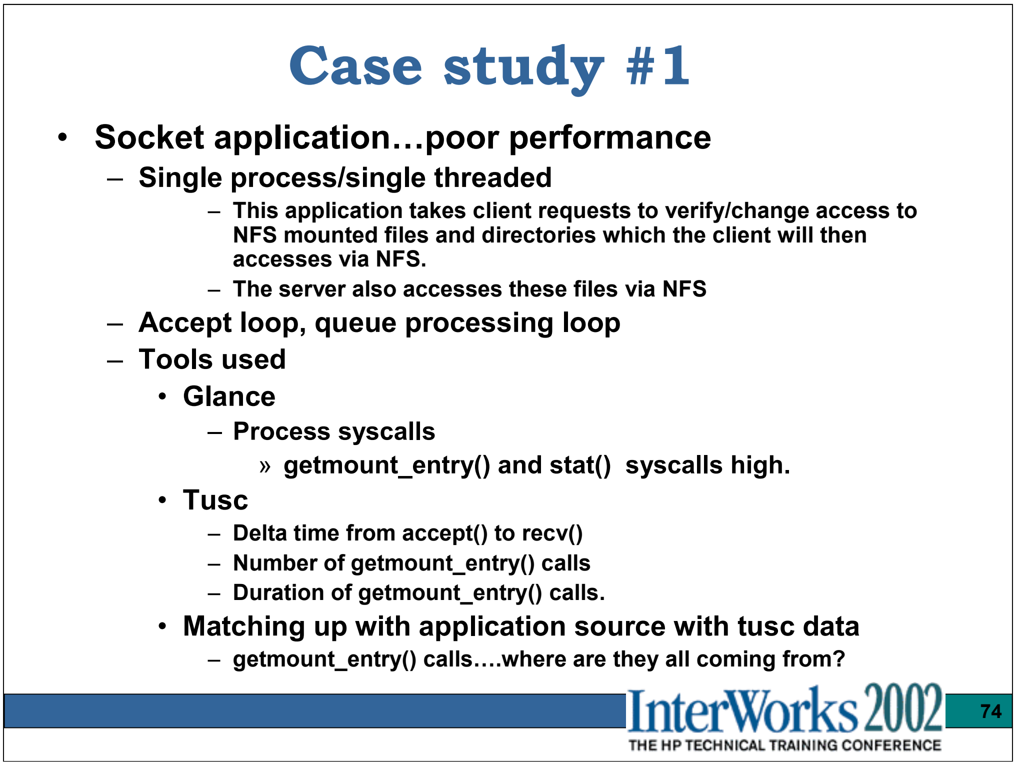
Socket/Appl tools – sample socket source code

- /usr/lib/demos/networking/socket
  - AF_INET sockets
  - async and sync models
  - UDP and TCP cleint and server code samples
    - Server code is multiprocess

- /usr/lib/demos/networking/af_unix
  - AF_UNIX sockets (local system IPC)
  - datagram and stream models
- Easy to read/hack and compile

The client and server AF_INET socket code provides sample usage for:

gethostbyname()
getservbyname()
getsockname()
socket()
connect()
send()
sendto()
recv()
recvfrom()
shutdown()
fork()
bind()
listen()
accept()
setsockopt()



Case study #1

- Socket application…poor performance
  - Single process/single threaded
    - This application takes client requests to verify/change access to NFS mounted files and directories which the client will then accesses via NFS.
    - The server also accesses these files via NFS
  - Accept loop, queue processing loop
  - Tools used
    - Glance
      - Process syscalls
        - » getmount_entry() and stat() syscalls high.
    - Tusc
      - Delta time from accept() to recv()
      - Number of getmount_entry() calls
      - Duration of getmount_entry() calls.
    - Matching up with application source with tusc data
      - getmount_entry() calls….where are they all coming from?

The single threaded process was falling behind in processing client requests. The clients had a timeout/retry algorithm that caused the server to do the work only to find the clients tcp connection had been closed when it timed out. The snowball was starting to roll downhill….

