10 Geneology

The Solaris 2.2 kernel TCP/IP software was developed under contract for Sun by Mentat Inc., of Los Angeles, California.

The kernel IP multicast code was developed by Sun from code written for SunOS 4.x by Steve Deering of Xerox Palo Alto Research Center (PARC).

The improvements to the STREAMS framework to support multi-threaded protocol code were developed by Sun.

The socket library and module were developed by Sun as part of the SVR4 joint development with AT&T.

The user-level Internet services (inetd, telnet, rlogin, etc.) were ported by Sun from SunOS 4.x. They were originally ported from the 4.3 BSD release.

ARP packets from the network, nor packets from ARP destined to the network.

ARP maintains a simple database of information, gathered from a variety of sources. Client applications and modules can provide information directly to ARP, and they may request ARP to broadcast ARP Requests for a particular protocol address. Information is also obtained from unsolicited ARP Responses.

ARP does not maintain a cache for its clients. In particular, responses to client requests are simply sent to the client, which ARP assumes will maintain its own cache. In this way ARP is isolated from a client's caching policies, and the client can organize its cache as it sees fit.

ARP presents a uniform command interface to both kernel clients like IP and to applications. The command format permits complete generality in specification of protocol addresses, hardware addresses, and address masks. In particular, ARP has sufficient flexibility to function with IP and Link Layer multicast address formats.

For compatibility with existing applications, IP recognizes the complete set of BSD "ARP ioctls". IP converts these ioctls into ARP commands, sends the resulting command to ARP, and where appropriate, converts ARP's reply back into the format expected by the original ioctl.

9 Configurable parameters

All of the Solaris 2.2 TCP/IP modules support a number of configurable parameters. These parameters may be viewed or set by the system administrator via the **ndd** command. These "ndd'able" parameters should all be considered "under the hood" in the sense that they may change from release to release. Many make sense only to those who are knowledgable of the implementation. But they can provide a useful degree of control to the advanced system administrator. Some of the "ndd'able" parameters are:

TCP -- Change the default connection "notify" and "abort" timeouts; Change the default TTL for TCP connections; Toggle the handling of the urgent pointer compatible with 4.2 BSD; Change the bounds for "anonymous" port numbers (those selected by the kernel when an application binds to port number 0).

UDP -- Change the default TTL for UDP packets; Change the bounds for "anonymous" port numbers.

IP -- Enable/disable IP forwarding; Enable/disable forwarding of directed broadcasts; Enable/disable path MTU discovery; Change the "flush" timeout for IREs.

The default TCP instance also plays a crucial role when a stream is closed while a connection is still established. Although it is possible simply to abort the connection with a reset and free all associated resources when the stream close occurs, Solaris 2.2 TCP supports the notion of a "detached close". When a stream is closing while a connection is still established, TCP saves the closing stream's instance data within the context of the default TCP instance. Since the corresponding IP is also closing, IP will send subsequent inbound packets to the default TCP instance, which can then complete the orderly release and FIN-ACK handshake using the closed stream's saved instance data.

7 UDP and RAWIP

UDP is inherently much simpler than TCP, and, as a result, The Solaris 2.2 implementation offers few surprises. Like TCP, UDP relies on IP for checksum computation, and for delivery of inbound datagrams to the stream or streams bound to the port in the UDP header.

UDP checksums are enabled by default in Solaris 2.2.

The RAWIP module is pushed above IP to provide applications a TPI interface to IP. This module can also be accessed via the socket interface by using socket type SOCK_RAW and with any IPPROTO_xxx protocol type. Special code for support of IPPROTO_ICMP (used by **ping** program) and IPPROTO_IGMP (used by multicast routing) is included in RAWIP. Privileged applications using RAWIP can supply IP headers or ask RAWIP to add them.

8 ARP

The basic features of ARP's design are:

- Basic ARP services are provided in the kernel.
- ARP is not part of the main data path through IP.
- ARP caching policies are left to its client protocols.
- The module is not dependent on any specific Link Layer or protocol.

The left two streams in Figure 1 show how ARP is configured into the kernel, out of the main data path through IP and transport layer modules. One stream permits ARP to send and receive packets from the network without the involvement of IP. The second stream provides a path for communication between IP and ARP. Note that ARP only interacts with the network on the stream directly connected to the device; IP never receives

cast version and a non-multicast version, so that redundant checks resulting from multicast support can be excluded from the main data path.

