MacTCP 1.0 uses MPW C 2.1 calling conventions.

Building Alternate Link Access Protocol Modules for MacTCP

MacTCP version 1.1 contains built-in LocalTalk and Ethernet link access protocol (LAP) support. Support for additional LAPs can be added through hooks in the MacTCP drivers. This document describes how to add alternate LAP modules to MacTCP.

Note: Alternate LAP support is not a tested and supported feature of MacTCP 1.1. While we believe this information is correct, Apple is not responsible for the accuracy of this information and reserves the right to make incompatible interface changes until the final specification is published.

References

- *EtherTalk and Alternate AppleTalk Reference* (Engineering Part No. 6588279)
- *MacTCP Administrator's Guide* (APDA No. M0261LL/A)
- *MacTCP Programmer's Guide* (APDA No. M0261LL/A)

Overview of MacTCP Operation

MacTCP is implemented as a Macintosh driver that loads itself at system start-up time using an INIT-31. MacTCP is partitioned into

- a basic driver that implements all of IP, ICMP, UDP and TCP
- independently written LAP modules that interconnect IP with the link device drivers

A LAP implements the data link layer of the protocol stack which deals specifically with hardware addressing and data encapsulation. Ethernet's data link layer encapsulates the upper layer protocol data in an Ethernet packet which consists of a 48-bit Ethernet source, destination identifiers, and the packet type of the data. Every hardware network interface has a specific protocol for encapsulating the upper layer protocol data.

In addition to encapsulating data, a MacTCP LAP must also handle the link-specific mechanism for converting IP addresses into link addresses (such as ARP on Ethernet), link-specific configuration procedures (such as KIP on LocalTalk), and any other link-specific functions, such as address probes to support MacTCP dynamic address acquisition.

MacTCP is provided with LocalTalk and Ethernet link layer interfaces. Other link layer protocols that are used in the TCP/IP environment include token ring, serial line (SLIP), and 802.2 Ethernet.

The LAP modules are written, compiled, linked, and loaded independently from the basic driver. IP and LAPs communicate through a pre-defined interface that includes a dispatch table and

MacTCP 1.0 uses MPW C 2.1 calling conventions.

shared data structures. LAP procedures are called using MPW C[1] calling conventions and must access arguments and preserve registers appropriately.

Alternate LAP Files

MacTCP supports alternate LAPs by interpreting resources stored in MacTCP LAP files in the System Folder. These files must be of file type ‘mdev’. The developer can specify any creator signature. These ‘mdev’ files are similar in functionality to the ‘adev’ file used by AppleTalk. The ‘mdev’ file has the ability to add link layer protocol support to MacTCP. This support includes selection of the LAP used by MacTCP from the MacTCP Control Panel interface. The minimum resources the ‘mdev’ LAP file must contain are described in the following table:

Resource type	Resource ID	Description
mdev	-4032	Returns Control Panel information
ICN#	-4032	Contains LAP icon
iplp	-4032	Contains LAP code resource
BNDL	128	Finder information
FREF	128	Finder information
owner resource		Finder information

The ICN# and STR resources are the icon and the string that the MacTCP ‘cdev’ displays in the LAP section of the Control Panel window. In addition, if the ‘mdev’ file contains the BNDL, FREF, and an owner resource, and if the ‘mdev’ file has its bundle-bit set, the ICN# will appear in the Finder. (Refer to *Inside Macintosh*, Volume 1, for more information about the ICN#, BNDL, FREF, and owner resources.)

The following sections describe how MacTCP uses these resources There are three different stages of interaction between MacTCP and a LAP: configuration, system startup, and operation.

Configuration

When the MacTCP Control Panel is opened, the ‘cdev’ displays the LAP information for the AppleTalk—and, if available—Ethernet-type LAPs that are built into version 1.1. Then it searches the System Folder for files of type ‘mdev’ which contain the alternate LAP information and code resources for MacTCP. For each ‘mdev’ file found, the Control Panel attempts to load an ‘mdev’ resource with an ID of -4032. If the resource is not found, the Control Panel continues scanning the System Folder.

As the ‘cdev’ accesses each ‘mdev’ file, each ‘mdev’ responds by passing various information back to the MacTCP ‘cdev’, including the icon and icon string to be displayed.