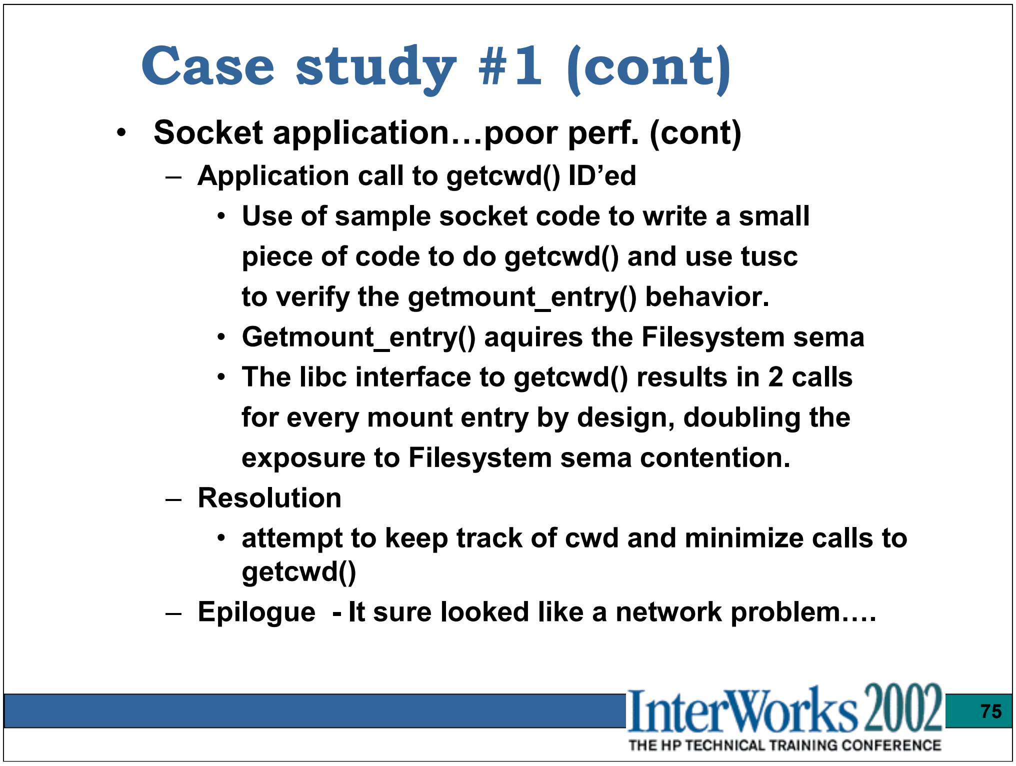
A modest number of new clients were added recently, but it was 'felt' that the server should be able to keep up. In reality it looked like a response-time cliff had been reached and the additional client load had pushed us over the edge.

The developers explained the basic design:

Select/accept new connections, put their request in a queue, when no more pending connections, go into processing loop to handle the queued requests (no laughing please). When the requested file/dir permission change has been accomplished, reply to the client, and go process the next request. When all queued requests are processed, go back to the select/accept loop.

They had tested the client connection handling in the past but were now seeing request handling times five times higher than expected.

So what was this server process spending its time doing?



Case study #1 (cont)

- Socket application…poor perf. (cont)
  - Application call to getcwd() ID’ed
    - Use of sample socket code to write a small piece of code to do getcwd() and use tusc to verify the getmount_entry() behavior.
    - Getmount_entry() aquires the Filesystem sema
    - The libc interface to getcwd() results in 2 calls for every mount entry by design, doubling the exposure to Filesystem sema contention.
  - Resolution
    - attempt to keep track of cwd and minimize calls to getcwd()
  - Epilogue - It sure looked like a network problem….

To find out where this process was spending its time a tusc trace was taken.

The time from accept() to close() for the socket connection was timed at 160ms.

