RN110007B

HP-UX 11.0 Release Notes, Networking, Part 2 of 2

Document Information Table

HP-UX 11.0 Release Notes, Networking, Part 2 of 2 Parts
CHAPTER 7: Networking

TCP/IP
@@
For 10.30:

The TCP/IP stack is streams-based in 10.30. Some of the pre-10.30 BSD kernel data structures may no longer exist.

The following lists the new features for 10.30. These features are described in detail in this section.

* /etc/rc.config.d/netconf changes.
* Logical interfaces allow multiple IP addresses per card.
* IP/IEEE802.3 requires SNAP encapsulation and a separate logical interface.
* ndd utility for examining and modifying system-wide transport parameters.
* New PPP software replaces PPL to provide both PPP and SLIP connections.

Obsolete Features
****

* lanconfig (functionality included in ifconfig/lanadmin).
* **ifalias** (functionality included in ifconfig).
* nettune (functionality included in ndd).
* ppl (replaced by pppd).

Changes to /etc/rc.config.d/netconf
****

The 10.30 installation script will modify any existing /etc/rc.config.d/netconf file to match the new file syntax. In most configurations, no further changes are necessary. However, systems that use IP over IEEE802.3 must change their IP configuration. Refer to the sections "Logical Interfaces: IP/Ethernet vs. IP/IEEE802.3" and "Configuring SNAP/IEEE802.3 Interfaces" for more information.

LANCONFIG_ARGS

Statements for LANCONFIG_ARGS are no longer supported.
Any LANCONFIG_ARGS statements are ignored.

INTERFACE_NAME

The interface name syntax for INTERFACE_NAME statements has been expanded to allow "logical interface" names. Logical interfaces and interface names are explained in the sections below.

New Statements: INTERFACE_STATE, DHCP_ENABLE

There are two new statements: INTERFACE_STATE and DHCP_ENABLE.

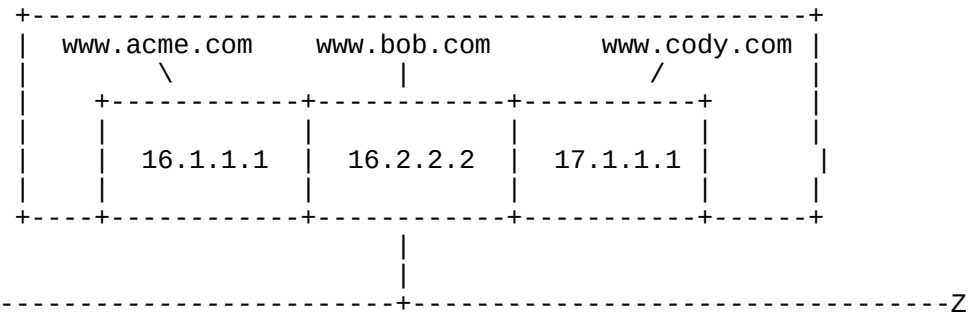
INTERFACE_STATE determines the logical interface state ("up", "down", or blank) at system boot time. If nothing is specified, the state defaults to "up".

DHCP_ENABLE determines if the system is a Dynamic Host Configuration Protocol (DHCP) client and if the IP parameters (IP address, subnet mask, and so on) will be set using DHCP. The default is 0 (the system is not a DHCP client). The DHCP_ENABLE statement was added at 10.20.

Logical Interfaces: IP Multiplexing

Logical interfaces let you multiplex IP addresses over a physical interface (IP multiplexing). With IP multiplexing, you configure multiple IP addresses for a single physical interface.

IP multiplexing allows a single system to be seen as multiple systems with multiple IP addresses and host names, even if the system has only one physical interface card. This functionality allows multiple instances of applications to appear as if they reside on separate machines. For example, an Internet Services Provider (ISP) may provide websites for the Acme, Bob, and Cody companies. Using IP multiplexing, a single system with one physical interface can appear to have three IP addresses, each home to a different website. From the outside world, it would appear as if the Acme, Bob, and Cody companies have websites on different machines.