[1]MacTCP 1.0 uses MPW C 2.1 calling conventions.

MacTCP 1.0 uses MPW C 2.1 calling conventions.

Each ‘mdev’ file may

support more than one link of the same LAP type and, if so, must also return information about the number of identical icons to display and the strings for each. For example, in Ethernet the MacTCP ‘cdev’ places the string “Ethernet(*n*)” under each icon, where *n* equals each card’s slot number.

If loading was successful, the Control Panel calls the ‘mdev’ resource at offset 0 using the following calling conventions:

Call:	A0 (long)	Address of the GetMDEV routine being called
	D0 (long)	101 == ‘Get MDEV information’
	D2 (long)	Value returned from previous GetMDEV call, or 0 if first call
Return:	A0 (long)	Pointer to Pascal-format string to place under icon
	D0 (byte)	Status flag
	D1 (word)	NuBus slot number
	D2 (long)	Value to give next GetMDEV call in D2

On return, the ‘mdev’ routine should return a status value of -1 or 0 in D0 to indicate a valid return. In this case, A0 points to a Pascal-format string to be displayed under the icon, D1 contains the NuBus slot number of the board, and D2 can be used for communication with the next call to the ‘mdev’ routine. The NuBus slot number is passed uninterpreted to the LAP’s initialization routine and can then be used for private communication. It is recommended that the LAP use 0 for machines without NuBus slots and NuBus machines with a single link card so that users can reconfigure their hardware without reconfiguring MacTCP.

Startup

At system startup time, the MacTCP INIT-31 code loads the main driver code into the system heap[2] and reads the IP configuration information from the ‘ipln’ resource contained in either the MacTCP prep file or the MacTCP driver file. The configuration information includes the name of the file that contains the LAP code. If the name is NIL, the LAP code resides in the MacTCP driver file. The *activeLap* field contains the resource ID of the LAP code resource.

The INIT-31 code then loads the indicated LAP code resource (of type ‘iplp’) into memory with the driver (usually the system heap) and locks it down. This resource contains the code that actually encapsulates and transmits the LAP-specific packets.

As a final start-up step, the LAP is called at offset 0 using C calling conventions to allocate a LAP_Info record, fill in LAP parameters, and return a pointer to this structure. All subsequent calls to the LAP are made through this record to the individual procedures. For a C language description of the LAPInfo record, refer to the section “struct LAPInfo {}” in the appendix of this

[2]On a Macintosh 512K enhanced computer, MacTCP does not load at INIT-31 time but rather must be installed into the device table using the Installer. It opens when the application opens the MacTCP driver and is loaded into the application heap. MacTCP removes itself when the application heap is reinitialized.

MacTCP 1.0 uses MPW C 2.1 calling conventions.

document. The LAP should allocate any memory it might need later at this time[3], rather than at LAP_init time, since the system heap may be resized such that memory allocations may fail.

The LAPInfo structure is comprised of a fixed part that is the same for all LAPs that contain a dispatch table and LAP operating parameters, followed by a variable length part whose format is defined by each LAP. This variable part can include storage for ARP tables, buffers, control blocks, or any other storage needed by the LAP. The list of LAP parameters that need to be initialized are described in the section "struct LAPInfo *LAP_main {}" in the appendix of this document.

After a pointer to the LAPInfo structure is returned to the driver, the IP configuration information is copied into it.

Operation

When the MacTCP driver is opened for the first time, it initializes all of its protocols, starting with the LAP and proceeding through IP, ICMP, TCP and UDP. IP calls the LAP_init code by calling through the dispatch table that was set up earlier. The LAP code can access its configuration information (i.e., the slot number) stored in the LAPInfo structure. The LAP code shouldn't modify any of the fields in the IP configuration resource.

Note: MacTCP 1.1 limits the size of the LAP header that is automatically allocated in the read and write buffers to 48 bytes.

Serial Line Connections

Serial connections should be treated differently than other LAP interfaces because serial lines are used most efficiently by disconnecting them after the network transaction has been completed. Since MacTCP has no way of determining when a link has become idle, it should be shut down; therefore, the serial link interface should ask the user for a time interval that will prompt him/her if the line should be disconnected. The user should always be able to override disconnection though it may be prudent to force disconnection if there is no response.