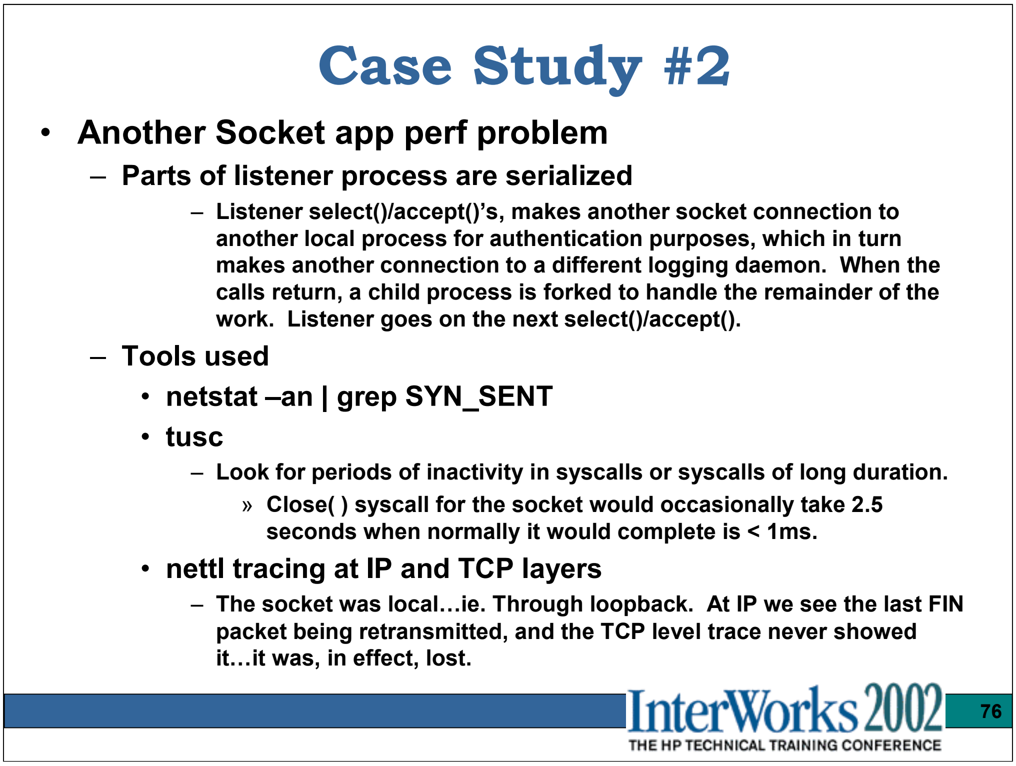
The bulk of the time was spent doing many repeated getmount_entry() calls and subsequent stat() calls against those mount points. Not every request resulted in the getmount_entry() call flood.

This matched the basic description the developer gave for the design…he said they open and read the /etc/mnttab file to see if the requested path is even mounted. Then go through every mount entry stat’ing it and seeing if it’s still alive…remember these are NFS mounted file systems. If, so the permissions and ownership are changed and the reply sent back to the client.

It was assumed the way they were reading the /etc/mnttab file was the cause of the getmount_entry() system calls….not so.

They said they were doing setmntent() and then repeated getmntent() libc calls. To see if indeed this was the cause, we wrote a small piece of code to do exactly that. No getmount_entry() calls were made. They then looked in their code and saw that for some types of requests, they made a getcwd() call just prior to the setmntent() and getmntent() loop.

Upon further investigation, the libc routine getcwd() is the one doing all the getmount_entry() calls. It also does so holding the filesystem semaphore.



Case Study #2

- **Another Socket app perf problem**
  - Parts of listener process are serialized
    - Listener select()/accept()'s, makes another socket connection to another local process for authentication purposes, which in turn makes another connection to a different logging daemon. When the calls return, a child process is forked to handle the remainder of the work. Listener goes on the next select()/accept().
  - Tools used
    - netstat –an | grep SYN_SENT
    - tusc
      - Look for periods of inactivity in syscalls or syscalls of long duration.
        - » Close( ) syscall for the socket would occasionally take 2.5 seconds when normally it would complete is < 1ms.
    - nettl tracing at IP and TCP layers
      - The socket was local…ie. Through loopback. At IP we see the last FIN packet being retransmitted, and the TCP level trace never showed it…it was, in effect, lost.

