Network Working Group **Philip Budne**
DRAFT **Shiva Corporation**
July 1991

KIP AppleTalk/IP Gateway Functionality

$Revision: 1.10 $

July 26, 1991

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 encapsulations have been called ‘‘KIP’’ encapsulation; IP in DDP (herein called DDP/IP) has also been called ‘‘MacIP’’ 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 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 typical configuration for running KIP was a Kinetics FastPath 1, 2 or 3 (a 10Mhz 68008 with 48K of RAM). The term ‘‘Kbox’’ is used to refer to a FastPath running KIP.

KIP can speak both EtherTalk Phase 1 and UDP/DDP on Ethernet. This has been called ‘‘doubletalk’’. The ability to speak EtherTalk Phase 2 as well has been called ‘‘tripletalk’’.

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.

3. Data types

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 */
```

4. UDP/DDP encapsulation

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

4.1. 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 ‘‘tunneling’’ model in that endnodes can be easily implemented and it allows complete interconnectivity between routers without the quadratic number of links required using explicit point to point 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)

4.2. UDP/DDP network numbers

KIP and CAP 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. If there is only one gateway on the Ethernet, the gateway LocalTalk uses another arbitrary constant in the top octet.

IP subnet	UDP/DDP net	EtherTalk net	GW LocalTalk net
129.2.50.0	55.50	57.50	56.50
129.2.100.0	55.100	57.100	56.100
129.2.200.0	55.200	57.200	56.200

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.

Note:

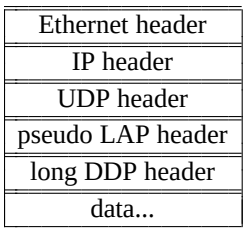
It has been suggested that ‘‘Phase 2’’ UDP/DDP could be implemented allowing larger IP (sub)nets to appear as AppleTalk network ranges.

4.3. 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.

4.4. UDP/DDP Packet Format

Here is how a UDP/DDP datagram appears when transmitted on Ethernet;



4.5. pseudo LAP header

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 long and short DDP headers 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). The LAP header is ignored on KIP UDP/DDP input, and a long DDP packet must follow.

4.6. KIP Routing Table

The following 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;       /* 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 SUBTYPE bits have not always been recognized as as TYPE specific.

Finally, the ''Core'' flag should be associated only with Kbox routes.

The KIP AppleTalk routing table contains 3 types of routes:

4.6.1. direct

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

4.6.2. local

Member `node` in the `aroute` struct is the AppleTalk node number of an AppleTalk router accessible on either the directly connected LocalTalk or EtherTalk interface (learned via RTMP). The `flags` field is zero and the `hops` field is non-zero.

4.6.3. IP

Member `node` in the `aroute` struct is an IP address of another kbox or an IP network.

AppleTalk ''IP'' routes come in 4 flavors:

4.6.3.1. Net

This AppleTalk network is coresident with an IP network with a path that supports directed broadcasts. The `arouteBMask` bits of the `flags` field of the `aroute` struct are the number of low order bytes (zero to three) of ones (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.

4.6.3.2. Host

This AppleTalk network is 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.

4.6.3.3. Kbox

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.

4.6.3.4. 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.

4.7. 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 arouteTYPE and 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	A/P	B/P
Host	A/P	C/Q
Kbox	C/P	C/P
Async	C/R	C/S

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

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 of type `struct aroute *` and points to the AppleTalk routing table entry for the DDP destination net.

4.7.1. IP destination address algorithms

The following are the algorithms for calculating the IP destination host address of the UDP/DDP datagram expressed `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;
```

4.7.1.1. Algorithm A

Used to deliver unicast (non-broadcast) Host and broadcast Net packets directly to the UDP/DDP destination host on a UDP/DDP network. The high 24 bits of the IP address are taken from `node` and ddp destination node number determines the low 8 bits.

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

4.7.1.2. Algorithm B

Used to broadcast packets to ‘‘Net’’ type networks, relying on directed broadcast delivery. The IP destination is formed using the `node` field and as many low order bytes of ones as specified by the arouteBMask field of `flags`. If the destination AppleTalk network is the connected Ethernet UDP/DDP network, `kipconf.ipbroad` (the configured IP broadcast address) 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 submasks which are not integral octets of ones?