When the link has disconnected the procedure passed in, the LAP_Init call should be called notifying MacTCP that the link is no longer active. MacTCP will call the link configuration call when the link is to be used again by the application. This interface is not only of interest to serial lines but also to any interface that may need to be reconfigured over a period of time.

[3]The MacTCP driver handles the differences in heaps depending on whether or not it is running on a 512Ke and the LAP can assume that any NewPtr allocation will happen from the correct heap.

MacTCP 1.0 uses MPW C 2.1 calling conventions.

VM Issues

System 7.0 will make virtual memory available on Macintosh computers. MacTCP expects that all calls to virtual memory will be made as a deferred function call. The functional specification for VM goes into greater detail on the details. For more information, see *Inside Macintosh*, Volume VI.

MacTCP 1.0 uses MPW C 2.1 calling conventions.

Appendix: Interface Details

This appendix defines the primary interface between IP and the LAP.

struct IPConfig {}

IPConfig holds all of the IP and LAP-specific configuration information. This information is read in from the 'ipln' configuration resource.

```
struct IPConfig {
    int             version;                /* version number */
    long            flags;                  /* options flags enabled for this LAP */
    long            dfl_ip_addr;            /* default IP address */
    long            dfl_net_mask;           /* default subnet mask */
    long            dfl_broadcast_mask;     /* default broadcast mask */
    long            dfl_gwy_addr;           /* default gateway address */
    unsigned char   server_lap_addr[8];     /* default server lap address (8 bytes) */
    long            configIPAddr;           /* mechanism by which IP addr was obtained */
    long            configNetMask;          /* mechanism by which net mask was obtained */
    long            dfl_dyn_low;            /* dynamic address field lower bound */
    long            dfl_dyn_high;           /* dynamic address field upper bound */
    String(32)      dfl_zone;               /* AppleTalk gateway zone */
    /* align to word */
    Boolean         load;                   /* true - load driver into system */
    Boolean         admin;                  /* true - allow admin cdev functions */
    Boolean         netLock;                /* true - lock net number */
    Boolean         subnetLock;             /* true - lock subnet number */
    Boolean         nodeLock;               /* true - lock node number */
    Boolean         filler1;                /* filler */
    int             activeLap;              /* ID of LAP */
                                            /* builtin: 128 - AppleTalk, 129 - Ethernet */
    int             slot;                   /* slot number if Mac II */
    String(32)      filename;               /* file name if external net interface */
    };

#define String(size) struct {unsigned char length; unsigned char text[size];}

#define ipBadWDSerr  -23034                 /* error in WDS format */
```

The flags field has the following defined values:

```
#define useFixedAddr  0x800                 /* used fixed node address */
#define useDynamic    0x400                 /* pick node address dynamically */
#define useRARP       0x200                 /* use reverse-ARP to get IP address */
#define useSMQ        0x100                 /* use ICMP subnet mask query packet */
#define useBootP      0x40
#define useKIPGW      0x008                 /* use a KIP gateway */
#define useFixedGW    0x004                 /* use fixed gateway address */
#define useRIP        0x002                 /* use RIP to find gateways*/
#define useProxyARP   0x001                 /* use proxy-ARP on this net */
```