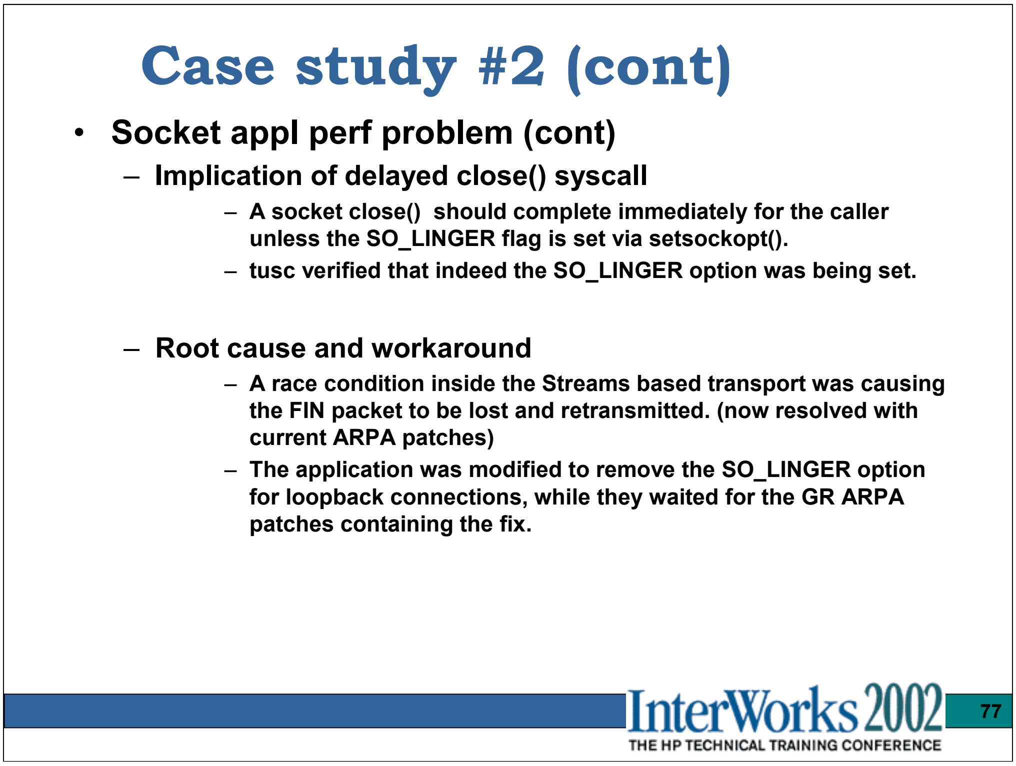
The netstat-an command showed a large number of local TCP connection going through loopback, but in the SYN_SENT state. Since this is loopback and we know the host is up and listening on the port, the only reason for this was a full listen queue. This was indeed the case as verified by looking and the kernel socket structure for the socket.

The response time from the main listener process was slow and erratic, as measure from accept() to the first reply of data to the client via the send() syscall.

Again the question is 'what is this process spending it's time doing?'.

We ran tusc on the main listener as well as the two other pertinent processes and spotted an intermittent close() syscall on a socket that was delayed for 2.5 seconds. During this period the process was blocked. Why? The socket accepted had a setsockopt() call made to set so_linger which would delay the close() from returning until the TCP connection had completely shutdown.

A nettl trace of the ns_ls_ip and ns_ls_ip layer was taken. A loopback connection should shutdown gracefully with no delays.



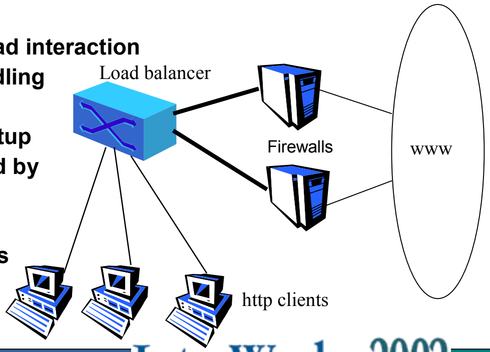
Case study #2 (cont)

- Socket appl perf problem (cont)
  - Implication of delayed close() syscall
    - A socket close() should complete immediately for the caller unless the SO_LINGER flag is set via setsockopt().
    - tusc verified that indeed the SO_LINGER option was being set.
  - Root cause and workaround
    - A race condition inside the Streams based transport was causing the FIN packet to be lost and retransmitted. (now resolved with current ARPA patches)
    - The application was modified to remove the SO_LINGER option for loopback connections, while they waited for the GR ARPA patches containing the fix.

