DRAFT

Philip Budne
Shiva Corporation
February 1992

KIP AppleTalk/IP Gateway Functionality

$Revision: 1.25 $
February 26, 1992

1. Status of this Memo

This DRAFT documents the functionality of the Stanford ‘‘KIP’’ AppleTalk/IP ‘‘Gateway’’ (also called the ‘‘SEAGate code’’, ‘‘IP-Ether/AppleTalk Gateway’’, or ‘‘Croft Gateway’’). This draft document will be submitted to the RFC editor as an Informational Document. Distribution of this memo is unlimited.

> *Note:*
> *This memo was started as an effort to descibe the ‘‘IPTalk’’ protocol for the AppleTalk-IP Working Group of the Internet Engineering Task Force (IETF). It became apparent that since no protocol standard or description existed that implementation specific information was unavoidable. KIP is the prototypical implementation and is freely available in source form.*

2. Terminology

The KIP ‘‘Gateway’’ implements protocols for encapsulation of IP datagrams [RFC-791] in DDP for transmission over AppleTalk [Sidhu90] internets (IP in DDP), and encapsulation of DDP datagrams in UDP [RFC-768] for transmission over IP internets (DDP in UDP). Both have been called ‘‘KIP’’ encapsulation; IP in DDP (herein called DDP/IP) has also been called ‘‘MacIP’’, ‘‘Dartmouth encapsulation’’ and ‘‘Croft encapsulation’’ while DDP in UDP (herein called UDP/DDP) has been called ‘‘UDPTalk’’, ‘‘IPTalk’’, ‘‘AppleTalk in IP’’, ‘‘AppleTalk over UDP’’, and ‘‘Phase 1 IP tunneling’’.

The first ‘‘AppleTalk-Ethernet’’gateway created at Stanford University around 1985 and was known as the ‘‘Stanford Ethernet AppleTalk Gateway’’, or ‘‘SEAGate’’. It consisted of a Forward Technologies Multibus 68000 CPU card and a simple LocalTalk as "AppleBus") interface. The SEAGate source code and design documents were freely available to members of the Internet community. The initial

version provided simple transport of encapsulated IP datagrams from AppleTalk-based Macintosh computers to Ethernet based IP hosts. The Kinetics FastPath was created using this technology (c.1986) and consisted of a 10Mhz 68008 with 48K of battery backed SRAM, an Intel i82586 Ethernet Chip and Zilog 8530 SCC.

The SEAGate code was ported to the FastPath and continued to evolve, and was renamed ‘‘KIP’’ for the ‘‘Kinetics IP’’ Gateway. The name ‘‘KIP’’ is used in this document only in reference to the gateway software. Unless otherwise noted, all descriptions refer to the June 1988 (06/88) release of KIP. The term ‘‘Kbox’’ is used here to refer to a FastPath running KIP (or any functionally similar hardware and software).

KIP can speak both EtherTalk Phase 1 and UDP/DDP on Ethernet. This has been called ‘‘doubletalk’’.

The primary existing UDP/DDP host implementation of AppleTalk is the Columbia AppleTalk Package (CAP) for UN*X systems. All references to CAP are for Columbia Version 5.0 (using abkip.c).

The KIP sources use the historical terms ‘‘AppleBus’’ and ‘‘AppleTalk’’ in reference to the ‘‘LocalTalk’’ physical medium, as KIP pre-dates the introduction EtherTalk (and thus the use of the name AppleTalk for the protocol family rather than the medium).

All constants are in decimal unless otherwise noted.

The following ‘C’ type definitions apply to all data structures and code:

```
typedef unsigned char   u_char;   /*  8 bit ordinal */
typedef unsigned short  u_short;  /* 16 bit ordinal */
typedef unsigned long   u_long;   /* 32 bit ordinal */
typedef long iaddr_t;             /* internet address */
```

3. UDP/DDP encapsulation

This section describes the algorithms and data structures used by KIP to implement UDP/DDP encapsulation.

3.1. Motivation

UDP/DDP encapsulation was developed before Apple established a standard method for the transmission of AppleTalk over Ethernet. One of the primary strengths (and weaknesses) of UDP/DDP is that it uses static, centrally administered routing information for the UDP/DDP backbone eliminating the chatter generated by RTMP and ZIP. In addition one gains the ability to use the large infrastructure of existing IP internets to transport AppleTalk over long distances.

IP networks on which UDP/DDP routers and endnodes are homed are assigned AppleTalk network numbers. This distinguishes UDP/DDP from a strict point-to-point ‘‘tunneling’’ model in that endnodes can be implemented and it allows complete interconnectivity between routers without a quadratic number of links.

Because of the static nature of UDP/DDP routes, partial (non-transitive) network connectivity can be engineered (ie; A can see B and C, but B and C cannot see each other).

3.2. UDP/DDP network numbers

KIP and CAP configuration files usually represent AppleTalk network numbers as a dotted pair of decimal octets, thus network 258 is represented as 1.2. AppleTalk network numbers for UDP/DDP networks are chosen manually, and are arbitrary; In fact, KIP does not learn from ‘‘seed’’ routers at all, and all AppleTalk port network numbers must be manually configured. Each IP network which is to contain UDP/DDP hosts (ie; CAP clients and servers) must be assigned a UDP/DDP AppleTalk network number.

A common convention for the representation of 8 bit subnets of a class B network is to put the subnet number in the low octet and an arbitrary constant in the top octet. If the subnet is also used for EtherTalk a different high octet is used.

IP subnet	UDP/DDP net	EtherTalk net
129.2.50.0	55.50	57.50
129.2.100.0	55.100	57.100
129.2.200.0	55.200	57.200

As on any Phase 1 network, there can only be 254 hosts on a UDP/DDP network. Only the first 254 hosts (low octet equal to 1 through 254) of a network with more than 8 bits of host number are eligible to participate in UDP/DDP.

3.3. UDP/DDP Node numbers

Each CAP UDP/DDP host must be associated with one UDP/DDP network (even if the underlying IP host is multi-homed). The AppleTalk node number of a UDP/DDP host is always the low order octet of the host’s IP address on the connected UDP/DDP network.

3.4. UDP/DDP Address resolution

Since UDP/DDP encapsulation relies on the underlying IP layer for transport and address resolution, no additional Address Resolution Protocol is required.

3.5. UDP/DDP packet format

UDP/DDP packets are encapsulated in UDP with the IP destination address and UDP source and destination ports calculated based on route type.