struct LAPInfo {}

LAPInfo is allocated during system start-up and holds all of the LAP-specific information.

```
struct LAPINFO {
   b_32             our_ip_addr;        /* LAP's IP address */
   b_32             our_net_mask;       /* LAP's IP net-mask */
   b_32             ip_broadcast_addr;  /* IP's broadcast address */
   struct IPConfig  lc;                 /* copy of IP LAP configuration resource */
   OSErrProcPtr     lapInit;            /* pointer to once-only LAP init routine */
   OSErrProcPtr     lapOpen;            /* LAP open routine */
   OSErrProcPtr     lapClose;           /* LAP close routine */
   voidProcPtr      lapUnload;          /* LAP unload routine, undoes LapInit */
   OSErrProcPtr     lapAttach;          /* LAP attach PH routine */
   OSErrProcPtr     lapDetach;          /* LAP detach routine */
   OSErrProcPtr     lapOutput;          /* LAP output routine */
   OSErrProcPtr     lapControl;         /* LAP control routine */
   voidProcPtr      lapFault;           /* LAP fault isolation routine */
   OSErrProcPtr     lapStatistics;      /* LAP statistic reading routine */
   voidProcPtr      lapConfigure;       /* LAP-specific configuration routines */
   BooleanProcPtr   lapProbe;           /* send a LAP-specific address probe packet */
   BooleanProcPtr   lapRegister;        /* register our IP address on the network */
   voidProcPtr      lapFindGateway;     /* LAP-specific means of finding a gateway */
   BooleanProcPtr   lapGwyCheck;        /* LAP-specific means of verifying gateway up */
   /* IP parameters */
   ip_addr          dfl_dns_addr;       /* address of DNS from config protocol */
   Handle           dnslHndl;           /* handle to DNS configuration resource */
   Ptr              dnsCache;           /* pointer to space allocated for dns cache */
   int              dnsCacheSize;       /* size of cache allocated, in bytes */
   /* LAP parameters */
   int              headerSize;         /* LAP header space requirements */
   int              trailerSize;        /* LAP trailer space requirements */
   int              outMaxPacketSize;   /* size of maximum output packet */
   int              inMaxPacketSize;    /* size of maximum input packet */
   int              maxDataSize;        /* size of maximum data packet */
   int              numConnections;     /* number of separate network connections */
   unsigned long    versionFlags;       /* version number flags */
   voidProcPtr      ip_ph;              /* pointer to IP's protocol handler */
   Ptr              ipGlobals;          /* pointer to IP's A5 */
   short            link_unit;          /* unit number of link driver */
   Boolean          addressConflict;    /* TRUE if address conflict discovered */
   int              lapType;            /* IP LAP hardware type number */
   int              lapAddrLength;      /* size of LAP address field */
   unsigned char    *lapAddrPtr;        /* pointer to LAP address field */
   unsigned long    reserved;           /* MacTCP reserved field */
   };
```

Any entry in the dispatch table can be defined as NIL; the driver checks before calling any LAP routine to see if it is defined. The following LAPInfo fields are of interest to the LAP module developer:

lc	A copy of the IPConfig record read from disk. The only fields of interest are the slot and dfl_gwy_addr.
headerSize	Size of the LAP link header in bytes. Used by IP to allocate buffers.
outMaxPacketSize	Not currently used.
inMaxPacketSize	Not currently used.

MacTCP 1.0 uses MPW C 2.1 calling conventions.

maxDataSize	Size of the largest IP datagram, including IP header, that can be sent over the link. Used to fragment outgoing packets.
lapType	Used to construct LAP-independent BootP requests, lapType is the type of LAP address in the BootP request packet.
lapAddrLength	Used to construct LAP-independent BootP requests, lapAddrLength is the length, in bytes, of the LAP address in the BootP request packet.
lapAddrPtr	Used to construct LAP-independent BootP requests, this points to the LAP's link address.
ipGlobals	The LAP must ensure that IP's register A5 (global data pointer) is set up before calling the IP packet reception or write completion routines.
addressConflict	A flag by which the LAP can communicate back to IP that another node is using the same IP address we are using. This flag is primarily used during the dynamic assignment of node addresses and also checked after a call to the LAP_register routine (see below).

MacTCP 1.0 uses MPW C 2.1 calling conventions.