Note that the IP addresses assigned to a card may be on the same subnet or on different subnets. In this example, two of the IP addresses for the card are part of the same network (16.1.1.1 and 16.2.2.2) and the third address is from a different network (17.1.1.1).

There are no performance advantages or disadvantages. All transport resources (buffers) are pooled and used by all logical interfaces with no partitioning for the individual interfaces.

To use IP multiplexing with routers, the router devices must be able to map multiple IP addresses to the same MAC address. Some vendors call this functionality "multinetting" or "secondary subnets" and may limit the maximum number of IP addresses mapped to the same MAC address. For example, the HP 650 router supports a maximum of 16 IP addresses mapped to the same MAC address.

Logical Interfaces: IP/Ethernet vs. IP/IEEE802.3

Logical interfaces are also used when an interface card is used for both IP/Ethernet and IP/IEEE802.3 packets. At HP-UX 10.30, sending IP packets using ethernet and sending IP packets using IEEE802.3 requires two separate logical interfaces of the transport. To send IP packets using ethernet and IEEE 802.3, you must configure two logical interfaces with two different IP addresses. In addition, the IP addresses must be in two different subnets. (In prior releases, IP packets could be sent over ethernet or IEEE 802.3 simultaneously using the same IP address.) Refer to the section "Interface Names" and "Configuring SNAP/IEEE802.3 Interfaces" for more information on configuring logical interfaces for IP/IEEE802.3.

In addition, at 10.30, all IP packets sent over IEEE802.3 must use

Sub-Network Access Protocol (SNAP) encapsulation. In SNAP, the IEEE802.3 DSAP and SSAP fields are both hexadecimal AA. The 802.3 control field is 3 (unnumbered information or data). Following this is the SNAP header: an organization code and type field. The organization code is 24-bits and assigned by IEEE, and is typically 0x000000. The 16-bit type field is used to carry an ethernet type value. For example, the type value for IP is 0x0800, as shown below.

dest MAC	src MAC	DSAP	SSAP	ctl	organize	type	data ...
		AA	AA	03	00 00 00	08 00	IP hdr

IEEE802.3 and DTCs, JetDirect, MPE-XL

DTCs and JetDirect cards communicate with IEEE802.3 link-level (DLPI) applications on HP-UX hosts. These applications run directly over the IEEE802.3 protocol (IP is not used) and do not require a SNAP/IEEE802.3 logical interface. SNAP/IEEE802.3 logical interfaces are only required for IP packets sent over IEEE802.3.

All supported MPE-XL systems (version 4.0 and later) support ethernet.

Interface Names

Interface names used for ifconfig commands and /etc/rc.config.d/netconf statements can have a logical instance number appended to the card name (for example, lan0:1). The new syntax is:

nameX[:logical_instance]

Where:
name is the class of interface, such as lan (Ethernet LAN, token ring, FDDI, or Fibre Channel links), snap (IEEE802.3 with SNAP encapsulation), atm (ATM), du (Dial-up), ixe (X.25), and mfe (Frame Relay).

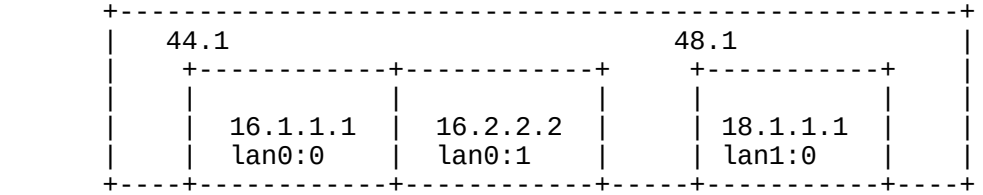
X is the Physical Point of Attachment (PPA). This is a numerical index for the physical card in its class. For LAN devices, the lanscan command will display the name and PPA number concatenated (such as lan0) in the Net-Interface NamePPA column.

