TP-LINK®

TCP/IP协议基本实现

SMB交换 聂勇

2012-10-16

目录

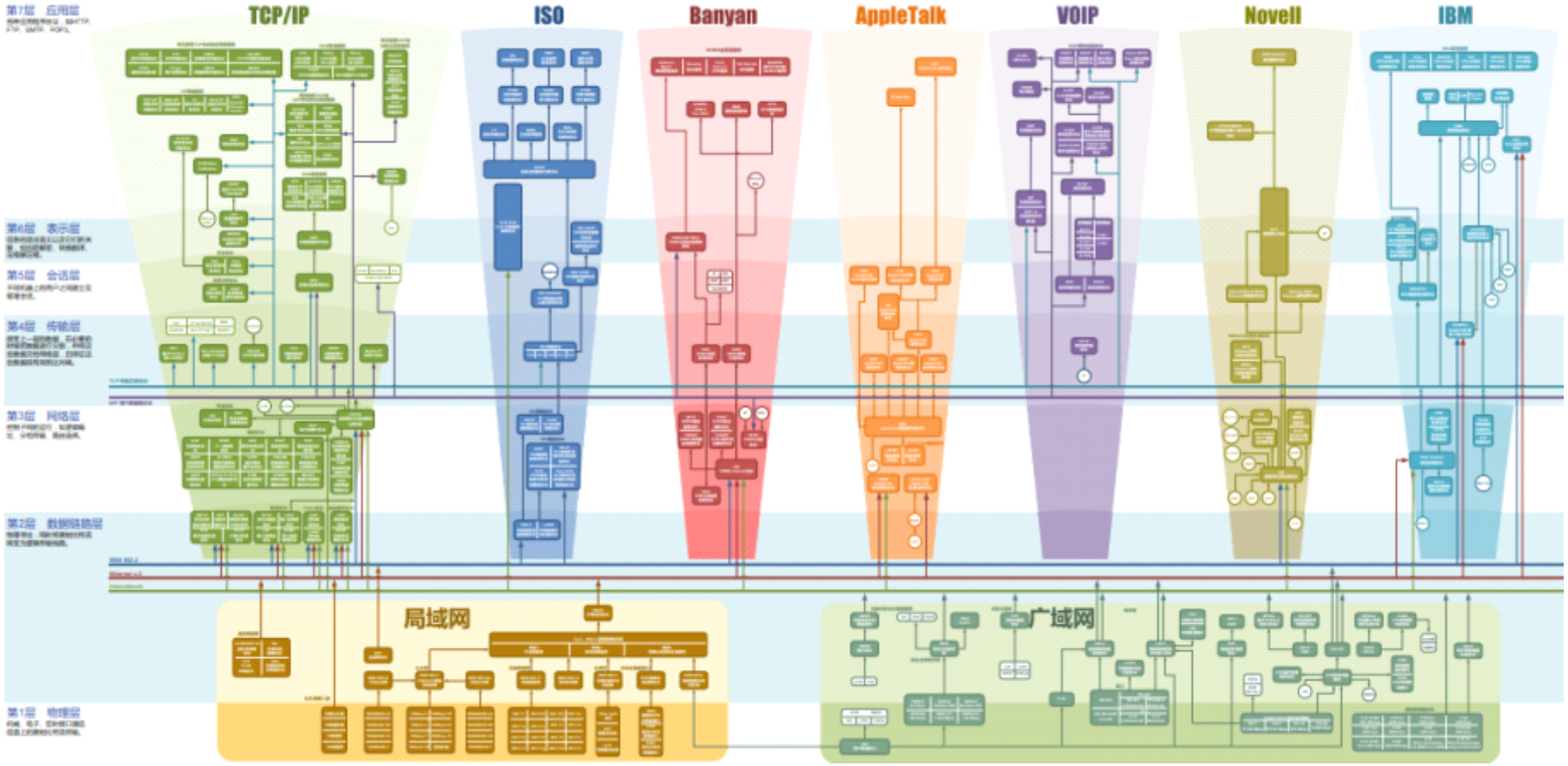
TP-LINK®