struct IPBuf {}

The IPBuf is the standard buffer passed between protocol modules inside MacTCP. It is composed of a Macintosh Operating System I/O parameter block, space for saving the I/O completion routine pointer of higher protocol levels, and eight write data structure (WDS) entries. The WDS entries form a length, pointer pair to allow MacTCP to gather writes. The WDS's are terminated by an entry with 0 length.

```
typedef struct wdsEntry {
    unsigned short      length;        /* length of buffer */
    char *              ptr;           /* pointer to buffer */
    } wdsEntry;

struct icmpEchoInfo {
    unsigned long echoRequestOut;
                        /* time in ticks of when the echo request went out */
    unsigned long echoReplyIn;
                        /* time in ticks of when the reply was received */
    struct rdsEntry echoedData;
                        /* data received in responce */
    Ptr options;
                        /* IP Options */
    unsigned long userDataPtr;
                        /* userDataPtr for app stuff */
    };


typedef struct ipbuf {
    CntrlParam          iop;           /* MAC OS I/O Param block */
    struct icmpEchoInfo echoInfo;
    struct ipbuf        *segipb;       /* pointer to segment's IPB if fragmented */
    struct LapInfo      *lap;          /* LAP pointer */
    ProcPtr             lap_ioc;       /* local net completion routine */
    ProcPtr             ip_ioc;        /* IP completion routine */
    ProcPtr             tp_ioc;        /* transport completion routine */
    ProcPtr             data_ioc;      /* data IOC */
    struct wdsEntry      laphdr;       /* local net header */
    struct wdsEntry     ip;            /* IP header */
    struct wdsEntry     tp;            /* TCP/UDP or ICMP header */
    struct wdsEntry     data;          /* TCP/UDP data */
    struct wdsEntry     d1;
    struct wdsEntry     d2;
    struct wdsEntry     d3;
    struct wdsEntry     d4;            /* 8 wds entries plus 0 terminator */
    short               flag;          /* zero terminator to WDS */
    char                packet[];      /* start of variable length pkt */
} ipbuf;
```

struct LAPInfo *LAP_main()

LAP_main is called during system start-up to allocate a LAP_Info structure and fill in the dispatch table and basic LAP parameters. The LAP_Info structure must begin with the fixed fields defined earlier; however, the LAP is free to allocate additional memory following this fixed header for other uses such as buffers, ARP tables, etc. LAP_main is only called at the application level.

```
struct LAPInfo *LAP_main() {
      struct LAPInfo *lap;

      lap = (struct LAPInfo *) NewPtr(sizeof(struct LAPInfo));
      if (lap != nil) {
             bzero((Ptr)lap, sizeof(struct LAPInfo));

             lap->lapInit = LAP_init;
             lap->lapOpen = LAP_open;
             lap->lapClose = LAP_close;
             lap->lapAttach = LAP_attachPH;
             lap->lapDetach = LAP_detachPH;
             lap->lapOutput = LAP_write;
             lap->lapControl = LAP_control;
             lap->lapFault = LAP_fault;
             lap->lapRegister = LAP_register;
             lap->lapProbe = LAP_probe;
             lap->lapConfigure = LAP_configure;
             lap->lapStatistics = LAP_statistics;
             lap->lapFindGateway = nil;
             lap->lapGwyCheck = nil;

             lap->headerSize = sizeof(struct LAP_header);
             lap->outMaxPacketSize = MAX_LAP_PKT;
             lap->inMaxPacketSize = MAX_LAP_PKT;
             lap->maxDataSize = MAX_LAP_PKT - sizeof(struct LAP_header);

             lap->lapType = ARP_LAP;
             lap->lapAddrLength = sizeof(struct LAP_addr);
             lap->lapAddrPtr = (Ptr) &lap->our_lap_addr;

             lap->lap_broadcast_addr.en_hi = 0xFFFF;
             lap->lap_broadcast_addr.en_lo = 0xFFFFFFFF;

             lap->ipGlobals = GetA5();
             lap->addressConflict = false;
             SetLAPPtr(lap);
             }
      return(lap);
      }
```

OSErr LAP_init(lap, transitionProc)