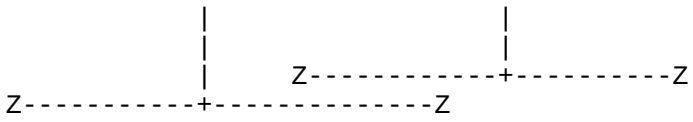
logical_instance is an index corresponding to the logical interface for the specified card. The default is 0. The interface name lan0 is equivalent to lan0:0.

The first logical instance (logical instance 0) for a card type and card instance (lan0:0, lan1:0, snap0:0) is known as the initial interface. The initial interface for a card/encapsulation type must be configured before subsequent logical interfaces. For example, you must configure lan2:0 (or lan2) before configuring lan2:1. Once you have configured the initial interface, you can configure the subsequent logical interfaces in any order.

Configuring Multiplexed IP Addresses

In this example, the system has two physical interfaces and three logical interfaces.





lanscan Output:

```
Hardware Station     Crd Hdw   Net-Interface  NM  MAC       HP-DLPI DLPI
Path     Address     In# State NamePPA        ID  Type      Support Mjr#
44.1   0x080009267C14 0   UP    lan0 snap0     1   ETHER     Yes     119
48.1   0x080009260C85 1   UP    lan1 snap1     2   ETHER     Yes     119
```

Config Statements:

```
 ifconfig lan0:0 inet 16.1.1.1
 ifconfig lan0:1 inet 16.2.2.2
 ifconfig lan1:0 inet 18.1.1.1
```

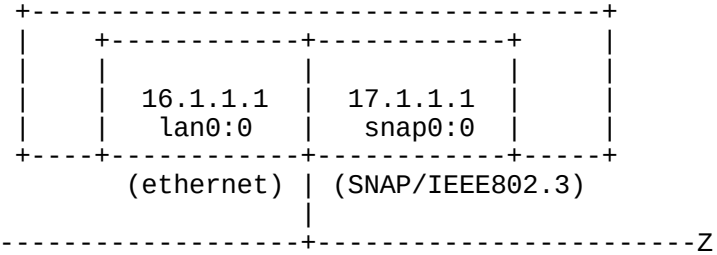
/etc/rc.config.d/netconf Statements:

```
 INTERFACE_NAME[0]=lan0
 IP_ADDRESS[0]=16.1.1.1
 SUBNET_MASK[0]=255.0.0.0
 BROADCAST_ADDRESS[0]=""
 INTERFACE_STATE[0]=up
 DHCP_ENABLE[0]=0

 INTERFACE_NAME[1]=lan0:1
 IP_ADDRESS[1]=16.2.2.2
 SUBNET_MASK[1]=255.0.0.0
 BROADCAST_ADDRESS[1]=""
 INTERFACE_STATE[1]=up
 DHCP_ENABLE[1]=0

 INTERFACE_NAME[2]=lan1
 IP_ADDRESS[2]=18.1.1.1
 SUBNET_MASK[2]=255.0.0.0
 BROADCAST_ADDRESS[2]=""
 INTERFACE_STATE[2]=up
 DHCP_ENABLE[2]=0
```

Configuring SNAP/IEEE802.3 Interfaces

****

In this example, the system has one physical interfaces and two logical interfaces. One logical interface is used for SNAP/IEEE802.3.



ifconfig Statements:

```
 ifconfig lan0:0 inet 16.1.1.1
 ifconfig snap0:0 inet 17.1.1.1
```

/etc/rc.config.d/netconf Statements:

```
 INTERFACE_NAME[0]=lan0:0
 IP_ADDRESS[0]=16.1.1.1
 SUBNET_MASK[0]=255.0.0.0
```

```
 BROADCAST_ADDRESS[0]=""
 INTERFACE_STATE[0]=up
 DHCP_ENABLE[0]=0

 INTERFACE_NAME[1]=snap0:0
 IP_ADDRESS[1]=17.1.1.1
 SUBNET_MASK[1]=255.0.0.0
 BROADCAST_ADDRESS[1]=""
 INTERFACE_STATE[1]=up
 DHCP_ENABLE[1]=0
```

Note that the IP address for snap0:0 cannot be in the same subnet as the IP address for lan0:0. Also note that the logical instance number for the snap logical interface (snap0:0) is 0, not 1. Logical instance numbers are sequenced from 0 for each interface name.

lanconfig
****

* The lanconfig command and its ieee flag have been obsoleted.