Each UDP/DDP packet contains a dummy three byte LAP header (destination, source and type) the same as LocalTalk and EtherTalk Phase 1. This has the helpful effect of assuring alignment of the encapsulated data (otherwise the DDP header would cause odd alignment). UDP/DDP packets output by KIP always have a LAP destination of 250 and a LAP source of 206 (which spells ‘‘FACE’’ in a hex dump).

Packets originated by CAP have the source and final destination nodes numbers equal to the DDP source and destination regardless of the destination network!! KIP and CAP only send packets of LAP type 2 (long DDP).

KIP ignores the LAP header on input, and a long DDP packet must follow.

3.6. UDP/DDP Routing

The following ‘C’ data structure is used by KIP to internally represent all AppleTalk routes:

```
struct aroute {
    u_long  node;       /* next hop/IR: AT node OR IP addr */
    u_short net;        /* atalk net number, 0 if unused */
    u_char  flags;      /* flags */
    u_char  hops;       /* number hops to this net */
    u_char  zone;       /* zone index + 1 */
    u_char  age;        /* age of entry in 20 second ticks */
    u_char  port;       /* index of port route came in on */
};

/* flag values */

#define arouteTYPE    0xe0   /* up to 8 types */
#define arouteCore    0x10   /* 'node' is a "core" gateway */
#define arouteAA      0x08   /* this entry received via AA */
#define arouteSUBTYPE 0x03   /* TYPE specific information */
```

```
/* Kbox Type */
#define arouteKbox    0x80   /* 'node' is IP addr of a Kbox */
# define arouteEtalk   0x01  /* 'net' is for Kbox EtherTalk */

/* Net Type */
#define arouteNet    0x40    /* IP net allows directed cast */
# define arouteBMask  0x03   /* directed bcast format mask */

/* Host Type */
#define arouteHost   0x20    /* IP rebroadcast host */
# define arouteAsync 0x02    /* virtual async atalk net */
```

Note:
The `arouteTYPE` *field should be treated as an enumeration despite the assignment of separate bits for the existing values.*

Similarly, the `arouteSUBTYPE` *bits have not always been recognized as* `arouteTYPE` *specific.*

Finally, the `arouteCore` *flag should be associated only with ''Kbox'' routes.*

The KIP AppleTalk routing table contains three types of routes; ''Direct'', ''Local'', and ''IP''.

3.6.1. Direct

The routes for the directly connected LocalTalk and EtherTalk networks have zero `hops`, `node`, and `flags` fields.

3.6.2. Local

For routes learned via RTMP on the LocalTalk or EtherTalk interfaces, `node` is the AppleTalk node number of an AppleTalk router on the connected network, and port is the index of the associated interface. The `flags` field is zero and the `hops` field is non-zero.

3.6.3. IP

For UDP/DDP routes (destinations that will be reached by sending the AppleTalk DDP packet encapsulated in UDP), member `node` in the `aroute` struct is the IP address of another Kbox or an IP network. The flags field contains information on the type of ''IP'' route.

AppleTalk ''IP'' routes come in four flavors; ''Net'', ''Host'', ''Kbox'', and ''Async''.

3.6.3.1. Net

If the `arouteTYPE` bits of the `flags` field are equal to `arouteNet`, the AppleTalk network `net` is a UDP/DDP net coresident with an IP network that supports directed broadcasts. The `arouteBMask` bits of the `flags` field of the `aroute` struct are the number of low order octets (zero to three) of ones (hex `0xff`) to logically OR with the network number in the `node` field to produce the broadcast address. KIP has a ‘‘Net’’ route to the directly connected Ethernet/IP network by default.

3.6.3.2. Host

If the `arouteTYPE` bits of the `flags` field are equal to `arouteHost`, the appletalk network `net` is a UDP/DDP net coresident with an IP network that does not support directed broadcasts. The `node` field of the `aroute` struct is the address of a ‘‘rebroadcast host’’ which can broadcast packets on the destination network.

3.6.3.3. Kbox

If the `arouteTYPE` bits of the `flags` field are equal to `arouteKbox`, this AppleTalk network is reached by sending the DDP datagram encapsulated in UDP to the router located at address `node`. ‘‘Kbox’’ routes can refer to either ‘‘local’’ routes learned by other gateways via RTMP and distributed by the ‘‘core’’ gateway mechanism, or routes downloaded from the AppleTalk Administrator (AA) host in the initial route table. The UN*X AA server implementation distributed with KIP is `atalkad`. The `arouteEtalk` bit in the `arouteSUBTYPE` field exists as internal information used by `atalkad` to differentiate LocalTalk and EtherTalk networks for configuration purposes.

3.6.3.4. Async

‘‘Async’’ is a recent extension which allows datagrams for this AppleTalk network to be forwarded to a single IP host with the UDP port demultiplexed by host. This allows the gateway to deliver DDP datagrams for each possible host on the Async network to a different UN*X user process with no intermediary. ‘‘Async’’ networks are implemented as a subtype of ‘‘Host’’ networks, with `arouteSUBTYPE` set to `arouteAsync`. Implementations which do not understand ‘‘Async’’ routes will treat them as ‘‘Host’’ routes, with undesireable results.

3.7. IP Route Algorithms

In ‘‘Kbox’’, ‘‘Net’’, and ‘‘Host’’ routes the destination UDP port is demultiplexed based on the destination DDP socket (except for DDP broadcasts to ‘‘Host’’ networks). This allows the gateway to deliver DDP datagrams for each possible DDP socket to a different UN*X user process with no intermediary.

The following table shows how the `aroute` type and the DDP destination node select algorithms to calculate the UDP destination port and IP destination address.

IP dest addr algorithm / UDP dest port algorithm.

IP Route Type	DDP unicast	DDP broadcast
Net	Alg A/Alg P	Alg B/Alg P
Host	Alg A/Alg P	Alg C/Alg Q
Kbox	Alg C/Alg P	Alg C/Alg P
Async	Alg C/Alg R	Alg C/Alg S

The UDP source port is always calculated by applying Algorithm P to the ddp source socket.

In the following code fragments the variable `ddp` is of type `struct DDP` and describes the long DDP header of the outgoing AppleTalk packet;

```
struct DDP {
    ...
    u_short dstNet;         /* dest net */
    u_short srcNet;         /* src net */
    u_char  dstNode;        /* dest node */
    u_char  srcNode;        /* src node */
    u_char  dstSkt;         /* dest socket */
    ...
} ddp;
```

The variable `ar` is a pointer to the `aroute` struct for the DDP destination net in the AppleTalk routing table.