LAP_init is called by IP when the MacTCP driver is first opened. LAP_init must open the link hardware drivers, taking into account the differences between systems with slots and those without, and do any other initialization work. The disconnectProc is to be used mainly with link interfaces that may shut down over time. A serial line is an example of such an interface. See the section on serial connections for more details on how this procedure should be used. LAP_init is called at the application level.

```
OSErr LAP_init(lap, transitionProc)
struct LAPInfo *lap;
ProcPtr transitionProc;
{
     struct SpBlock sb;
     struct SlotDevParam xpb;
     OSErr  rc = -1;

     lap->ip_ph = nil;
     if (NGetTrapAddress(SlotManage,OSTrap) !=
          NGetTrapAddress(UnimplTrap,OSTrap)) {
          /* running on a system with a slot manager, determine which slot card is in */
          /* search through Nubus for the first networking card */
          bzero((char *)&sb, sizeof(sb));
          sb.spSlot = lap->lc.slot;
          sb.spCategory = catNetwork;
          sb.spCType = typeEtherNet;
          sb.spTBMask = spDrvrSWMask | spDrvrHWMask;
          rc = SNextTypeSRsrc(&sb);

          if ((lap->lc.slot != 0) && (sb.spSlot != lap->lc.slot)) {
                 /* slot of network card as set by Control Panel, isn't valid */
                 rc = smEmptySlot;
                 }
          if (rc == noErr) {
                 /* open link hardware driver */
                 rc = OpenSlot((ParmBlkPtr)&xpb, false);
                 lap->link_unit = xpb.ioRefNum;
                 }
          }
     if (rc != noErr) {
            /* running on older system, just try opening driver using its name */
            rc = PBOpen((ParmBlkPtr)&xpb, false);
            lap->link_unit = xpb.ioRefNum;
            }
     if (rc == noErr) {
            rc = PBControl();     /* get our LAP address from board */
            }
     if (rc == noErr) {
            rc = PBControl();     /* attach our packet read handler to the driver */
            }
     if (rc == noErr) {
            rc = PBControl();     /* attach our ARP packet read handler to the driver */
            }
     if (rc == noErr) {
            rc = ARP_init(lap);   /* initialize  ARP handler */
            }
     return(rc);
     }
```

MacTCP 1.0 uses MPW C 2.1 calling conventions.

The transitionProc is used to indicate to the upper layer protocol that the link protocol has undergone some change of state. The following is a description of the calling interface to the transitionProc.

```
unsigned long LapTransition(unsigned long event)
```

The following are the values that the event field can have:

0	The link level driver has been opened
1	This value is not used
2	The link level driver has closed
3	The driver is asking the upper layer protocol if it is all right to close the LAP; if a nonzero value is returned, the upper layer protocol has denied the close
4	The driver was denied a close request by another upper layer protocol

When MacTCP is called with the lap opened transition event after the driver has been called with the lap closed transition event the 'mdev' will be called in the following manner.

At SystemTask time the 'mdev' init call will be made with a nil value for the transitionProc followed by the LAP_configure call.

MacTCP 1.0 uses MPW C 2.1 calling conventions.

OSErr LAP_open(lap)

LAP_open is not currently called by the MacTCP driver.

```
OSErr LAP_open(lap)
struct LAPInfo *lap;
{
      return(noErr);
      }
```

OSErr LAP_close(lap)

LAP_close is called by IP during driver closing to close the LAP. LAP_close is called at the application level or system task time.

```
OSErr LAP_close(lap)
struct LAPInfo *lap;
{
     /* close hardware driver, etc. */
     return(noErr);
     }
```

voidProcPtr LAP_unload(lap

LAP_unload is called by IP during driver reseting to reset the LAP before it is opened

```
OSErr LAP_Unload(lap)
struct LAPInfo *lap;
{
     DisposPtr((Ptr)lap);
     return(noErr);
     }
```

OSErr LAP_attachPH(lap, ptr)

LAP_attachPH is called by IP to specify the address of the IP packet reception routine. It is assumed that the LAP knows how IP datagrams are encapsulated on the link. LAP_attachPH is only called at application level or system task time.

```
OSErr LAP_attachPH(lap, ptr)
struct LAPInfo *lap;
IPPHProc ptr;
{
     lap->ip_ph = ptr;
     return(noErr);
     }

typedef void (*IPPHProc) (struct rdStruct *rds);

struct rdStruct {
     ProcPtr   ph_rp;        /* pointer to "ReadPacket" routine (a4) */
     ProcPtr   ph_rr;        /* pointer to "ReadRest" routine 2(a4) */
     long      ph_bytesleft; /* number of bytes left to read */
     union {
     /* storage for standard AppleTalk LAP protocol handler */
             struct {
                  long      phb_a4;   /* used by driver */
                  long      phb_a0;   /* used by driver */
                  long      phb_a1;   /* used by driver */
                  long      phb_a2;   /* used by driver */
                  } phb;

               /* storage for a buffered LAP interface */
               struct {
                    b_8 *   lnb_ptr;/* pointer to next byte to get from buffer */
                    } lnb;

               /* storage for reassembled IP packet */
               struct {
                 struct fragment  *rsmb_fragbuffer;   /* pointer to fragment data buffer */
                 b_8 *            rsmb_ptr;           /* pointer to next byte to get from buffer *
                 int              rsmb_bytesleft;     /* number of bytes left in this buffer */
                 int              rsmb_byteoffset;    /* offset of next byte in rsm'd pkt */
                 } rsm;
               } rdsparm;
     Boolean          lapBroadcast;  /* LAP-level broadcast */
     Boolean          ipBroadcast;   /* IP-level broadcast */
     struct LAPInfo   *lap;          /* information about receiving LAP */
     struct wdsEntry  laphdr;        /* local net header */
     struct IPwdsEntry ip;           /* ip header */
     struct wdsEntry  tp;            /* TCP/UDP or ICMP header */
     struct wdsEntry  data;          /* TCP/UDP data */
};
```