* The ether and snap802.3 options of the lanconfig command have been integrated into the ifconfig command.

* The LANCONFIG_ARGS variable in /etc/rc.config.d/netconf is ignored.

* The Token Ring rif option (source routing) is now part of the lanadmin command. To disable source routing:

```
    /usr/sbin/lanadmin -B off <PPA_number>
```

  To enable:

```
    /usr/sbin/lanadmin -B on <PPA_number>
```

  To display the current setting:

```
    /usr/sbin/lanadmin -b <PPA_number>
```

  By default, source routing is enabled on HP Token Ring devices. Use the lanscan command to determine the PPA numbers for LAN devices.

ifalias
****

The **ifalias** command that used to be in /usr/contrib/bin/ is obsolete. Its functionality has been integrated into the ifconfig command (IP multiplexing/logical interfaces).

netstat Command
****

The netstat "-r" option "Use" column no longer reflects the number of packets transmitted using a particular route. Generally, packet counts are not recorded for each route. However, if a local application sends packets to a local IP address, that number of packets will be reflected in the local host route.

Users can use the lanadmin command to determine the number of packets transmitted from a specific network interface. However, this will not distinguish one route from another.

NOTE: This behavior may be changed in a future patch or release without notice.

```
netstat
************************************************************************

The following netstat options are no longer supported:

-p arp
-p probe
-A
-rs
-t
-m

With the support of multiple logical interfaces, the netstat -i command
displays all IP interfaces (logical interfaces) configured through
ifconfig or the /etc/rc.config.d/netconf file.  There are four changes
in the netstat -i output:

* The "Name" field of netstat -i may include interface names
  corresponding to logical interfaces with the format lan0,
  lan0:1, lan0:2, and so on.

* The Ierrs, Oerrs, Collision counts have been removed in 10.30.
  The Errors/Collision information for each physical interface is
  available through the lanadmin command instead.

* The Ipkts and Opkts counts for each interface are for TCP/UDP/IP
  traffic only (such as IP packets).

* Physical interfaces that have not been configured using
  ifconfig will not be displayed by netstat.  Use lanscan to
  display all physical interfaces on the system.

An example is shown below:

# netstat -i

Name       Mtu Network         Address                 Ipkts      Opkts
lo0       4136 127.0.0.0       localhost                 206        206
lan0      1500 15.1.4.0        hpaa0                 2986637     219346
lan1      1500 192.1.0.0       hpaa1                       0          0
lan1:1    1500 202.1.0.0       hpaa2                       0          0

Changes to netstat -r
------------------------------------------------------------------------
  1. The PmtuTime field in the netstat -r output is no longer supported.

  2. The output will show a host route for each IP interface.
     These routes are automatically created when the interfaces
     are configured via the ifconfig command or through the
     /etc/rc.config.d/netconf file.

nettune and ndd
************************************************************************

The supported utility /usr/bin/ndd lets administrators modify
system-wide transport parameters in the working kernel.  The ndd utility
replaces the nettune command.  Some of internal kernel variables from
releases prior to 10.30 may no longer exist (such as tcp_sendspace,
tcp_dont winscale, and so on).  Therefore, adb scripts developed for
pre-10.30 releases may not work on 10.30 systems.

Administrators can use the file /etc/rc.config.d/nddconf to have ndd
modify network kernel parameters every time the system is booted.

SLIP and PPP
************************************************************************
```

The ppl command for the SLIP protocol and its associated configuration files is obsolete. Use the new pppd command to configure and administer SLIP and PPP links. The configuration requirements for pppd are similar to those used for ppl, but the configuration files have different names and formats. Migrating existing SLIP links to PPP links is explained in the manual Installing and Administering PPP (part number B2355-90137).

netman and ni Drivers

The netman and ni drivers are not supported in HP-UX 10.30. The device files /dev/netman and /dev/ni are no longer available.