The second FIN in the three-way shutdown (FIN→ ← FIN/ACK ACK→) was being dropped between IP and TCP, causing a retransmission timer to pop, and a delay of the close. Loosing packets on a loopback connection is not suppose to happen.

Case study #3

- Poor http performance through firewall
  - Load balancing equipment for http traffic load.
    - The HP Firewall seems to have a consistently higher connection count at all times compared to other HW vendors running the equivalent Firewall product, thus the Load Balancer sends more traffic to the other Firewalls resulting in ‘low connection handling’ rates on the HP.
    - http daemons are mutliprocess and multithreaded
  - Tools used
    - Tusc
      - » Syscall trace of thread interaction and connection handling
    - Nettl
      - » Trace connection setup and teardown filtered by kernel threadID
    - httpd daemon log files.
      - » What the httpd thinks it’s doing.

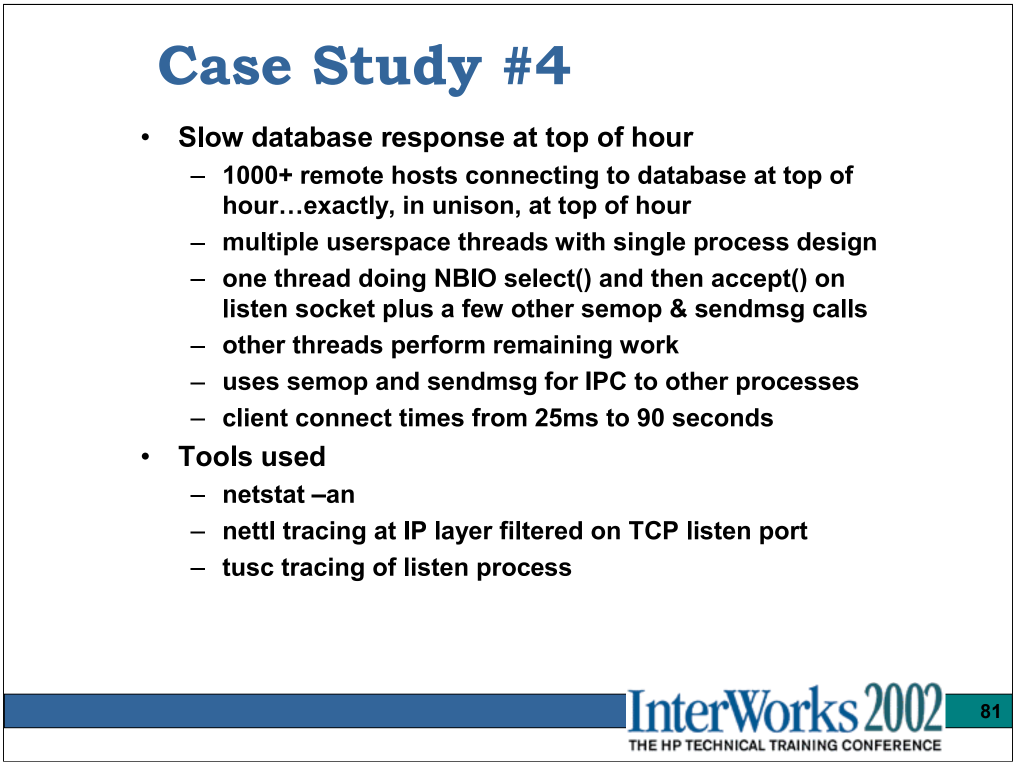


Case Study #3 (cont)

- Poor http performance thru Firewall
  - nettl was used to watch a typical connection from start to finish. Inconsistent connection times noted…some fast, some slow.
    - Connection termination was non-graceful… TCP reset packets.
  - httpd daemon logs gave no indication of errors other than failing to perform some name lookups…DNS
  - tusc was used to trace the threads of on of the ten httpd daemons.
    - all threads seemed to be waiting (ksleep() syscall) on a resource/lock held by another thread within the same process.
    - Backtracking the thread that released the resource (it was the one making kwapeup() calls) it appeared to be doing DNS queries.
      - open of /etc/nsswitch.conf
      - sendto() calls etc. to port 53

Case Study #3 (cont)