OSErr LAP_detachPH(lap)

LAP_detachPH is called by IP during driver closing to release the binding to the IP packet reception routine. LAP_detachPH is only called at the application level or system task time.

```
OSErr LAP_detachPH(lap)
struct LAPInfo *lap;
{
      lap->ip_ph = nil;
      return(noErr);
      }
```

MacTCP 1.0 uses MPW C 2.1 calling conventions.

OSErr LAP_write(lap, dest, ipb)

LAP_write is called by IP to initiate the transmission of a packet. Packets are formatted as a Macintosh Operating System I/O parameter block followed by a write descriptor structure (WDS). IP automatically reserves the first WDS entry for the LAP header. In MacTCP 1.1, this WDS entry points to an area that is a maximum of 48 bytes long. If a larger LAP header is needed, the LAP will need to provide space from its own buffers. The LAP_write is responsible for handling all aspects of translating from an IP destination address to its local network destination address. This includes correctly interpretting the IP broadcast address (MacTCP only sends the 1's form) and handling packets addressed to the same node if the link does not do loopback. LAP_write may be called at interrupt level and cannot make synchronous system calls. LAP_write may return a negative error code upon error; otherwise it must return zero.

```
OSErr LAP_write(lap, dest, ipb)
struct LAPInfo *lap;
ip_addr dest;
struct ipbuf *ipb;
{
      struct LAP_header *en_hdr;
      struct arp_info *arp;
      struct LAP_addr *ena;
      OSErr rc;

      en_hdr = (struct LAP_header *)ipb->laphdr.ptr;
      if ((en_hdr == nil) || (ipb->laphdr.length < sizeof(struct LAP_header))) {
            /* WDS for the LAP wasn't set-up correctly */
            return(ipBadWDSErr);
            }

      ipb->laphdr.length = sizeof (struct LAP_header);

      /* mark I/O as in progress */
      ipb->iop.ioResult = inProgress;

      /* save away caller's IOC address */
      ipb->ip_ioc = ipb->iop.ioCompletion;
      ipb->iop.ioCompletion = (ProcPtr) LAP_writecomplete;

      /* format LAP link header */

      /*
      translate IP destination address into link destination address, remembering to
      handle IP broadcasts (we send only 1's) and loopback if addressed to ourselves
      */

      return(rc);
      }
```

OSErr LAP_fault(lap, lnd)

LAP_fault is called by ICMP in response to errors conditions, such as retransmission time-outs, detected at higher protocol levels such as TCP. The LAP code should do whatever diagnosis or state resetting possible to improve communication with the indicated destination. For LAPs with ARP tables, this might include clearing the ARP table entry in case it is old or has been corrupted. LAP_fault may be called at interrupt level and cannot make synchronous system calls.

```
OSErr LAP_fault(lap, lnd)
struct LAPInfo *lap;
ip_addr lnd;
{
     /* do whatever LAP-specific processing that is required, such as clearing an
        ARP table entry, etc. */
     return(noErr);
     }
```

void LAP_configure(lap)

LAP_configure is called by the MacTCP driver at system initialization time to do LAP-specific node configuration, such as reverse-ARP. LAP_configure can modify the our_ip_addr and our_net_mask fields if the configuration is successful. If the configuration fails, LAP_configure should not modify the contents of our_ip_addr and our_net_mask fields. LAP_configure is called at application level or system task time after the LAP has been initialized and can thus send and receive LAP packets.

```
OSErr LAP_configure(lap)
struct LAPInfo *lap;
{
/* do LAP specific stuff such as setting up a RARP listener and sending RARP requests */

       return;
       }
```