第一章 TCP/IP概述

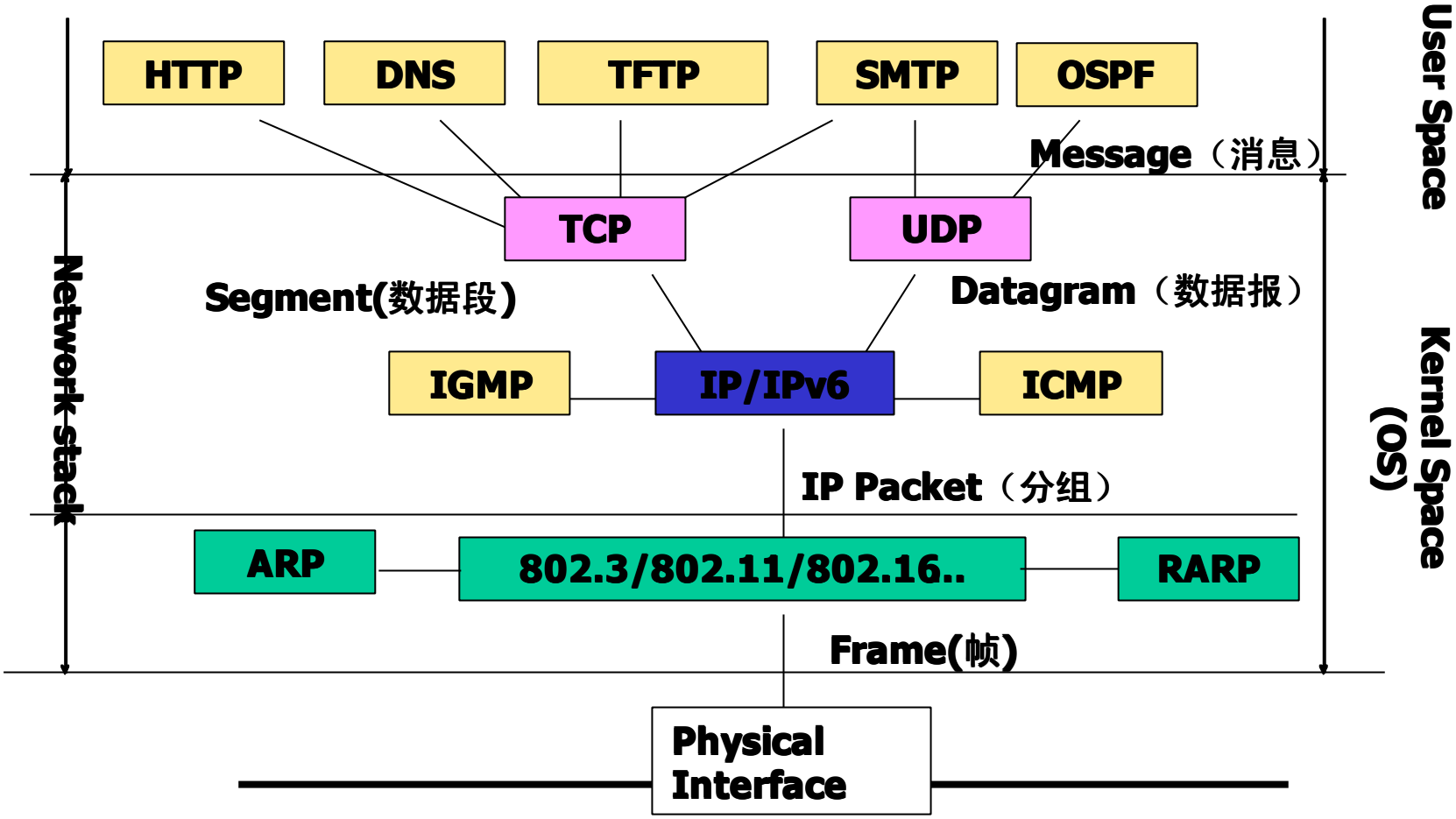
深圳市普联技术有限公司
SHENZHEN TP-LINK TECHNOLOGIES CO., LTD.