IP provides full ICMP support as required by RFC 1122. IP maintains statistics as defined by MIB-II database in RFC 1213. Both features are distributed throughout IP. System administrators may view the MIB-II statistics by running the **netstat -s** command.

IP supports the Path MTU Discovery protocol as specified in RFC 1191. This algorithm automatically determines the MTU of the Internet path to any IP destination. TCP uses the path MTU to optimally segment data and avoid IP fragmentation. UDP uses the path MTU to optimally fragment datagrams.

The Solaris 2.2 supports the ICMP Router Discovery protocol as specified in RFC1256. This protocol allows a Solaris 2.2 host to automatically and dynamically configure IP routing without needing to participate in the routing protocol. When operating as an IP router, Solaris 2.2 supports the router specific parts of RFC 1256. The router discovery protocl is implemented as a user-level daemon process.

Added robustness is achieved in Solaris 2.2 IP through its support of multiple "default" routes.

6 TCP

TCP is designed to maximize performance on the main data path for an established connection. This optimization is achieved in part by IP's knowledge of TCP, which permits IP to deliver inbound datagrams to the correct TCP instance and to compute checksums on behalf of TCP. In addition TCP incorporates a novel ACK strategy which can substantially reduce ACK overhead on high performance nets. And, of course, TCP fully implements RFC 1122 requirements for Slow Start, Nagle Algorithm, Round Trip Estimates, and Congestion Avoidance.

6.1 Connection Setup and Tear-down

As discussed above, IP knows enough about TCP headers to deliver incoming datagrams for established connections to the correct TCP instance. All other datagrams for TCP are sent to a single default TCP instance (omitted from Figure 1 for clarity). This default TCP is responsible for establishing connections and orderly release shutdowns. This functional split permits the messy details of connection management to be isolated from the highly optimized main data path through TCP and IP.

stream. Referring to Figure 1, this corresponds to a message being processed in one of the "IP Device" instances of IP being sent directly to a queue for one of the devices below an "IP Module" instance.

If IP has no fully resolved IRE for the destination address, IP proceeds, with the assistance of ARP, to create such an entry from what it knows about directly connected subnets, assigned gateways, default gateways, etc.

When IP requires address resolution, it sends a message with its request to ARP. Assuming that ARP can obtain the Link Layer address, it will send its response message back to IP. At this point IP can update its route table and send the message containing the datagram to the correct device stream.

5.2 Inbound Datagrams

In general, a transport layer module registers with IP to receive datagrams for a specific protocol. All such modules bound to a specific protocol will receive copies of inbound packets marked for that protocol. TCP and UDP are special cases.

Since IP knows about UDP and TCP headers, IP can be more specific for UDP and TCP protocols. By extracting address information from the transport layer header, IP is able to forward inbound datagrams directly to the correct instance of UDP or TCP. Two major benefits obtain by distributing packets to the correct stream at the IP level. First, upstream modules can be simpler, and second, on a multiprocessor system, parallel threads of execution for processing of inbound datagrams can begin sooner.

5.3 Other Features

IP computes all checksums for inbound and outbound datagrams. This includes not only the IP header checksum, but also, in the TCP and UDP cases, the transport checksum for the encapsulated datagram. IP knows enough about TCP and UDP headers to access the checksum fields in their headers. TCP and UDP initialize header fields in such a way that IP can efficiently compute checksums on their behalf. There are some performance efficiencies to be gained by computing the checksum in only one place. Furthermore, this procedure allows checksum computation to be skipped for intramachine datagrams.

IP provides full multicast support as described in RFC 1112 and the DVMRP 1.2 extensions to RFC 1075 for multicast routing. Multicast logic is totally isolated from mainline IP logic. Several functions have both a multi-

4 Compatibility

Solaris 2.2 TCP/IP provides both source and binary compatibility for socket-based applications developed under SunOS 4.x. The Solaris 2.2 socket library and module support all of the commonly used socket semants. The BSD interface and routing ioctls commands, used by **ifconfig**, **arp**, **route** and other utilities, are fully supported. In addition, Solaris 2.2 TCP/IP fully supports the various IPPROTO_IP and SOL_SOCKET level socket options described in Section 6.11 of UNIX Network Programming by R.W. Stevens, Prentice Hall 1990.