4.7.1.3. Algorithm C

Used to deliver Host broadcast, and all Kbox and Async packets to the IP host specified in the `node` (next IR) field .

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

4.7.2. 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;
```

4.7.2.1. Algorithm P

Used to deliver 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. ‘‘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]. See discussion below. UDP ports 768 and 200 are never used, since a DDP socket of zero is illegal.

```
#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
    udp.uh_dport = ddp.dstSkt - 128 + ddpNWKSBase;
```

4.7.2.2. Algorithm Q

Deliver DDP broadcasts for ‘‘Host’’ networks to a rebroadcast host. KIP will broadcast the packet using `kipconf.ipbroad`, while the Un*x `atalkrd` server distributed with KIP can be configured with a list of addresses to forward the packet to.

```
#define rebPort 902

udp.uh_dport = rebPort;
```

4.7.2.3. Algorithm R

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;
```

4.7.2.4. Algorithm S

DDP Broadcasts for ‘‘Async’’ networks are delivered to a single UDP port so that the `asyncad` daemon can rebroadcast the packet to all async dialin users on the Un*x host.

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

udp.uh_dport = aaBroadPort;
```

4.7.3. Discussion

The primary benefit realized by the DDP/UDP port mapping expressed in Algorithm A 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’’). 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 process router has to listen on and send from 254 different UDP ports.

Note:
Mention Ed Moy’s work here?

4.8. 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.

4.9. CAP routing

Each CAP client keeps the IP address, DDP network and node numbers of the last host from which it received a packet. This means that CAP is able to return some packets without sending them first to a local gateway. 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.

4.10. 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 by a single process called `atis`. `Atis` listens on the Name Information Socket (NIS) for lookup requests and replies to them. CAP servers must send special register and unregister requests to atis to modify the NIT.

4.11. CAP Zones

Each CAP host is staticly configured with a Zone name, and `atis` will only answer NBP lookups in its own zone. NBP Lookups from the KIP always include the zone (e.g. never ‘‘*’’). See section on the magic ‘‘ALL’’ Zone.

4.12. 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 AppleTalk administrator host (ipadmin), IP debug host (ipdebug) or from the IP address of any Kbox aroute marked as from the AA in the aroute table. The Un*x AA server implementation distributed with KIP is `atalkad`.

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:

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)

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

Two recent, commonly implemented extensions are:

```
#define aaROUTEM  32  /* more routing info */
#define aaROUTER  33  /* request routing info */
```

The following are aa protocol extensions which are not widely implemented, and will not be discussed further;

CMU extension:

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

An LBL-KIP extension?

```
#define aaRELOAD  9   /* reload code using BOOTP */
```

Extensions used by Webster;

```
#define aaPROXY   62  /* nbp proxy arp table */
#define aaPROXYQ  63  /* nbp proxy arp table request */
```

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

```
#define aaROUTERANGE  9  /* table update with ranges */
#define aaROUTERANGEQ 10 /* table update and query with ranges */
```

4.12.1. `aaCONF` packet

On startup KIP sends a minimum length `aaCONF` packet to the ‘‘AA’’ host as a configuration request, the aahost then replies with a full configuration in the data portion of an `aaCONF` 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 */
};
```

4.12.2. Routing Tuples

The following AA packet types contain routing tuples: `aaROUTEI`, `aaROUTE`, `aaROUTEQ`, and `aaROUTEM` of the following format:

```
/*
 * 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 */
};
```

Note:
This differs from RTMP info in that it contains the explicit address of the ultimate destination.

4.12.3. `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 its routing table marked as "from AA".

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

On startup KIP sends an `aaROUTEI` request every 60 seconds until a reply is received.

4.12.4. `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 core 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 the gateway has learned from RTMP and from ‘‘core’’ gateways.

On receipt of an `aaROUTEQ` KIP merges the routes into it’s routing table and replies with an `aaROUTE` packet.

Like `aaROUTEQ` The `aaROUTE` packet contains Kbox routes (with the gateway IP address in the `node` field) for each AppleTalk route the gateway has learned from RTMP and and from ‘‘core’’ gateways.