TP-LINK

OSI网络模型实现之一
——TCP/IP协议栈



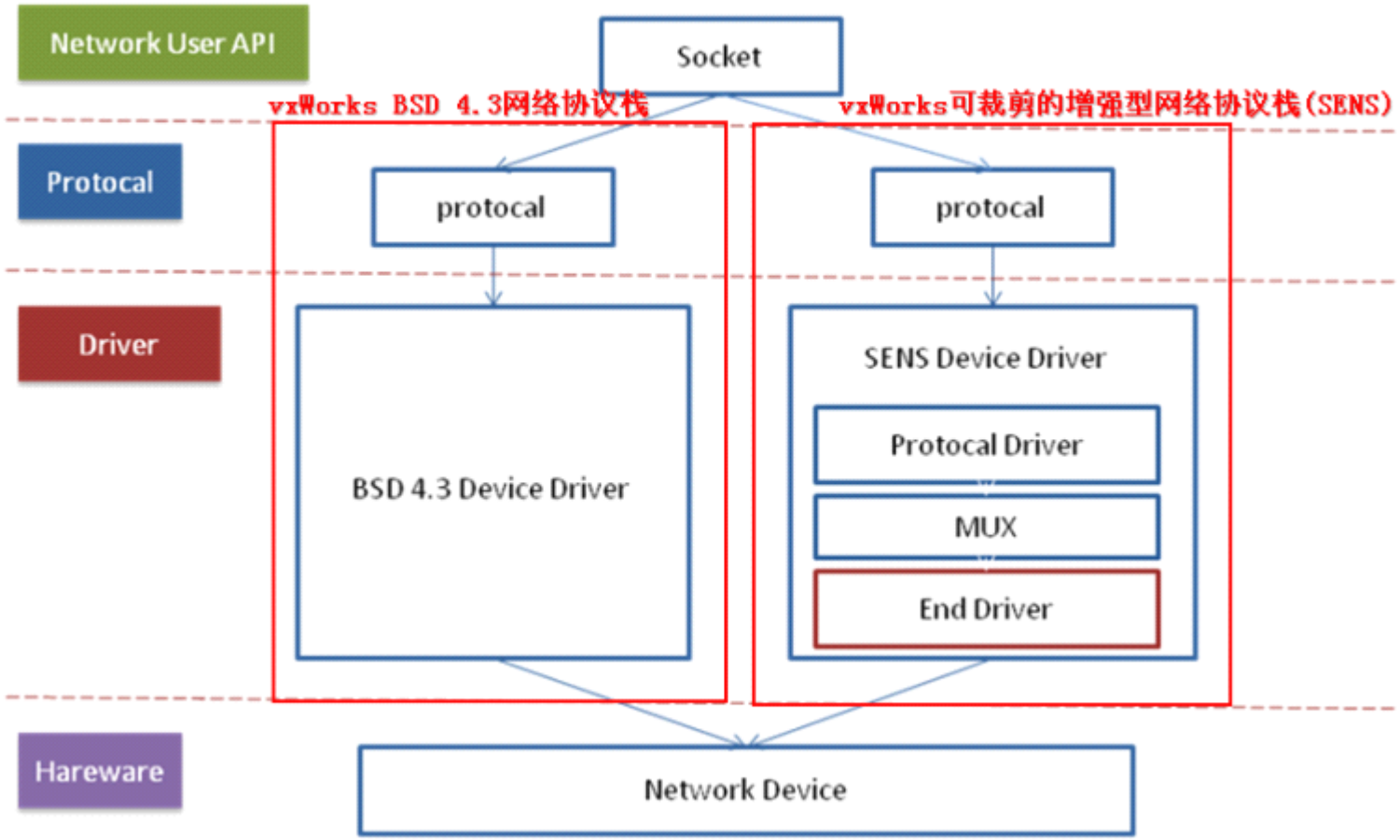
TCP/IP协议栈基本框架



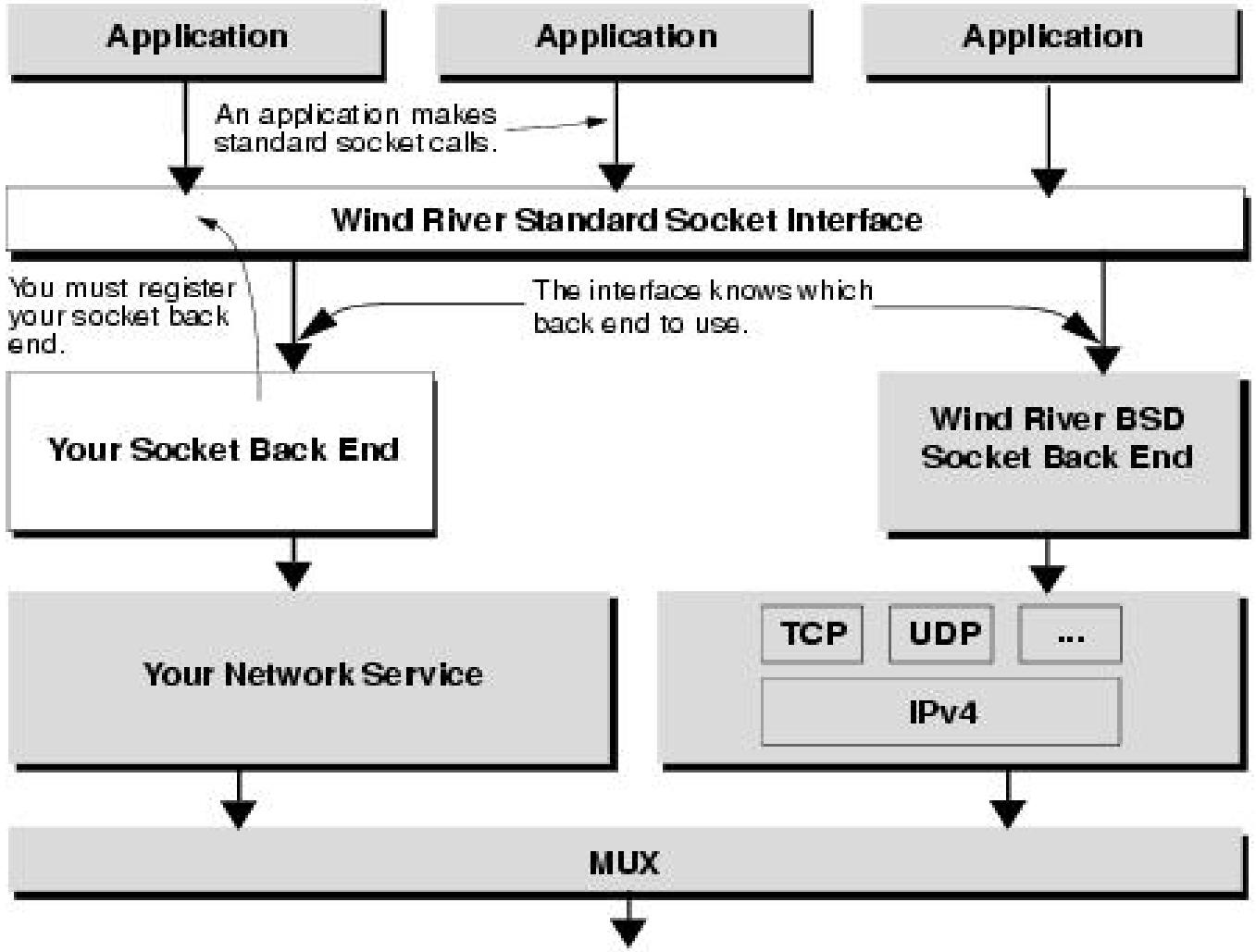
基本术语

- 不同Internet层之间传递数据内容的名词 *（图8-7）*
  - 报文：传输层交给IP的数据，传输层首部+应用数据
  - 数据报：报文+IP首部
  - 分片/分组：数据报太长，进行分片处理。IP层交给数据链路层进行传输的数据叫做分组
  - 帧：分组+以太网头
- BSD 4.3协议栈
  - 又叫做Net/3，在BSD 4.3系统（基于unix）中的TCP/IP实现。该实现是所有Unix或者非Unix系统的起点。