3.7.1. IP destination address algorithms

The following are the algorithms for calculating the IP destination host address of the UDP/DDP datagram expressed in `'C'`.

The variable `ip` is of type `struct ip` and describes the IP header of the outgoing IP datagram;

```
struct ip {
    ...
    iaddr_t ip_src;   /* IP source address */
    iaddr_t ip_dst;   /* IP dest address */
} ip;
```

3.7.1.1. Algorithm A

Algorithm A is used to deliver unicast (non-broadcast) packets on ‘‘Host’’ networks and broadcast packets on ‘‘Net’’ networks directly to the destination host on a UDP/DDP network. The IP destination is formed by taking the high 24 bits from `node`, and the low 8 bits from the DDP destination node number.

```
ip.ip_dst = (ar->node & ~0xff) | ddp.dstNode;
```

3.7.1.2. Algorithm B

Algorithm B is used to broadcast packets to ‘‘Net’’ networks, relying on directed broadcast delivery. The IP destination is formed using the `node` field and as many low order octets of ones as specified by the `arouteBMask` field of `flags`. If the destination AppleTalk network is the connected Ethernet UDP/DDP network, the configured IP broadcast address (member `ipbroad` configuration) is used as the IP destination.

```
u_long ipbroadtypes[] = { 0, 0xff, 0xffff, 0xffffff };

/* directly connected UDP/DDP net? */
if( ddp.destNet == ifie.if_dnet ) {
    /* use configured IP broadcast */
    ip.ip_dst = kipconf.ipbroad;
}
else {
    /* create directed broadcast */
    ip.ip_dst = ar->node |
                ipbroadtypes[ ar->flags & arouteBMask ];
}
```

Note:
note here regarding use of N0? submasks which are not integral octets of ones?

3.7.1.3. Algorithm C

Algorithm C is used for delivery of broadcasts on ‘‘Host’’ networks, and all packets on ‘‘Kbox’’ and ‘‘Async’’ networks to a specific host address. The IP destination is the IP host specified in the `node` (next IR) field .

```
ip.ip_dst = ar->node;
```

3.7.2. UDP destination port algorithms

The following are the algorithms for calculating the UDP destination port of the UDP/DDP datagram expressed in `'C'`, using the following structure for the output UDP header;

```
struct udp {
    u_short     uh_src;                 /* source port */
    u_short     uh_dst;                 /* destination port */
    ...
} udp;
```

3.7.2.1. Algorithm P

Algorithm P is used for delivery of all packets on ‘‘Net’’ and ‘‘Kbox’’ networks and unicast packets on ‘‘Host’’ networks. Packets for each possible DDP socket are directed to a different UDP port on the destination host.

‘‘Static’’ or ‘‘Well Known’’ DDP destination sockets (below 128) originally used a UDP port base of 768, however in April of 1988 the NIC reserved ports 201 through 208 for use by KIP [RFC-1060]. UDP ports 768 and 200 are never used, since use of DDP socket zero is illegal. See the ‘‘Discussion’’ section below.

```
#define ddpWKSBase  200   /* start of NIC WKS range */
#define oddpWKSBase 768   /* start of old WKS range */

#define ddpNWKSBase 16384 /* non WKS... */

/* "well known" (ie; static) socket number?? */
if( ddp.dstSkt < 128 ) {
    if( kipconf.startddpWKS == 0 ) {
        /* no WKS range configured
         * be backwards compatible
         */
        startddpWKS = oddpWKSBase;
    }
    else {
             /* usually ddpWKSBase (200) */
             startddpWKS = kipconf.startddpWKS;
    }
    udp.uh_dport = ddp.dstSkt + startddpWKS;
}
else /* not a WKS, use NWKS port range */
    udp.uh_dport = ddp.dstSkt - 128 + ddpNWKSBase;
```

3.7.2.2. Algorithm Q

Algorithm Q is used to deliver DDP broadcasts for ‘‘Host’’ networks to a ‘‘rebroadcast’’ server.

```
#define rebPort 902

udp.uh_dport = rebPort;
```

3.7.2.3. Algorithm R

Algorithm R is used to deliver DDP unicast packets for ‘‘Async’’ networks to the async server host based on the DDP destination node. This is done so that each dialin user running the `async` program can receive packets destined for their node without additional handling on the server host.

```
#define aaBasePort 0xAA00       /* Async AppleTalk base port */

udp.uh_dport = aaBasePort + ddp.dstNode;
```

3.7.2.4. Algorithm S

Algorithm S is used to deliver DDP Broadcasts for ‘‘Async’’ networks to a single UDP port so that the `asyncad` daemon can forward the packet to all `async` dialin users on the UN*X host.

```
#define aaBroadPort 750         /* aabroad in /etc/services */

udp.uh_dport = aaBroadPort;
```

3.7.3. Discussion

The primary benefit realized by the DDP/UDP port mapping expressed in Algorithm P is the efficient implementation of AppleTalk on systems with a system level UDP implementation and no system level AppleTalk (ie; Vanilla BSD UN*X), as system level UDP port demultiplexing is used to deliver the DDP packets directly to the user process.

Creating this one-to-one relationship between DDP sockets and UDP ports on the UDP/DDP host is possible because the size of the DDP socket space is much smaller than the UDP port space (2^8 vs. 2^16). Furthermore the UDP ports in the WKS and NWKS ranges can be used for other purposes, but only those ports can be used for UDP/DDP applications.

However it is difficult to implement a UDP/DDP AppleTalk interface on a system with both formalized UDP and DDP layers (ie; a UDP/DDP ‘‘adev’’ for a Mac using ‘‘MacTCP 1.0’’). A similar difficulty is seen implementing a UDP/DDP AppleTalk router in a user process on a system with a typical system level UDP implementation (ie; UN*X), as the router process has to listen on and send from 254 different UDP ports.

Note:
Mention Ed Moy’s work here?

3.8. KIP UDP/DDP input processing

When KIP receives UDP input on a port in either the ‘‘Well Known’’ or ‘‘Not Well Known’’ DDP socket range, it strips the UDP and pseudo LAP headers, and passes the (long) DDP packet to the regular DDP input layer.

3.9. KIP Route Aging

TBS.

3.10. CAP routing

Each CAP client keeps the IP address, DDP network and node numbers of the last host from which it received a UDP/DDP packet. This means that CAP is able to return some packets without sending them first to a local gateway (in violation of Phase 1 rules)!! If on output the DDP destination net and node both match the cached information, the packet is sent to the cached IP address, otherwise the packet is routed to a single statically configured local gateway.