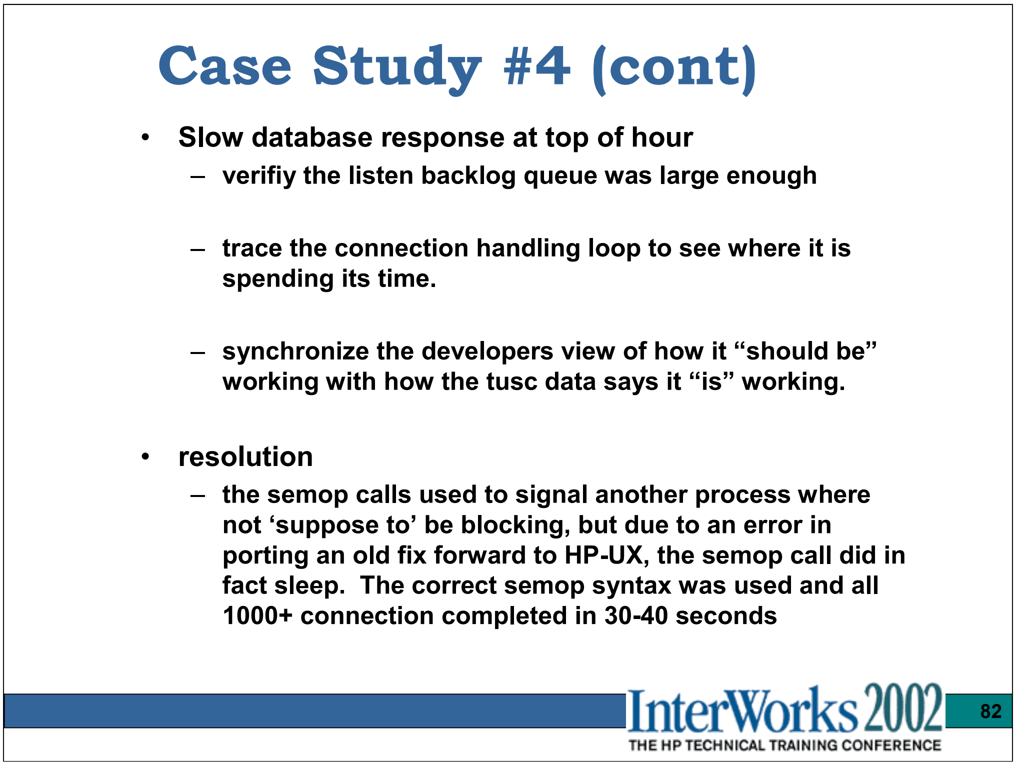
- Poor http performance thru Firewall – resolution
  - The tusc data pointed to DNS calls made via gethostnyname() as the source of the thread mutex lock/resource contention.
  - Code inspection of the libc.1 and libnsl.1 library routines showed that the resolver routines they were using had a mutex lock to single thread all DNS queries for threads within a single process….a legacy protection from days when DNS resolver routines were not thread safe. A patch for libnsl.1 removed this unnecessary mutex lock.
  - sample threaded socket code was written to test/verify the mutex lock behavior.
  - With this intermittent hold up of connection processing removed, the connection 'handling rate' outperformed the competition's HW running the same firewall product by 20%



Case Study #4

- Slow database response at top of hour
  - 1000+ remote hosts connecting to database at top of hour…exactly, in unison, at top of hour
  - multiple userspace threads with single process design
  - one thread doing NBIO select() and then accept() on listen socket plus a few other semop & sendmsg calls
  - other threads perform remaining work
  - uses semop and sendmsg for IPC to other processes
  - client connect times from 25ms to 90 seconds
- Tools used
  - netstat –an
  - nettl tracing at IP layer filtered on TCP listen port
  - tusc tracing of listen process

The general investigation approach was to us tusc on the listener process to see how quickly he was handling the incoming connections. Then map out where he was spending his time and, using average execution times for each phase/step, go hunting for any events/trace entries that differed greatly from the average.