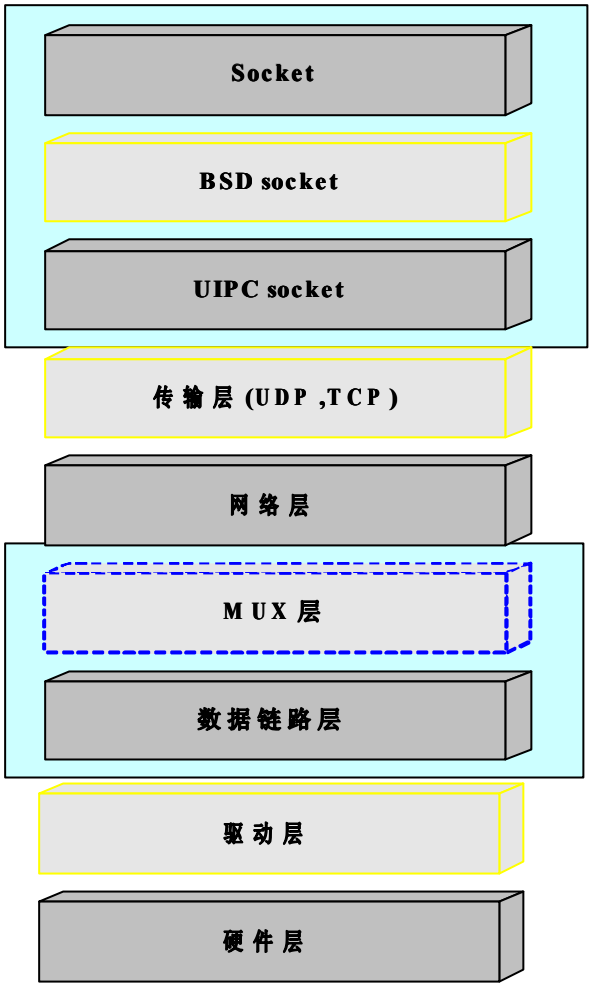
vxWorks的TCP/IP协议栈基本框架



vxWorks的TCP/IP协议栈可裁剪性



vxWorks的TCP/IP协议栈可裁剪性（续）



socket部分如此分层设计，network service部分（OSI中的传输层和网络层）就可以自由的进行裁剪，应用程序不会受到影响

MUX层如此设计，对上可以保证network service部分的自由裁剪，对下保证END驱动部分自由裁剪

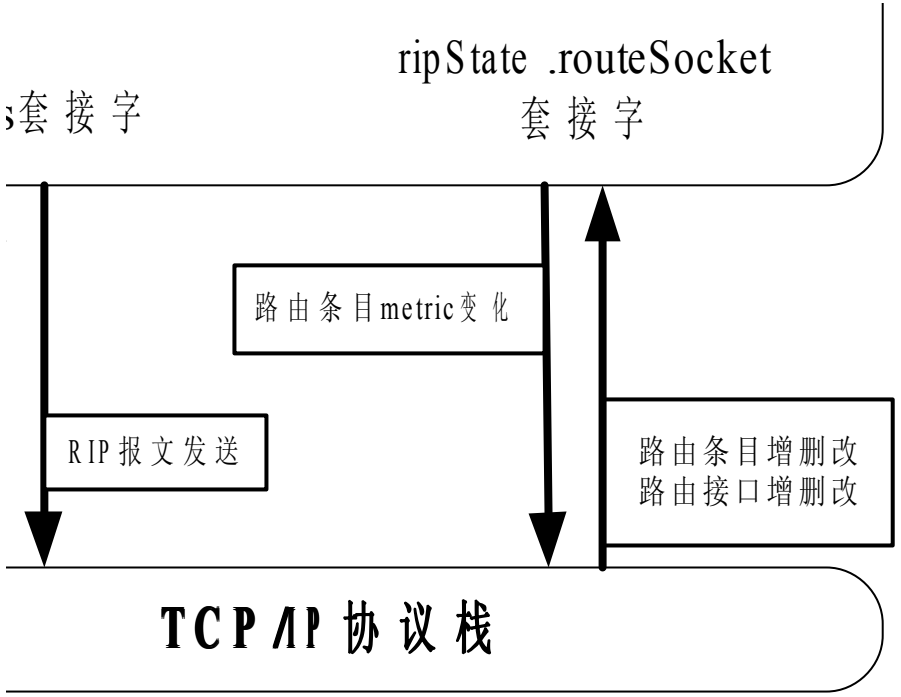
TP-LINK®

第二章 IP和UDP部分的代码实现

—RIP为例讲解

深圳市普联技术有限公司
SHENZHEN TP-LINK TECHNOLOGIES CO., LTD.

RIP与TCP/IP协议栈关系图



创建处理RIP报文的套接字

```
ripState.s = getsocket (AF_INET, SOCK_DGRAM, &ripState.addr);
if (ripState.s < 0)
    {
    log_err (RIP_V4_LOG, "Unable to get input/output socket.");
    priv_ripInitFlag = INIT_FAILED;
    return (ERROR);
    }
```

```
LOCAL int getsocket (domain, type, sin)
    int domain, type;
    struct sockaddr_in *sin;
{
    int sock, on = 1;

    if ((sock = socket (domain, type, 0)) < 0) {
                log_err (RIP_V4_LOG, "Error creating socket.");
        return (-1);
    }
```

在Internet域（AF_INET)创建了SOCK_DGRAM类型的套接字

创建处理路由表的套接字

```
ripState.routeSocket = socket (AF_ROUTE, SOCK_RAW, 0);
if (ripState.routeSocket < 0)
    {
    log_err (RIP_V4_LOG, "Unable to init route socket.");
    close (ripState.s);
    priv_ripInitFlag = INIT_FAILED;
    return (ERROR);
    }
```