3.11. CAP socket assignment

When a CAP client or server wishes to obtain a dynamic DDP socket it must search the UDP port range associated with the DDP ‘‘Not Well Known’’ socket range for a free local UDP port. Since the UDP port in both the old and new ‘‘Well Known’’ (static) ranges are below 1024, only processes executing as ‘‘super user’’ may open them.

Name Information Table (NIT) maintenance for CAP hosts is implemented in a single process, by a program named `atis` (the appletalk information server). `Atis` listens on the Name Information Socket (NIS) for `LkUp` requests and sends `LkUp-Reply` packets. CAP servers must send special register and unregister requests to `atis` to modify the NIT.

3.12. CAP Zones

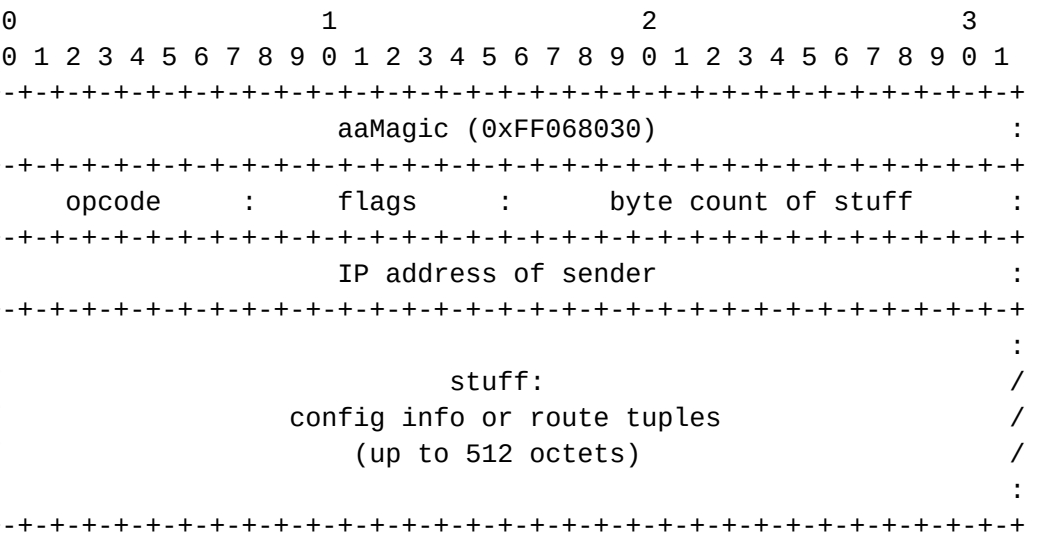
Each CAP host is staticly configured with a zone name, and `atis` will only answer NBP `LkUp`’s in its own zone. NBP `LkUp`’s sent by KIP always include the zone (never ‘‘*’’). See section on the magic `ALL` zone.

3.13. AppleTalk Administration (AA) packets

KIP is downloaded with a configuration that specifies only it’s own IP address, a default router and the address of an administrator host which will supply additional configuration information. KIP receives this configuration information and exchanges routing information on UDP port 901 (the `aaPort`). The packets used are called AppleTalk Administration, or ‘‘AA’’ packets. KIP will only accept AA packets sent from it’s configured AppleTalk Administrator host (member `ipadmin` in configuration), IP debug host (member `ipdebug` in configuration), or from any ‘‘Kbox’’ marked as from the AA in the `aroute` table.

KIP relies exclusively on the AA protocol to distribute routing information over UDP/DDP networks; RTMP packets are never exchanged. All AA packets have the

following structure:



The data portion of the packet (member `stuff`) is limited to 512 bytes to avoid IP fragmentation.

Here are the valid `opcode` values;

`aaCONF`	1	config request/reply
`aaROUTEI`	2	initial routes from AA
`aaROUTE`	3	route update
`aaROUTEQ`	4	route update and query
`aaRESTART`	5	force restart
`aaZONE`	6	initial zones from AA (obs)
`aaZONEQ`	7	DDP ZIP over AA

Two recent, commonly implemented extensions (originated by the University of Melbourne) are:

`aaROUTEM`	32	more routing info
`aaROUTER`	33	request routing info

The following are AA protocol extensions which are not widely implemented (if at all), and will not be discussed further;

A CMU extension:

```
#define aaROUTEIS 8   /* short initial route table */
```

LBL-KIP extensions (also in K-STAR 9.1);

`aaREBOOT`	`9`	force code download (via BOOTP)
`aaRESET`	`10`	reset code and config

Extensions used by Webster;