Case Study #4 (cont)

- **Slow database response at top of hour**
  - verifiy the listen backlog queue was large enough
  - trace the connection handling loop to see where it is spending its time.
  - synchronize the developers view of how it “should be” working with how the tusc data says it “is” working.
- **resolution**
  - the semop calls used to signal another process where not ‘suppose to’ be blocking, but due to an error in porting an old fix forward to HP-UX, the semop call did in fact sleep. The correct semop syntax was used and all 1000+ connection completed in 30-40 seconds

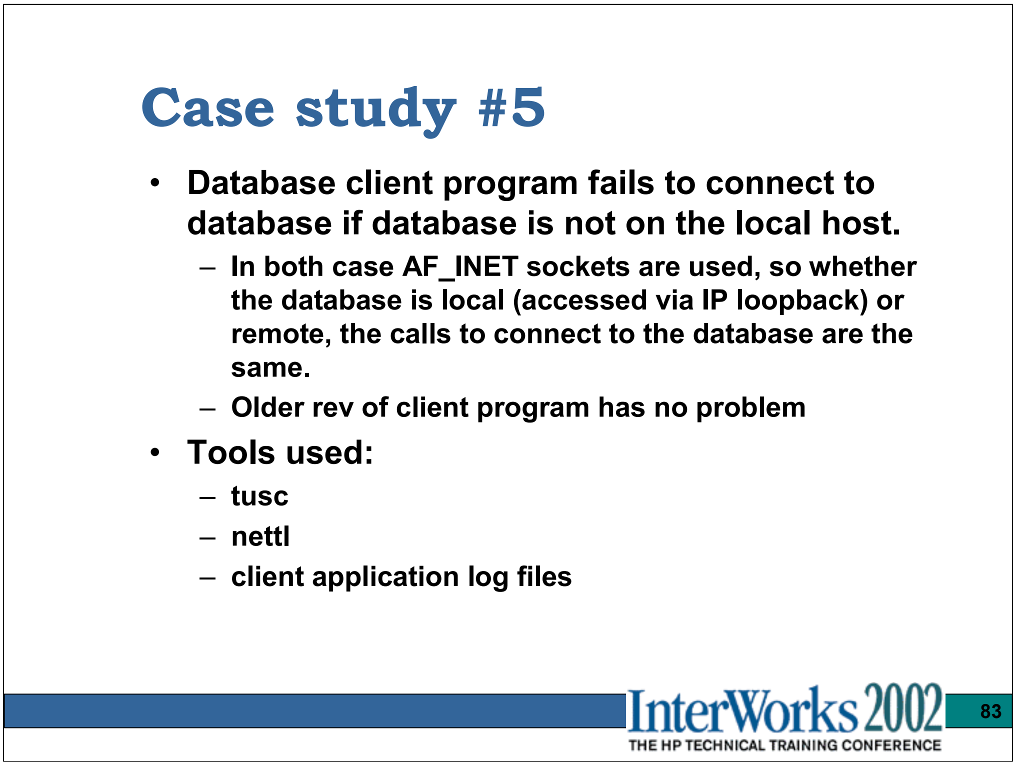
The listen() syscall was requesting a fairly large backlog queue, but the transport silently imposes it’s own limit based on the ndd tunable tcp_conn_request_max…….which was at the default of 20.

We increased the tcp_conn_request_max to 1024 and then the client TCP connections would at least all go to the ESTABLISHED state…netstat –an would show them all connected.

The tusc data for the listener process indicated that a typical client connection could be performed in 25ms, or about 40 per second on average.

Further investigation of the tusc trace for the slow connect cycles showed 15-3000ms of that time was spent in one semop call used to signal another process thread to handle the new connection. The developers said this semop call should NOT be blocking, but it obviously was.

The other platforms this database ran on did not experience this performance delay because a fix involved in making these semop calls not block was not correctly ported to HP-UX. The tusc data forced them to double check the source.