Socket options may be set and read using the socket *setsockopt* and *getsockopt* functions in The Solaris 2.2 Socket Library, or they may be manipulated directly using TPI T_OPTMGMT_REQ messages.

5 IP

IP is designed around its job as a packet forwarder. The main data path through IP has been highly optimized for both outbound datagrams to known, fully resolved addresses, and for inbound datagrams to ports which transport layer protocols have registered with IP. Other IP functionality, such as ICMP processing or interface configuration, has been coded to have minimal impact on the performance of IP's primary function of forwarding packets.

5.1 Outbound Datagrams

IP routing relies on an internal data structure called the Internet Routing Entry or IRE. There are several types of IRE. The basic categories are:

- Entries for fully resolved, route-specific addresses, including local addresses.
- Entries identifying a resolver for a specific address.
- Entries for networks.
- Entries for default gateways.

IREs are stored in one of several tables according to category. These tables are created by IP from configuration information supplied by utility programs such as **ifconfig**, and from information gained from ARP. Collectively, the IRE tables may be considered as IP's routing table.

When an outbound datagram is passed to IP, the destination address is looked up in IP's routing table. If a fully resolved IRE is found, i.e. one for which the Link Layer hardware address is already known, the message containing the datagram is sent directly to the corresponding device

rate instances of the same IP code. Likewise, the two “ARP Module” boxes are separate instances of the same ARP code, and the two “DLPI Device 1” boxes represent separate instances of the same driver code.

As shown in the figure, each of the device streams is linked under a “dummy multiplexor” using the STREAMS I_PLINK ioctl. This multiplexor has no functionality other than as an anchoring point for the device streams. The shaded arrows connecting the multiplexor to the modules below it serve to emphasize the fact that the multiplexor serves only as an anchoring point and that no data is passed to it.

3 Multiprocessor issues

The Solaris 2.2 TCP/IP is fully multi-threaded. It was designed from the outset to operate on a multiprocessor system or in a preemptive kernel environment. It takes advantage of some of the new features that were added to the STREAMS framework to support multi-threading:

- Synchronized access to resources such as queues.
- An implicit read-only lock which can be shared among all instances of a family of protocol modules. A reference to this lock is obtained by each thread entering a participating module.
- A write lock which can be obtained by any thread which is already a reader.

The modules assume that no shared data or STREAMS plumbing will change preemptively. Updating of shared data is performed only after obtaining the write lock associated with the current queue, and all other reader threads have exited.

By carefully isolating the need for exclusive access to shared data, The Solaris 2.2 TCP/IP minimizes the overhead needed to obtain locks and maximizes the utility of multiple processors. In particular, the write lock is never obtained while executing the main data path through a module.

In addition to paying careful attention to synchronization, any design which wants to take maximum advantage of multiple processors must introduce as much parallelism as possible. In Solaris 2.2, IP delivers inbound datagrams for UDP and established TCP connections directly to the correct stream, permitting multiple instances of each protocol module to operate simultaneously. Likewise for outbound datagrams, parallel streams are maintained from the time the put procedure of the transpor driver (TCP or UDP) is called until the put procedure of the target device driver is called. The commonly used network interface device drivers are also fully multi-threaded, allowing parallel streams to run until the packet is transmitted.

drivers is set up at the time a system boots either by the *ifconfig* program or, in the case of diskless machines, by the kernel.

For each interface requiring address resolution, e.g., "Device 1" in Figure 1, the configuration program creates a pair of streams: one with just ARP pushed above the device, and one with both ARP and IP pushed above the device. In this way both IP and ARP have direct access to the device and ARP is not in the main data path of packets destined for IP.

For point-to-point devices and others not requiring address resolution, only a single stream is opened with IP pushed above the device; this is the "Device 2" stream in Figure 1.

When an application wishes to open a transport, e.g., TCP or UDP, a stream with IP acting as a STREAMS device is created, and the corresponding transport module is pushed over IP. This operation is facilitated by the SVR4 **autopush** utility. The right two streams of Figure 1 show typical TCP and UDP streams, including optional application interface modules pushed above the transport modules.