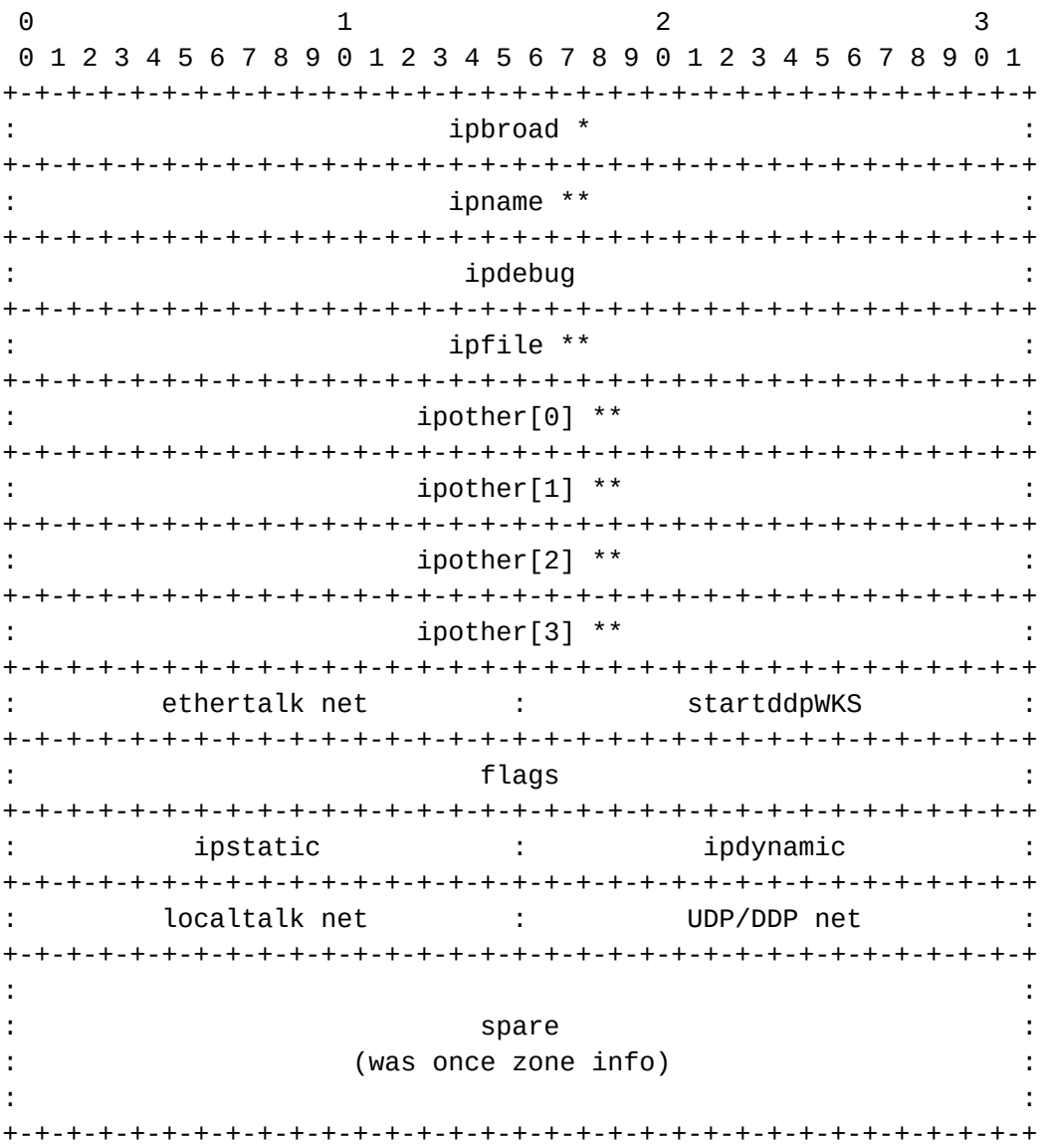
`aaPROXY`	`62`	nbp proxy arp table
`aaPROXYQ`	`63`	nbp proxy arp table request

Extensions proposed by Karen Frisa in ‘‘IPTalk routing for AppleTalk Phase 2’’ (7/27/90):

`aaROUTERANGE`	`9`	table update with ranges
`aaROUTERANGEQ`	`10`	table update and query with ranges

3.13.1. `aaCONF` packet

On startup KIP sends a minimum length (12 byte) `aaCONF` packet to the ‘‘AA’’ host as a configuration request. The AA host then replies with a full configuration `kipconf` structure in the data portion of an `aaCONF` packet;



Note:
*Entries with * are used by KIP and passed to DDP/IP clients via ‘‘IPGP’’. Entries with ** are NOT used by KIP and passed to DDP/IP clients via ‘‘IPGP’’.*

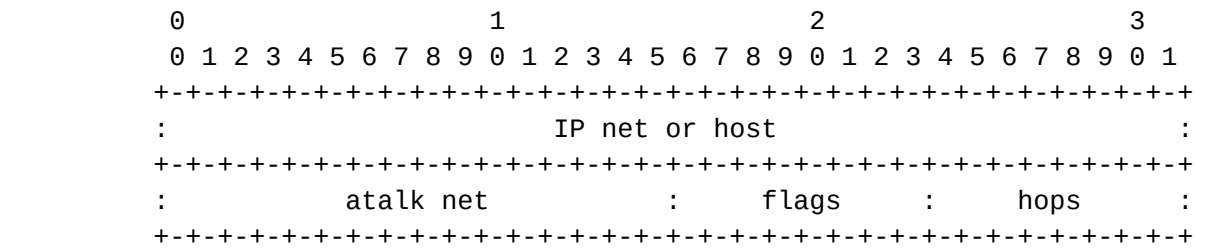
Values for `flags` (may be OR’ed together);

Flag	Value	Use
`conf_stayinzone`	1	No looking at other zones
`conf_laserfilter`	2	NBP filter LaserWriters

`conf_tildefilter` `4` Nbp filter "name~"

3.13.2. Routing Tuples

`aaROUTEI`, `aaROUTE`, `aaROUTEQ`, and `aaROUTEM` packets contain routing tuples (called `arouteTuple`'s) of the following format:



The `flags` field of the `arouteTuple` has the same flags as the `aroute` structure.

NOTE: This differs from RTMP in that it contains the explicit address of the ultimate destination (this has been called ''3rd party'' routing information).

3.13.3. `aaROUTEI` packet

The aahost sends KIP an `aaROUTEI` packet to install an initial (static) routing table. On receipt of an `aaROUTEI`, KIP flushed all routes marked as ''from AA'' (with the `arouteAA` flag set), and inserts the enclosed routes into it's routing table marked as ''from AA''.

The `atalkad route` command sends KIP an unsolicited `aaROUTEI` packet.

On startup KIP sends a minimum length (12 byte) `aaROUTEI` packet as a request every 60 seconds until a reply is received.

3.13.4. `aaROUTEQ` packet

KIP sends an `aaROUTEQ` packet once a minute to a ''core'' gateway – a ''Kbox'' with the `arouteCore` flag set. The ''core'' gateways are used round-robin. ''Core'' gateways send `aaROUTEQ` as well (KIP does not contain code to detect whether it is a ''core'' gateway or not), but a ''core''gateway never sends to itself.

The `aaROUTEQ` packet contains ''Kbox'' routes (with the gateway IP address in the `node` field) for each AppleTalk route that the gateway has learned via RTMP (those which qualify as ''Local'' routes).

On receipt of an `aaROUTEQ` KIP merges the routes into it's routing table and replies with an `aaROUTE` packet containing ALL routes which are not marked as ''from AA''.

3.13.5. `aaROUTE` packet

On receipt of an `aaROUTE` packet KIP merges the contained routes into it's routing table.

3.13.6. `aaRESTART` packet

KIP will restart upon receipt of an `aaRESTART` packet.

The `atalkad boot` command sends an `aaRESTART` packet to each gateway in `/etc/atalkatab`.

3.13.7. `aaZONE` packet

`aaZONE` is an obsolete packet format used for zone information. KIP 06/88 does not send or process `aaZONE` packets. An empty `aaZONE` packet was sent by KIP as a request, and returned with data by `atalkad`.

The format of `aaZONE` packet data is a list of (2 byte) AppleTalk network numbers terminated by a zero net number followed by a ''pascal'' string for the zone. The packet is terminated by an network number of all ones (hex `0xffff`).