在路由域（AF_ROUTE)创建了SOCK_RAW类型的套接字

协议域&协议类型

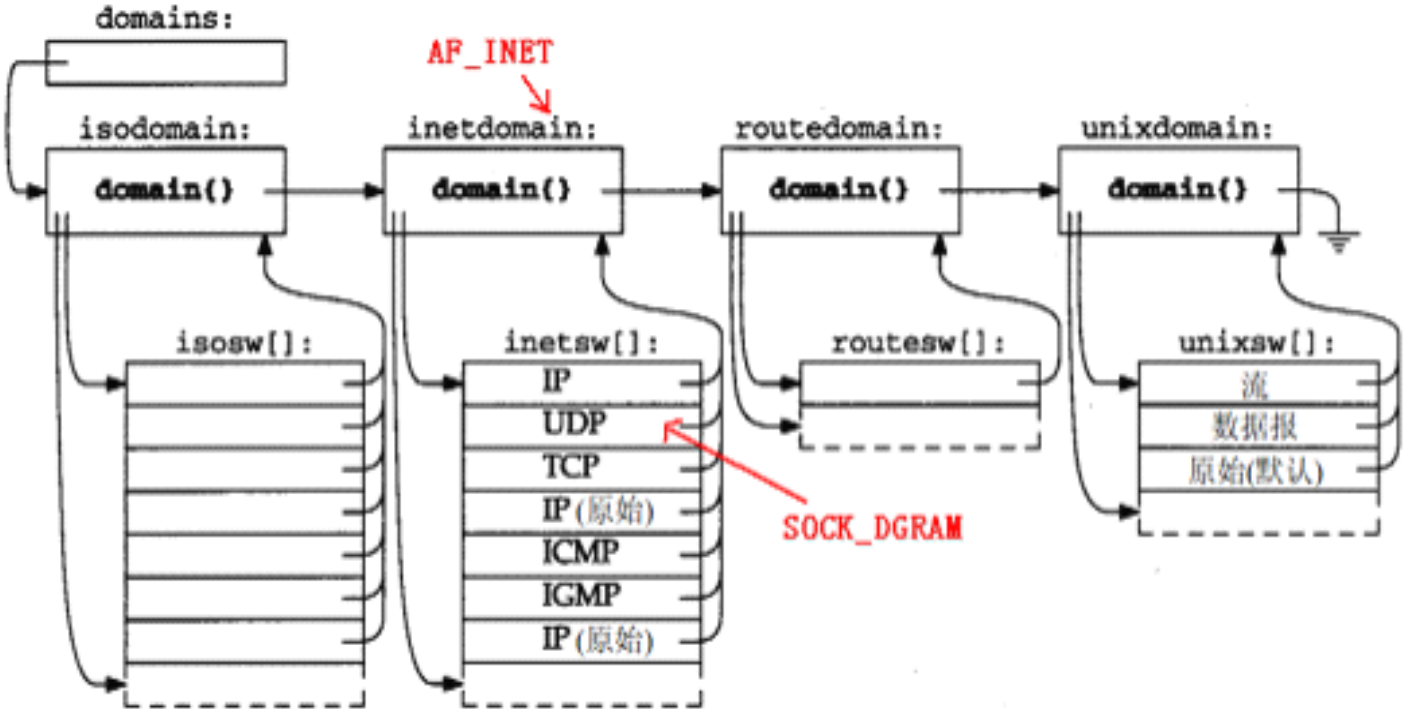


图7-16 初始化后的domain 链表和protosw 数组

每个域代表一种不同的网络协议，并且把该网络协议中的不同协议按照类型进行了区分。

代码讲解（1）

讲解：

协议栈支持同时操作多个网络协议（域、domain)，在代码实现中反映为一个域链表。

```
/*
 * Address families.
 */
#define AF_UNSPEC   0       /* unspecified */
#define AF_LOCAL    1       /* local to host (pipes, portals) */
#define AF_UNIX     AF_LOCAL    /* backward compatibility */
#define AF_INET     2       /* internetwork: UDP, TCP, etc. */
#define AF_IMPLINK  3       /* arpanet imp addresses */
```

示例：

Internet协议（AF_INET)的域节点

```
#ifndef VIRTUAL_STACK
struct domain inetdomain =
    { AF_INET, "internet", 0, 0, 0,
      (struct protosw *)inetsw,
      (struct protosw *)&inetsw[sizeof(inetsw)/sizeof(inetsw[0])], 0,
      in_inithead, 32, sizeof(struct sockaddr_in)
    };
#endif /* !VIRTUAL_STACK */
```

代码讲解（1）

讲解：

每个域包含多个协议，在代码实现中反映为一个protosw*类型的结构体数组。protosw*结构体包含了协议的所有操作函数指针。

```
struct protosw {
    short   pr_type;        /* socket type used for */
    struct  domain *pr_domain;  /* domain protocol a member of */
    short   pr_protocol;        /* protocol number */
    short   pr_flags;       /* see below */
/* protocol-protocol hooks */
    void    (*pr_input)  __P((struct mbuf *, int len));
                    /* input to protocol (from below) */
    int (*pr_output)     __P((struct mbuf *m, struct socket *so));
                    /* output to protocol (from above) */
```

示例：

Internet协议域，protosw*的结构体数组被命名为inetsw。

```
struct ipprotosw inetsw [IP_PROTO_NUM_MAX];
```

代码讲解（1）

综合示例：

下面以udp报文为例讲述IP层对报文的分发。

在*udpInstInit()*函数注册*udp_input()*函数到inetsw数组中。

```
if (_protoSwIndex >= sizeof(inetsw)/sizeof(inetsw[0]))
return (ERROR) ;

pProtoSwitch = &inetsw [_protoSwIndex];
        pProtoSwitch->pr_type       =  SOCK_DGRAM;
        pProtoSwitch->pr_domain     =  &inetdomain;
        pProtoSwitch->pr_protocol   =  IPPROTO_UDP;
        pProtoSwitch->pr_flags  =  PR_ATOMIC | PR_ADDR;
        pProtoSwitch->pr_input  =  (VOIDFUNCPTR)udp_input;
        pProtoSwitch->pr_output =  0;
        pProtoSwitch->pr_ctlinput   =  udp_ctlinput;
        pProtoSwitch->pr_ctloutput  =  ip_ctloutput;
        pProtoSwitch->pr_ousrreq    =  0;
```

在*ip_input()*函数中，根据报文类型调用*udp_input()*函数。

```
    _ipstat.ips_delivered++;
    (*inetsw[ip_protox[ip->ip_p]].pr_input)(m, hlen, ip->ip_p);
    return;
```

代码讲解（1）

综合示例：

根据domain和type查找到相应的protosw结构体。用于socket的创建。*socreate()*函数。

```
struct protosw *
pffindtype(family, type)
    int family;
    int type;
{
    register struct domain *dp;
    register struct protosw *pr;

    for (dp = domains; dp; dp = dp->dom_next)
        if (dp->dom_family == family)
            goto found;
    return (0);
found:
    for (pr = dp->dom_protosw; pr < dp->dom_protoswNPROTOSW; pr++)
        if (pr->pr_type && pr->pr_type == type)
            return (pr);
    return (0);
}
```