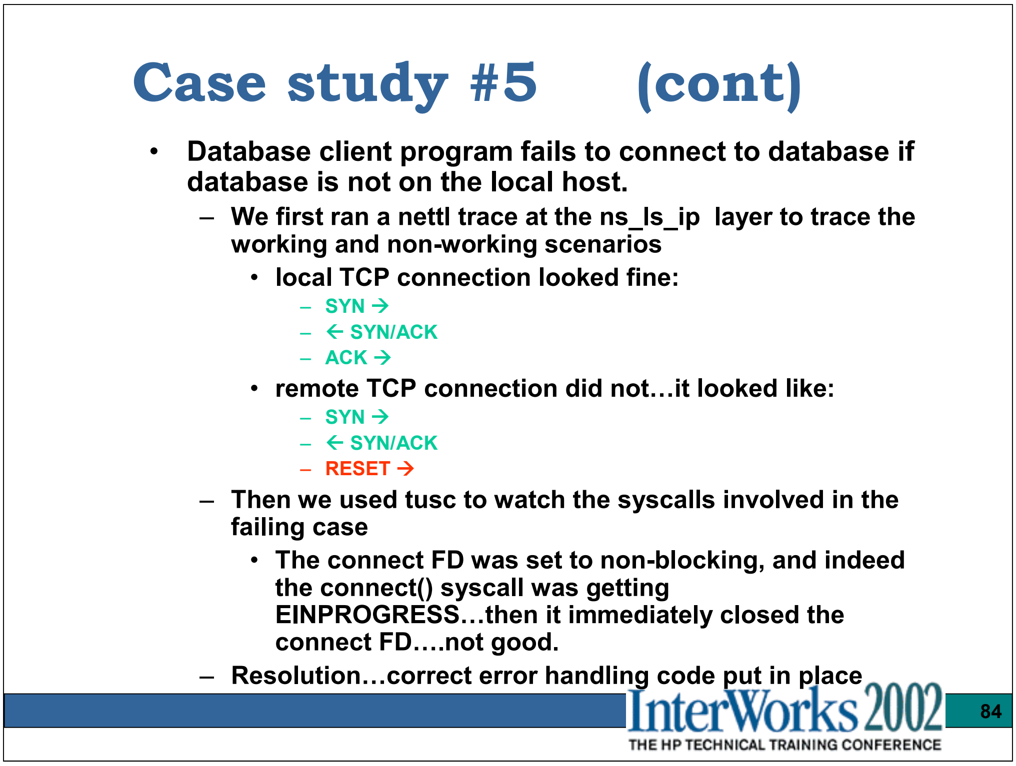


Case study #5

- Database client program fails to connect to database if database is not on the local host.
  - In both case AF_INET sockets are used, so whether the database is local (accessed via IP loopback) or remote, the calls to connect to the database are the same.
  - Older rev of client program has no problem
- Tools used:
  - tusc
  - nettl
  - client application log files

This was a 10.20 to 11.0 porting effort that failed outright for 11.X if the database accessed was on a remote node.

The application level log file simply showed a time out event when the remote database was opened.



Case study #5 (cont)

- Database client program fails to connect to database if database is not on the local host.
  - We first ran a nettl trace at the ns_ls_ip layer to trace the working and non-working scenarios
    - local TCP connection looked fine:
      - SYN →
      - ← SYN/ACK
      - ACK →
    - remote TCP connection did not…it looked like:
      - SYN →
      - ← SYN/ACK
      - RESET →
  - Then we used tusc to watch the syscalls involved in the failing case
    - The connect FD was set to non-blocking, and indeed the connect() syscall was getting EINPROGRESS…then it immediately closed the connect FD….not good.
  - Resolution…correct error handling code put in place

The tusc trace showing the failing connect:

```
1027621190.090693 [27747]{1293} #6 <0.000212> socket(AF_INET, SOCK_STREAM, 0) ........ = 9
1027621190.091067 [27747]{1293} #6 <0.000087> bind(9, 0x7f5cf584, 16) ................ = 0
  Family: AF_INET
  Port: 0
  Addr: 0.0.0.0
1027621190.091486 [27747]{1293} #6 <0.000057> fcntl(9, F_GETFL, 0) ................... = 2
1027621190.091835 [27747]{1293} #6 <0.000037> fcntl(9, F_SETFL, 6) ................... = 0
1027621190.092258 [27747]{1293} #6 <0.000167> connect(9, 0x7f5cf56c, 16) .............
ERR#245 EINPROGRESS
  Family: AF_INET
  Port: 23301
  Addr: 1XX.1XX.56.213
1027621190.092848 [27747]{1293} #6 <0.000359> close(9) ............................... = 0
```

The resolution was to correct the logic that was incapable of understanding and dealing with the EINPROGESS return on the non-blocking connect( ) call.