3.13.8. `aaZONEQ` packet

`aaZONEQ` packets contain regular DDP ZIP `Query` and `Reply` packets.

KIP sends `aaZONEQ` packets to the the AA host for routes marked as ''from AA'' or to the IP address in the `node` field of the route for routes learned via ''core'' gateways.

3.13.9. `aaROUTER` packet

This is a recent extension implemented in Rutgers KIP (RU-KIP) and various commercial offerings. If the `flags` field of the `aaconf` packet header returned by the aa host is non-zero, it is interpreted as a count of additional initial route packets. RU-KIP sends an `aaROUTER` packet with `flags` field of the AA packet header set to one to request the first additional routing packet.

3.13.10. `aaROUTEM` packet

This is a recent extension implemented in Rutgers KIP and various commercial offerings. `Atalkad` sends the *n*-th additional initial route packet in response to the `aaROUTER` request in an `aaROUTEM` packet. RU-KIP increments it's request serial number regardless of the contents of the `flags` field in the `aaROUTEM` AA packet header.

3.14. Aroute input Processing

The `atalkad` supplied with KIP 06/88 always sends a hopcount of zero for all routes in the `aaROUTEI` packet.

Note:
more TBS. talk about errors in hop count processing!!!

3.15. AppleTalk Administrator Packet Exchanges

The following tables show AA packet exchanges involving KIP.

3.15.1. restart

This occurs when an user issues the `atalkad boot` command. `Atalkad` sends an `aaRESTART` to each ‘‘Kbox’’ in `/etc/atalkatab`.

aahost	KIP
`aaRESTART`	
	KIP reboots

3.15.2. KIP boot sequence

prose TBS here.

aahost	KIP
	`aaCONF (no data)`
`aaCONF (with data)`	

followed by either the ‘‘atalkad route sequence’’ or ‘‘atalkad route sequence with extension’’

3.15.3. atalkad route sequence

The ‘‘atalkad route sequence’’ occurs as part of the ‘‘KIP boot sequence’’ or as the result of an `atalkad route` command. The contents of the `aaconf` struct `flags` field is shown in brackets.

aahost	KIP
`aaROUTEI[0]`	
	`aaROUTEQ`

3.15.4. atalkad route sequence with extension

The ‘‘atalkad route sequence with extension’’ occurs as part of the ‘‘KIP boot sequence’’ or as the result of an `atalkad route` command when the gateway implements the `aaROUTER` and `aaROUTEM` packets and the `flags` sent with the `aaROUTEI` packet were non-zero.

The contents of the `aaconf` packet `flags` field is shown in brackets.

aahost	KIP
`aaROUTEI[n]`	
	`aaROUTER[1]`
`aaROUTEM[1]`	
	`aaROUTER[2]`
`aaROUTEM[2]`	
	`...`
`...`	
	`aaROUTER[n]`
`aaROUTEM[n]`	
	`aaROUTEQ`

3.15.5. `aaZONEQ` exchange

KIP sends ZIP `Query` and `Reply` packets encapsulated in `aaZONEQ` packets to either the AA host, or other Kboxes to acquire the zone name for networks reached via UDP/DDP encapsulation.

KIP	aahost or KIP
`aaZONEQ(ZIP Query)`	
	`aaZONEQ(ZIP Reply)`

3.15.6. Core GW exchange

KIP sends RTMP learned ‘‘Local’’ routes to Kboxes marked as ‘‘core’’ gateways in it’s routing table. The core gateway responds with all UDP/DDP routes not marked as ‘‘from AA’’.

KIP	KIP Core GW
`aaROUTEQ`	
	`aaROUTE`

3.16. Rebroadcast Port

UDP packets input on port 902 (the `rebPort`) are accepted if the destination IP address matches the gateway address (broadcasts are not accepted, nor are packets

rerouted), and passed to the ddpinput routine for processing based on the embedded DDP destination.

If the packet is a DDP broadcast, KIP will broadcast the packet using `kipconf.ipbroad` as the IP destination, while the UN*X `atalkrd` server distributed with KIP can be started with with a list of addresses to forward the packet to on it's command line.

3.17. `atalkatab`

TBS.

4. NBP Filtering

KIP performs three types of NBP based filtering; ''Stay in Zone'', ''LaserWriter'', and ''Tilde''.

4.1. Stay in Zone filtering

''Stay in Zone'' filtering (enabled by the `conf_stayinzone` flag) is the most restrictive type. If enabled, machines on the gateway LocalTalk will only be able to access resources within their own zone through KIP. Access to ANY resource outside this zone will be prevented.

''Stay in Zone filtering'' is implemented by never sending NBP `LkUp` requests (when expanding NBP `BrRq` requests) to nets which are not in the same zone as the LocalTalk port, and by sending empty replies for ZIP `GetZoneList` requests, so that no zone list appears in the Macintosh ''chooser''.

4.2. LaserWriter filtering

''LaserWriter filtering'' (enabled by the `conf_laserfilter` flag) allows free access to LaserWriters in the gateway's LocalTalk zone by all members of this zone. However machines outside this zone will be unable to see any LaserWriters behind this Kbox.

''LaserWriter filtering'' is implemented by routing NBP `LkUp-Reply` packets which contain NBP type `LaserWriter` ONLY if the source net is in the same zone as the gateway's LocalTalk or the source and destination nets are both in the gateway's LocalTalk zone.

> *Note:*
> *Absolute determination of the source and destination zones are only possible in an AppleTalk Phase 1 environment!*