Copy-on-Write (COW)

Copy-on-write, one of the copy avoidance features, will no longer be supported in HP-UX 10.30. Checksum-offload and page remapping are still supported.

Loopback Interface (lo0)

Any attempt to change the address of primary loopback interface (lo0:0) will fail. lo0:0 is automatically configured to 127.0.0.1. Any address configured for the loopback logical interface (such as lo0:1) will be treated as a loopback address.

IP Packets to Local Host Address

All IP packets destined for the local host address will be looped back through the IP layer. These packets will not be sent to the drivers.

TCP Keepalive Packets

The algorithm of sending TCP keepalive packets has been changed.

The following kernel parameters affect TCP behavior for keepalive o packets:

tcp_keepalive_interval tcp_ip_abort_interval tcp_ip_abort_cinterval tcp_keepalive_detached_interval

The parameter tcp_keepalive_interval determines the amount of time that TCP waits for an idle connection with no unacknowledged data before sending keepalive packets. The default is 2 hours.

If the remote does not acknowledge the keepalive packet, TCP will use one of the following retransmission timers and terminate the connection when it elapses as follows:

State	Timer	Default
Established connection	tcp_ip_abort_interval	600000 ms (10 minutes)
Connection establishment	tcp_ip_abort_cinterval	240000 ms (4 minutes)
Connection terminating	tcp_keepalive_detached_interval	240000 ms (4 minutes)

The default values may change in future releases. For more information, use the command:

```
   ndd -h <timer_name>
```

Refer to the tcp(7p) and getsockopt(2) manpages for more information.

Networking Memory for Fragment Reassembly
****

The configurable kernel parameter netmemmax is no longer supported. There is an ndd tunable, ip_reass_mem_limit, that can be used to limit the fragment reassembly memory at run time.

For more information, use the command:

```
 ndd -h ip_reass_mem_limit
```

Netisr Priority
****

The streams-based TCP/IP does not support the network interface daemon, netisr. The kernel parameter netisr_priority does not exist in 10.30.

TCP Hash Table Size
****

The size of the hash table for TCP connections is determined by the kernel parameter tcphashsz. This kernel parameter should only be modified under the direction of Hewlett-Packard.

Interface information: ioctl SIOCGIFCONF/SIOCGIFNUM
****

The SIOCGIFCONF ioctl call will return all IP interfaces configured in the system. The ifr_name field of the SIOCGIFCONF ioctl may contain logical interface names. SIOCGIFCONF does not return information for physical interfaces that have not been configured using ifconfig or /etc/rc.config.d/netconf.

The new SIOCGIFNUM ioctl call will return the total number of IP interfaces (initial and any subsequent interfaces) configured in the system. This is useful for determining the size of the output buffer needed by SIOCGIFCONF.

TCP Timers: setsockopt TCP_ABORT_THRESHOLD and TCP_CONN_ABORT_THRESHOLD
****

The TCP timers tcp_ip_abort_interval and tcp_ip_abort_cinterval can be set for individual sockets using the setsockopt options TCP_ABORT_THRESHOLD and TCP_CONN_ABORT_THRESHOLD.

Refer to the section "TCP Keepalive Timers" in this document and the tcp(7p) and getsockopt(2) manpages for more information.

Transport-Independent RPC (TI-RPC)
@@@@@@@@@@

For 10.30:

The transport-independent RPC (TI-RPC) system provides a single, consistent programming interface across machines and network transport protocols. TI-RPC makes RPC applications transport-independent by delaying the binding of the application to a specific transport until the program is invoked.