扫描domains网络协议域链表

扫描inetsw[ ]结构体数组

数据结构图（2）

传输层（TCP/UDP)层、套接字的处理都是围绕着右图进行的。

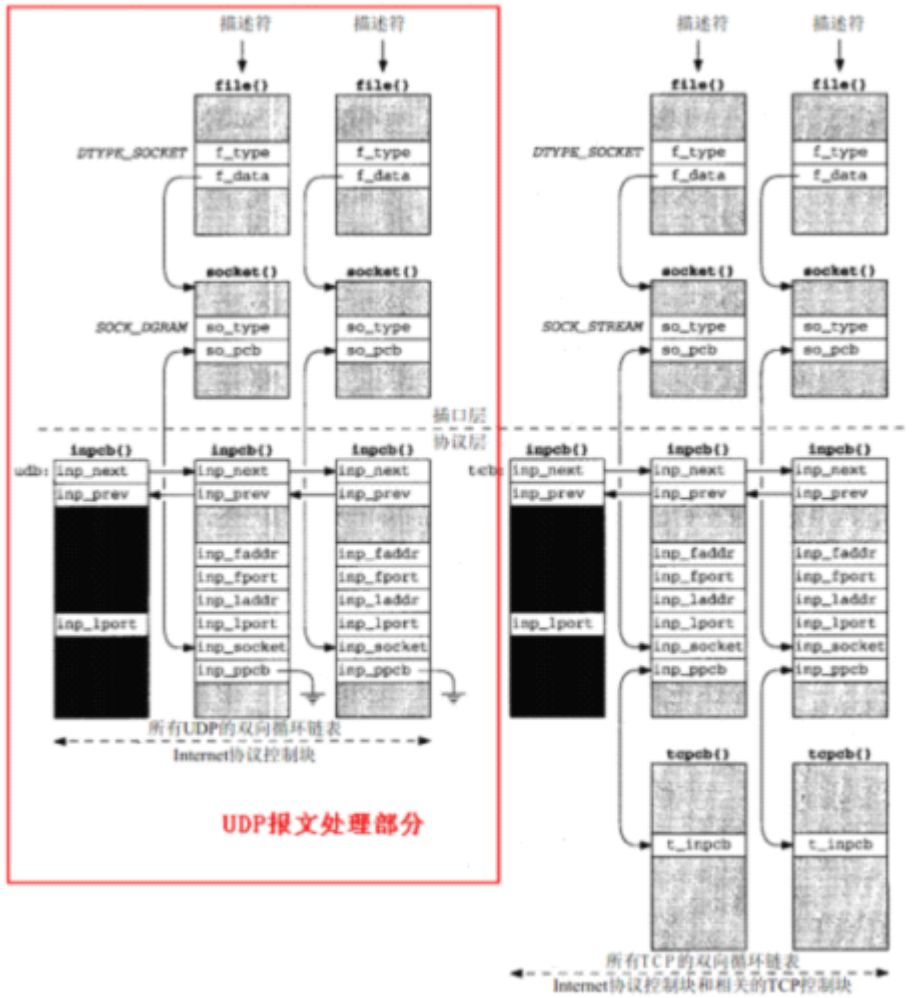


图22-1 Internet协议控制块以及与其他结构之间的关系

代码讲解（2）

讲解：

协议层使用协议控制块（Protocol Control Block）存放各UDP和TCP套接字所要求的多个信息片，分为Internet协议控制块和TCP协议控制块。

示例：

Internet协议控制块链名udb和TCP协议控制块链名tcb。

```
struct inpcbhead  udb;                  /* head of the UDP PCB list */
```

```
struct inpcbhead tcb;
#define tcb6    tcb  /* for KAME src sync over BSD*'s */
```

代码讲解（2）

讲解：

domain链表和protosw数组数据结构在协议初始化时完成，socket和udb链表、tcb链表数据结构是应用程序动态生成。

示例：

以SOCK_DGRAM类型套接字注册为例。具体流程参考*socreate()*函数和*udp_attach()*函数。

```
so = (struct socket *)DS_MALLOC(SOCKET_DS_SZ, SOCKET_DS_ID);

if (so == NULL)
    return (NULL);
```

```
    inp=(struct inpcb *)DS_MALLOC(pcbinfo->ipi_size,pcbinfo->ipi_zone);
if (inp == NULL) {
```

代码讲解（2）

综合示例：

udp对单播报文的接收处理过程。*udp_input()*函数。

查找接收该udp报文的pcb。

```
/*
 * Locate pcb for datagram.
 */
inp = in_pcblookup_hash(&udbinfo, ip->ip_src, uh->uh_sport,
    ip->ip_dst, uh->uh_dport, 1, m->m_pkthdr.rcvif);
if (inp == NULL) {
```

把数据加入到套接字接收队列。

```
if (sbappendaddr(&inp->inp_socket->so_rcv, append_sa, m, opts) == 0) {
            WV_NET_PORTIN_EVENT_2 (NETD_IP4_DATAPATH_EVENT, WV_NETD_WARNING,
                                   1, 9, uh->uh_sport, uh->uh_dport,
                                   WV_NETDEVENT_WARNING, WV_NETD_RECV,
                                   udp_input, WV_NETD_FULLSOCK)

    udpStat.udps_fullsock++;
    goto bad;
}
sorwakeup(inp->inp_socket);
```

代码讲解（2）

综合示例：

套接字单播报文的发送过程。*sosend()*函数。

```
error = (*so->so_proto->pr_usrreqs->
     pru_send)(so, sendflags, top, addr, control, p);
```