4.3. Tilde filtering

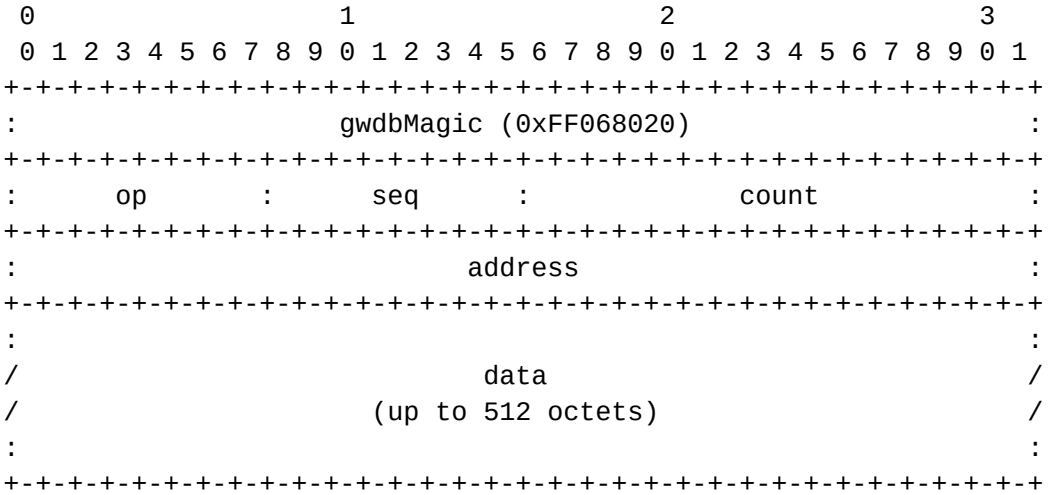
"Tilde filtering" (enabled by the `conf_tildefilter` flag) is similar to "LaserWriter Filtering". By default, all NBP names will be accessable outside the gateway LocalTalk zone. However if an NBP entity name ends in the tilde character "~" (e.g. `ourprinter~`), then this name will be unseen by any machines outside of the zone.

5. Magic `ALL` zone

The zone name `ALL` has magic properties to KIP; it allows CAP hosts on a single Ethernet to appear in disparate zones. NBP `LkUp`'s are always sent to nets in zone `ALL`, regardless of the target zone in the `BrRq` requests. KIP will never return the zone name `ALL` in a ZIP `GetZoneList` reply, so it will never be seen in a list of zones (ie; Mac "Chooser" or CAP `getzones`). Since KIP always replaces "*" with the source zone of the `BrRq` and `atis` checks the zone on incomming `LkUp`'s each CAP host will only appear in one zone.

6. Gateway Debug protocol

KIP provides for remote examine and deposit of memory locations by the configured `ipdebug` host via packets sent to UDP port 900 (the `gwdbPort`).



Op codes for gwdb;

`gwdbRead`	1	read memory
`gwdbWrite`	2	write memory
`gwdbCall`	3	not implemented

LBL KIP uses the following op codes:

```
gwdbStats   4
gwdbState   5
gwdbFrame   6
```

Shiva also uses the following op code:

```
gwdbGdb   99
```

6.1. gwdb Read function

`count` bytes starting at `address` in the gateway memory are copied to `data`, and the packet is returned.

6.2. gwdb Write function

`count` bytes of data from `data` are copied to `address` in the gateway memory and the packet is returned unmodified.

6.3. gwdb Call function

KIP does not implement this function! For unimplemented functions KIP clears the op byte and returns the packet.

6.4. gwdb State function

This function is implemented by K-STAR (and LBL KIP), and returns FastPath specific information in `data`;

```
/*
 * Structure of data area in a 'state' reply.
 */
struct gwdb_State {
    struct fp_version version;
    struct fp_state state;
};
```

6.5. gwdb Stats function

This function is implemented only by LBL KIP.

6.6. gwdb Frame function

This function is implemented by LBL KIP and K-STAR and returns the machine state of the processor; D0-D7, A0-A7, SR (left padded to 32 bits).

6.7. gwdb Gdb function

Shiva uses this function. The following remote gdb commands are supported:

Command	Function	Return value
g	return the value of the CPU registers	E*NN* / *data*
G	set the value of the CPU registers	E*NN* / OK
m*AA..AA*,*LLLL*	Read *LLLL* bytes at address *AA..AA*	E*NN* / *data*
M*AA..AA*,*LLLL*	Write *LLLL* bytes at address *AA..AA*	E*NN* / OK
c[*AA..AA*]	Continue [at address *AA..AA*]	S*NN*
s[*AA..AA*]	Step one instruction [from *AA..AA*]	S*NN*
?	What was the last signal?	S*NN*
k	kill (ignored)	

Where ‘‘*AA..AA*’’, ‘‘*LLLL*’’ and ‘‘*NN*’’ are in hex. ‘‘S*NN*’’ represents an exception encoded as a UN*X `signal` number *NN* and ‘‘E*NN*’’ represents an error condition. All returns marked ‘‘*data*’’ are streams of octets encoded in hexadecimal.

7. DDP/IP encapsulation

Note:
The Apple-IP Working Group of the IETF is working to extend and standardize the protocols for IP over AppleTalk, however no description exists of the historical implementation.

The DDP/IP encapsulation protocol consists of three parts; Encapsulation, Dynamic address assignment, and Address resolution.

7.1. Encapsulation

IP Datagrams are encapsulated in DDP packets of type 22 with DDP source and destination sockets of 72.

7.2. Dynamic Address Assignment

So that potential clients can locate the gateway, KIP responds to NBP `LkUp`’s of type `IPGATEWAY` with the dotted decimal representation of it’s address as the object name, and socket number 72.

TBS.

7.2.1. IPGP Protocol

The protocol used for dynamic address assignment is called IPGP (for the `IPGATEWAY` protocol), and is sent using ATP.

TBS.

7.2.1.1. packet format

TBS.

7.2.1.2. functions

TBS.

7.2.2. KIP address confirm behavior

TBS.

7.2.3. KIP reboot behavior

TBS.

7.3. Address Resolution

KIP uses AppleTalk NBP `LkUp`'s to achieve IP address resolution. This allows routing of IP packets within an AppleTalk zone, without requiring any changes to intermediate DDP routers.

Each DDP/IP host must NBP register a tuple of type `IPADDRESS` with the object name set to the dotted decimal representation of the host's IP address. This ensures that only one host is using an IP address at a time, and allows hosts (both Macintoshes and KIP) to locate IP hosts by address using NBP.

KIP BEHAVIOR TBS HERE.

Another method was previously used for DDP/IP IP Address Resolution. Work done in the summer of 1984 at Dartmouth [Sherman86] treated the LocalTalk as a distinct IP network, and sent ARP [RFC-826] using DDP type 23, and used hardware type 3 for ''AppleBus''.

This method was also used by early Kinetics software, and limits the IP (sub)net to a single LocalTalk wire. IP addresses on the LocalTalk (sub)net can either be assigned staticly, or LAP node number can be used for the IP address host part to provide a dynamic IP address!

7.3.1. NBP Proxy ARP

TBS.

8. Stanford KIP revision history

Below is an edited revision history for KIP.