All four instances of IP in Figure 1, the two labeled "IP Module" and the two labeled "IP Device", are instances of the same STREAMS code. That is, IP is configured into the kernel as both a device and as a module. The shaded box around the four IP instances emphasizes that these are sepa-

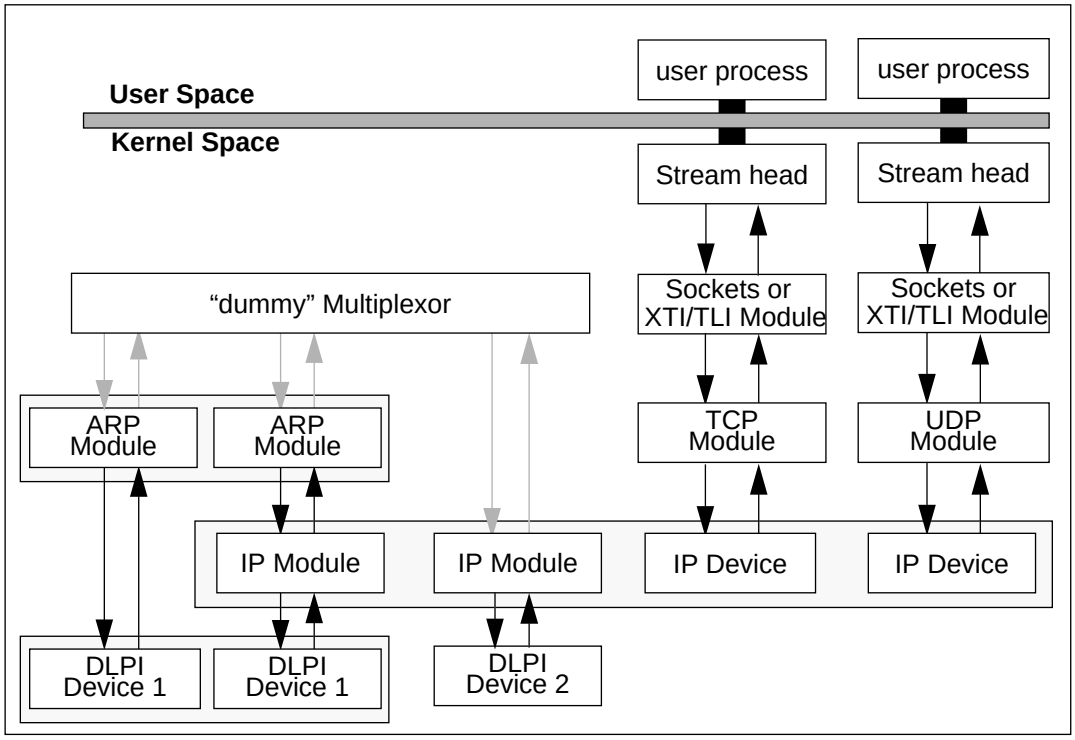

Figure 1: STREAMS configuration of Solaris 2.2 TCP/IP

Solaris 2.2 TCP/IP Design Overview

1 Introduction

This paper provides a functional overview of the kernel TCP/IP software in Solaris 2.2. It is designed to highlight important topics and to present the overall design.

The basic components of Solaris 2.2 TCP/IP are a set of STREAMS modules and kernel support code. The modules are TCP, UDP, RAWIP, IP, and ARP. The kernel support code includes timer support, a generalized module query facility, and miscellaneous functions providing support for common STREAMS operations.

The modules have minimal interdependencies, and communication between them makes use of standard STREAMS mechanisms. TCP, UDP, and the RAWIP module require the IP module, but other transport layer protocols may be used with IP. Likewise, IP and ARP modules are independent of each other. ARP knows nothing about IP and can be used with other protocol families, and IP can work with another address resolution module in place of ARP.

The transport layer protocol modules, TCP, UDP and RAWIP support the Transport Provider Interface (TPI) as the interface to the upper-level modules and applications. Existing modules, libraries, and applications written to use TPI will function without change.

Applications make use of TCP/IP via either the socket interface or the Transport Library Interface (TLI). Both the socket interface and TLI are implemented as a user-level library and a cooperating STREAMS module. In addition, the Solaris 2.2 TCP/IP provides full support for BSD interface and routing ioctls as well as socket options

2 Configuration

Figure 1 shows the basic STREAMS configuration of Solaris 2.2 TCP/IP. Each time the host system is started, a minimal STREAMS configuration must be created. The STREAMS "plumbing" for the network interface