TI-RPC replaces all the old RPC interface that reside in libc.2/a. All TI-RPC APIs now reside in libnsl.

Previously, with transport-specific ONC RPC, this binding was done at

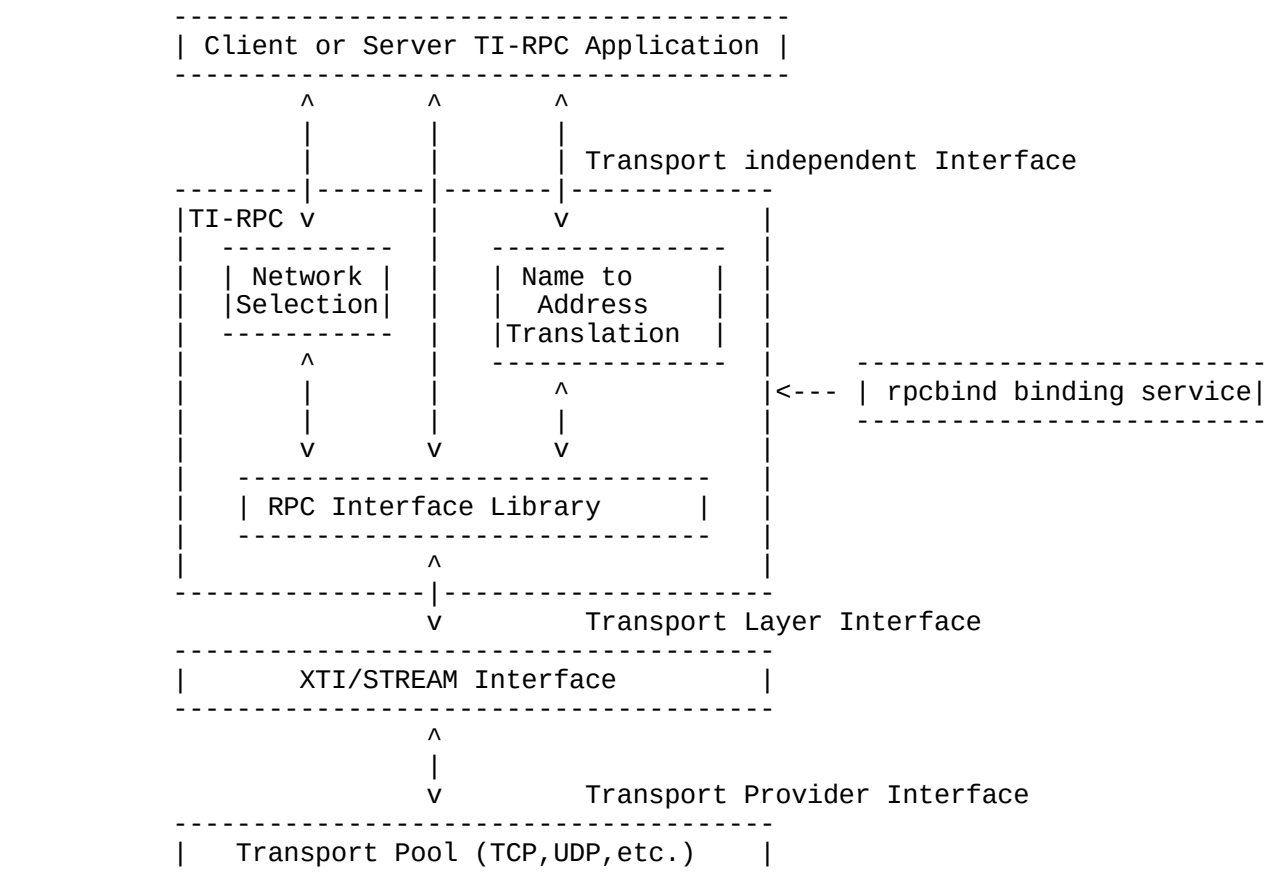
compile time, so the application could not take advantage of new transports unless the program was rebuilt. With TI-RPC, new transports can be used by the application if you update a network configuration file (/dev/netconfig) and restart the program. /etc/netconfig is part of the Network Selection library which facilitates run-time transport selection.