Boolean LAP_probe(lap, addr)

LAP_probe is called by the driver during initialization when the address is acquired “dynamically.” In the dynamic process, the driver randomly picks a node address in the range specified in the Control Panel. LAP_probe is then called to determine if that IP address is in use. LAP_probe returns TRUE if the address is not in use. The routine must determine what a suitable probing mechanism is and how long to wait for a response to any requests transmitted. After finding an available address, the driver will then call LAP_register to assert its use of the IP address. LAP_probe is called at the application level or system task time.

```
Boolean LAP_probe(lap, addr)
struct LAPInfo *lap;
ip_addr addr;
{
       int ticks;

       SendARP(lap, ARP_REQUEST, addr, &lap->lap_broadcast_addr);

       /* wait around a few milliseconds to see if anyone responds */
       ticks = TickCount() + 5;
       while ((TickCount() <= ticks) && (lap->addressConflict == false))
              /* wait around */
              ;
       return(!lap->addressConflict);
       }
```

OSErr LAP_register(lap)

LAP_register is called after initialization to assert the use of the node’s IP address. LAP_register is always called regardless of the mechanism (manual, server, dynamic) used to determine the node address.

```
Boolean LAP_register(lap)
struct LAPInfo *lap;
{
       /* verify that no-one else is using our address, lap->our_ip_addr */

       return(flag);
       }
```

OSErr LAP_findgateway(lap)

LAP_findgateway is called during initialization to perform network-specific searches for gateways. On AppleTalk, the routine would use the KIP protocol to find KIP-speaking boxes.

```
void LAP_findgateway(lap)
struct LAPInfo *lap;
{
       /* using LAP-specific means, search for a gateway on our local network. */

       /* If found, set its value in the configuration field */
       lap->lc.dfl_gwy_addr = gwyaddr;

       return;
       }
```

OSErr LAP_control(lap)

LAP_control is not currently called by the MacTCP driver.

```
OSErr LAP_control(lap)
struct LAPInfo *lap;
{
       return(noErr);
       }
```

MacTCP 1.0 uses MPW C 2.1 calling conventions.

OSErr LAP_statistics(lap)

LAP_statistics is called by the MacTCP driver to acquire link speed information and for network management. Decision on protocol timeouts and usable MTU are made with the values returned by this call.

```
OSErr LAP_statistics(lap, statsPtr)
struct LAPInfo *tempLap;
struct LAPStats *statsPtr;
{
struct ELAPInfo *lap;
#ifdef MPW3.0
#pragma unused (lap)
#endif
       lap = (struct ELAPInfo *)tempLap;
       statsPtr->ifType = 1;
       statsPtr->ifMaxMTU = MAX_ENET_PKT;
       statsPtr->ifSpeed = ETHERSPEED;
       statsPtr->ifPhyAddrLength = sizeof(struct Enet_addr);
       statsPtr->ifPhysicalAddress = (char *)&lap->our_lap_addr;
       statsPtr->AddrXlation.arp_table = &lap->arp_table[0];
       statsPtr->slotNumber = &lap->lc.slot;
       return(noErr);
       }

typedef struct LAPStats {
       short  ifType;
       char   *ifString;
       short  ifMaxMTU;
       long   ifSpeed;
       short  ifPhyAddrLength;
       char   *ifPhysicalAddress;
       union {
              struct arp_entry *arp_table;
              } AddrXlation;
       short  slotNumber;
       };

typedef struct Enet_addr {
       b_16 en_hi;
       b_32 en_lo;
       } Enet_addr;

typedef struct arp_entry {
       short         age;          /* cache aging field */
       b_16          protocol;     /* Protocol type */
       ip_addr       ip_address;   /* IP address */
       Enet_addr     en_address;   /* matching Ethernet address */
       };
```

MacTCP 1.0 uses MPW C 2.1 calling conventions.

Boolean LAP_gwycheck(lap, addr)

LAP_gwycheck is not currently called by the MacTCP driver.

```
Boolean LAP_gwycheck(lap, addr)
struct LAPInfo *lap;
ip_addr addr;
{
       return(true);
       }
```