8.1. 10/86

Improved IP address management (IPGP + NBP Arp). Centralized configuration and boot control. Full routing using NBP/RTMP/ZIP; ‘‘core’’ gateway scheme. Gateway debugging via net ddt. Simple integration with libraries such as CAP/KHOST. Improved packet throughput. *(Croft)*

8.2. 02/87

Bug fixes. *(Croft)*

8.3. 09/87

Implement zones (`aaZONE`) and zone filtering. *(Croft and Kim)*

8.4. 01/88

Ethertalk support. Magic zone name `ALL`. ATP ZIP support rewritten. DDP Echo support. atalkad configuration aids. *(Kim and Tappan)*

8.5. 06/88

Support for NIC UDP port range. Zone acquisitions are done on RTMP routes. EtherTalk fixes. Allzones was on for ‘‘unknown’’ zones. *(Kim)*

9. NIC Assigned UDP/DDP ports:

The following UDP ports were assigned for use by KIP in April of 1988 [RFC-1060].

Port	NIC Name	Use
201	`AT-RMTP`	AppleTalk Routing Maintenance
202	`AT-NBP`	AppleTalk Name Binding
203	`AT-3`	AppleTalk Unused
204	`AT-ECHO`	AppleTalk Echo
205	`AT-5`	AppleTalk Unused
206	`AT-ZIS`	AppleTalk Zone Information
207	`AT-7`	AppleTalk Unused
208	`AT-8`	AppleTalk Unused

In practice RTMP is never sent in UDP/DDP packets. The NBP and ECHO are ports are served by `atis`. The CAP `getzones`command ZIP `GetZoneList` packets to

KIP, but KIP always sends ZIP `Query` commands in `aaZONEQ` packets.

10. Programmer's Reference:

The following are the `'C'` data structure declarations used by KIP for packets described in this memo;

NOTE: All data is in ''network'' (native 68000) byte order.

```
/*
 * AppleTalk administration packets from AppleTalk
 * administrator host (AA) or other gateways;
 * configuration  / routing information packet.
 */

struct aaconf {
    u_long  magic;     /* magic number aaMagic */
    u_char  type;      /* op code */
    u_char  flags;
    u_short count;     /* byte count of 'stuff' */
    iaddr_t ipaddr     /* IP address of sender */
    u_char  stuff[512]; /* config info or route tuples */
};

#define aaconfMinSize 12
#define aaPort    901   /* udp port number */
#define aaMagic   ((u_long)0xFF068030)
```

Routing tuple contained in all `aaROUTE*` packets;

```
/*
 * Routing tuple.
 */
struct arouteTuple {
    long    node;           /* IP net or host address */
    u_short net;            /* atalk net number */
    u_char  flags;          /* flags, see aroute */
    u_char  hops;           /* hop count */
};
```

Configuration information from `aaROUTEI` packet;

```
struct kipconf {
    iaddr_t ipbroad;         /* IP broadcast addr */
    iaddr_t ipname;          /* address of name server */
    iaddr_t ipdebug;         /* address of debug host */
    iaddr_t ipfile;          /* address of file server */
    u_long  ipother[4];      /* other addresses for IPGP */
    u_short anetet;          /* EtherTalk AT net # */
    u_short startddpWKS;     /* UDP WKS range start */
    u_long  flags;           /* various bit flags */
#define conf_stayinzone  0x1 /* no looking at other zones */
#define conf_laserfilter 0x2 /* NBP filter LaserWriters */
#define conf_tildefilter 0x4 /* NBP filter "name~" */
    u_short ipstatic;        /* static IP addrs */
    u_short ipdynamic;       /* dynamic IP addrs */
    u_short atneta;          /* LocalTalk AT net # */
    u_short atnete;          /* UDP/DDP AT net # */
    u_char  spare[16];       /* was once zone info */
};
```

Gateway debug protocol (via ddt68 on UN*X);

```
struct gwdb {
    u_long  magic;          /* magic number gwdbMagic */
    u_char  op,seq;         /* op code, sequence number */
    u_short count;          /* byte count */
    u_long  address;        /* address of read/write */
    u_char  data[512];
};
```

11. Author's Note:

Portions of this document were copied from KIP source files covered by the following copyrights:

© 1984, Stanford Univ. SUMEX project.
May be used but not sold without permission.

© 1986, Kinetics, Inc.
May be used but not sold without permission.

© 1986, Stanford, BBN, Kinetics.

May be used but not sold without permission.

© 1986, Stanford Univ. CSLI.
May be used but not sold without permission.

© 1986, Stanford Univ. SUMEX project.
May be used but not sold without permission.

The following are trademarks;

AppleTalk, EtherTalk, LaserWriter, and LocalTalk are trademarks of Apple Computer, Inc.

FastPath is a registered trademark of Novell licenced for use exclusively by Shiva Corporation.

K-STAR is a trademark of Novell licenced for use exclusively by Shiva Corporation.

Kinetics is a registered trademark of Novell.

Mac and Macintosh are registered trademarks of Apple Computer, Inc.

UN*X is not a trademark of anyone.

12. References:

[RFC-768]
Postel, J.B., ‘‘User Datagram Protocol’’, Information Sciences Institute, University of Southern California, August 1980.

[RFC-791]
Postel, J.B., ‘‘Internet Protocol’’, Information Sciences Institute, University of Southern California, September 1981.

[RFC-826]
Plummer, D.C. ‘‘Ethernet Address Resolution Protocol’’, November 1982.

[RFC-1060]
Reynolds, J., and J. Postel, ‘‘Assigned Numbers’’, Information Sciences Institute, University of Southern California, March 1990.

[Sherman86]
Sherman, M. ‘‘A Network Package for the Macintosh using the DoD Internet

Protocols'', Technical Report PCS-TR86-124, Computer Network Laboratory, Department of Mathematics and Computer Science, Dartmouth College.

[Sidhu90]
Sidhu, G., R. Andrews and A. Oppenheimer, ''Inside AppleTalk, Second Edition'', Apple Computer, Inc. May 1990.

13. Acknowledgments:

The author is indebted to the following for their help, input and documentation; Scot Drysdale of Dartmouth University, Robert Elz of the University of Melbourne, Tom Evans of Webster Computer, David Hornsby of the University of Melbourne, Philip Koch of Dartmouth University (now at Apple Computer), Josh Littlefield of Cayman, Brad Parker of Cayman, Mark Sherman of Transarc (formerly at Dartmouth), Michael Swan of Neon Software (formerly of Kinetics).

And of course Bill Croft, Dan Tappan and Charlie Kim.

14. Author's Address:

Philip Budne
Shiva Corporation
1 Cambridge Center
Cambridge, MA 02142

Phone: 617-252-6300
EMail: phil@Shiva.COM