New RPC programming interfaces are added to the RPC library to support transport-independence. These new interfaces are Network Selection(NS) and Name to Address Translation(N2A). NS facilitates run-time transport selection and N2A provides universal addresses that are independent of the chosen transport. The TI-RPC library includes the older interfaces for backwards compatibility.

TI-RPC provides the following:

* Supports Multiple Networking Transports:
  * Enhances the heterogeneity of distributed applications
* Handles Differences in Networking Protocols:
  * Application development does not require low-level networking knowledge. Developers only specify procedures to be distributed.
* Allows Runtime Transport-Independence:
  * Users can run the same binary version of an application on any supported protocols.
  * Developers need not produce a different version of their application for each network protocol.
  * Eases application migration to new networking technologies.

The following figure illustrates the new TI-RPC software structure:



```
   -------------------------------------
```

At the top is your application. The interface to the RPC library has changed very little, except there are many new procedures. Typically, you will only need to call the the topmost layer of the RPC Interface Library. Very rarely will you need to use the Network Selection library and almost never will you need to call the Name to Address Translation modules.

Network Selection facilitates run-time transport selection. It provides the means to choose the transport on which an application should run. It is based upon two inputs, the netconfig database (/etc/netconfig) and the optional environment variable NETPATH. The netconfig lists the transports available on the host, including information about the transport such as its type, device name, and name-to-address translation module. Here are a few sample entries in the file:

```
/etc/netconfig:
#netid  type        flags   family protocol  device address
loadable module
udp     tpi_clts      v       inet    udp     /dev/udp
 -
tcp     tpi_cots_ord  v       inet    tcp     /dev/tcp
 -
```

Name-to-address translation provides universal addresses that are independent of the chosen transport.

TI-RPC was produced without affecting the existing ONC RPC protocol, which was always transport-independent. Because of that, ONC RPC applications can be run as is and even recompiled under TI-RPC. Applications that do not make any explicit socket system calls are source compatible with the new implementation. Applications that make socket system calls require minor changes.

Impact
****

Any applications that explicitly make socket system calls must now use its equivalent XTI calls.

X/Open Sockets: socklen_t
@@

The data type of some parameters and struct members is changed from size_t to socket_t.

Ostensibly, this change affects only applications compiled with the option -D_XOPEN_EXTENDED_SOURCE.

However, because socklen_t and size_t have the same underlying type, the change has no immediate consequences.

The data type of the following parameters and struct members has been changed from size_t to socket_t defined in <sys/socket.h>:

```
   int accept(..., socklen_t *addrlen);
   int bind(..., socklen_t addrlen);
   int connect(..., socklen_t addrlen);
   int getpeername(..., socklen_t *addrlen);
   int getsockname(..., socklen_t *addrlen);
   int getsockopt(..., socklen_t *optlen);
   ssize_t recvfrom(..., socklen_t *fromlen);
   ssize_t sendto(..., socklen_t tolen);
   int setsockopt(..., socklen_t optlen);
```

```
   struct cmsghdr {
      socklen_t  cmsg_len;
      ....
      };

   struct msghdr {
      ....
      socklen_t  msg_namelen;
      ....
      socklen_t  msg_controllen;
      ....
      };
```

The use of size_t is deprecated for these objects.

Currently, the socklen_t and size_t types are the same underlying type.
This is compatible with both the UNIX 95 and UNIX 98 profiles.

However, in a future release, socklen_t might be a different size.
In that case, passing a size_t pointer will evoke compile-time warnings,
which must be corrected in order for the application to behave correctly.

Applications that use socklen_t will avoid such migration problems.
For portability to the UNIX 95 profile, applications should
follow the X/Open specification (see xopen_networking(7)).