数据结构图（3）

对前面两张图的总结：

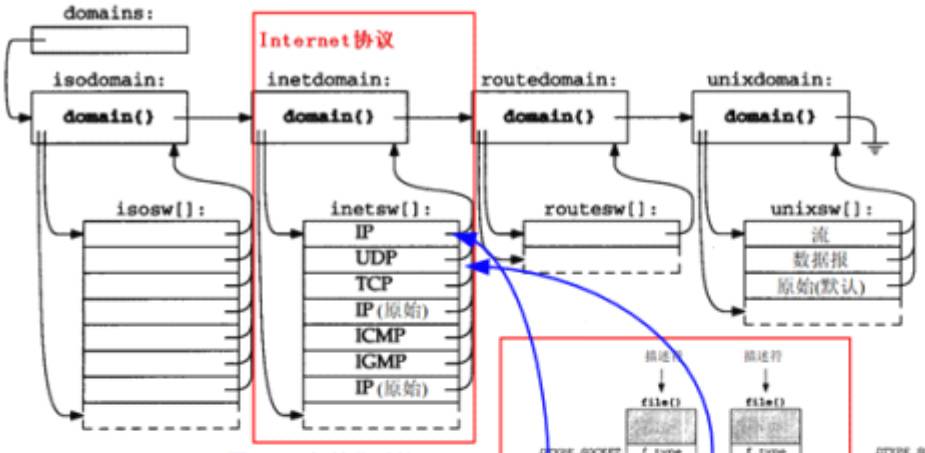


图7-16 初始化后的domain

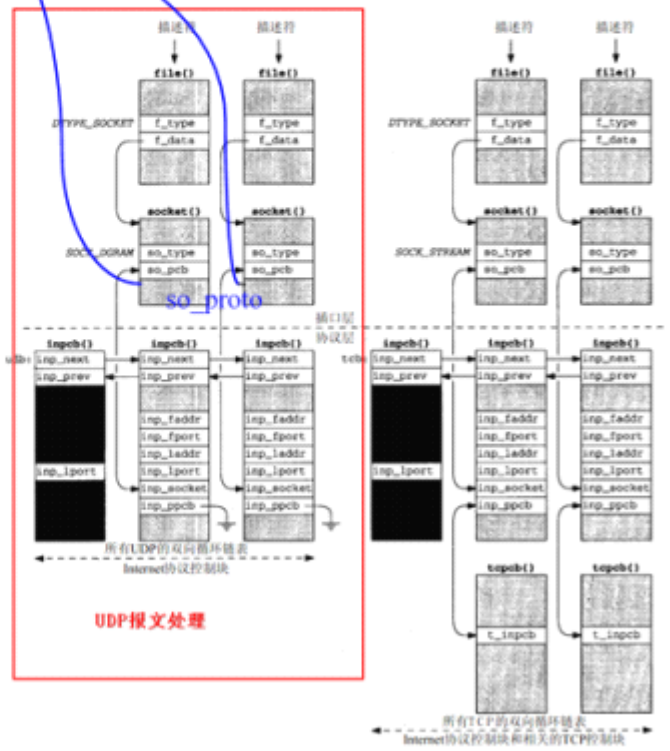


图22-1 Internet协议控制块以及与其他结构之间的关系

TP-LINK®

第三章 vxWorks中特有的部分

深圳市普联技术有限公司
SHENZHEN TP-LINK TECHNOLOGIES CO., LTD.

vxWorks协议栈特有（1）
——zbuf socket

zero-copy buffer socket

实现buffer在应用层与协议层之间无需拷贝。

vxWorks协议栈特有（2）
——RTP/RTPs

Real Time Process

应用无需编译到vxWorks image中，shell下直接执行。5.5.1及以下不支持。

实现原理：Kernel execution mode

vxWorks协议栈特有（3）
——Virtual TCP/IP Stacks

讲解：

vxWorks中运行2个TCP/IP stack instance。

stack共享一个任务（tNetTask）

stack的数据时独立的（routing table/network buffer memory pool/localized global variales）

vxWorks协议栈特有（3续）
——Virtual TCP/IP Stacks

示例：

ISPs的Virtual Private Routed Networks（VPRNs）

```
/* macros for "non-private" global variables */

#define VS_UDP_DATA ((VS_UDP *)vsTbl[myStackNum]->pUdpGlobals)

#define udpcksum              VS_UDP_DATA->_udpcksum
#define udp_log_in_vain       VS_UDP_DATA->_udp_log_in_vain
#define udb                   VS_UDP_DATA->_udb
#define udbinfo               VS_UDP_DATA->_udbinfo
#define udpstat               VS_UDP_DATA->_udpstat
#define udp_sendspace         VS_UDP_DATA->_udp_sendspace
#define udp_recvspace         VS_UDP_DATA->_udp_recvspace
```

更多（1）
——END驱动

参考《Hawkeye End驱动研究与分析-邱俊源.ppt》

参考《BCM47xx驱动分析.doc》——杨晓强

更多（3）
——协议栈内存池

参考《Receive and Send Packets on xCat.doc》
第3节 ——VxWorks网络体系结构（by刘家宁）

参考《VxWorks协议栈研究报告(正式版).doc》
第4节 ——内存池管理（by杨晓强） 或者《协议栈内存管理分析.doc》（by杨晓强）

更多（4）
——路由

UNIX系统 & BSD 4.3