4.12.5. `aaROUTE` packet

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

4.12.6. `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`.

4.12.7. `aaZONE` packet

`aaZONE` is an obsolete packet format used for zone information. KIP 06/88 does not support `aaZONE`. An empty `aaZONE` packet was sent by KIP as a request and returned by `atalkad`.

The format of `aaZONE` packets 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 (0xffff).

4.12.8. `aaZONEQ` packet

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

KIP sends `aaZONEQ` packets for nets which it has learned from the AA host or via ‘‘core’’ gateways.

4.12.9. `aaROUTER` packet

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

4.12.10. `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. RU-KIP increments it’s request serial number regardless of the contents of the `flags` field of the `aaROUTEM` packet `aaconf` struct header.

4.13. 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!!!

4.14. AppleTalk Administrator Packet Exchanges

The following tables represent AA packet exchanges performed by KIP.

4.14.1. atalkad boot command

This occurs when an user issues the `atalkad boot` command.

aahost	KIP
aaRESTART	
	KIP reboots

4.14.2. KIP boot sequence

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

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

4.14.3. 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]	
	no reply
aaZONE	
	no reply

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

aahost	KIP
aaROUTEI[n]	
	aaROUTER[1]
aaROUTEM[1]	
	aaROUTER[2]
aaROUTEM[2]	
	...
...	
	aaROUTER[n]
aaROUTEM[n]	
	no reply
aaZONE	

4.14.5. `aaZONEQ` exchange

Prose TBS.

KIP	aahost/KIP
aaZONEQ(ZIP Query)	
	aaZONEQ(ZIP Reply)

4.14.6. Core GW exchange

Prose TBS.

KIP	KIP Core GW
aaROUTEQ	
	aaROUTE

4.15. Rebroadcast Port

UDP packets input on the port 902 (the `rebPort`) are accepted only if the destination IP address matches the Gateway’s address (broadcasts are not accepted, nor are packets rerouted) and passed to the ddpinput routine for delivery based on the embedded DDP destination.

4.16. `atalkatab`

TBS.

5. NBP Filtering

KIP performs 3 types of NBP based filtering;

5.1. Stay in Zone filtering

''Stay in Zone filtering'' (enabled by the `conf_stayinzone` flag in `kipconf.flags`) 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'' ''Lookup'' requests (when expanding NBP ''Bridge'' requests) to nets which are not in the same zone as the LocalTalk port, and by sending empty replies for ZIP ''Get Zone List'' requests.

5.2. LaserWriter filtering

''LaserWriter filtering'' (enabled by the `conf_laserfilter` flag) allows free access to 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 of the LaserWriter's on this Kbox.

''LaserWriter filtering'' is implemented by only routing NBP replies which contain NBP type ''LaserWriter'' if the source net is not in the same Zone as the gateway's LocalTalk or the source and destination nets are both in the gateway's LocalTalk Zone.

5.3. 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.

6. Magic ''ALL'' Zone

The Zone name ''ALL'' has magic properties to KIP; it allows CAP hosts on a single ethernet to appear in different Zones. NBP lookups are always sent to nets in Zone ''ALL'', regardless of the target zone in the ''Bridge'' requests. KIP will never return the Zone name ''ALL'' in a ZIP ''Get Zone List'' reply, so it will never be seen in a list of Zones (ie; Mac ''Chooser'' or CAP `getzones`).

Zone ''ALL'' is usually used only for UDP/DDP nets (ie; `atalkatab` ''H'' or ''N'' lines).

7. Gateway Debug protocol

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

```
/*
 * 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];
};

#define gwdbMagic       ((u_long)0xFF068020)
#define gwdbPort        900     /* udp port number */

/* op codes */
#define gwdbRead        1
#define gwdbWrite       2
#define gwdbCall        3
```

LBL KIP and K-STAR use the following op codes:

```
#define gwdbStats       4
#define gwdbState       5
#define gwdbFrame       6
```

Shiva also uses the following op code:

```
#define gwdbGdb         99      /* Shiva extension */
```

7.1. gwdb Read function

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

7.2. gwdb Write function

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

7.3. gwdb Call function

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

7.4. gwdb Stats function

This function is implemented only by LBL KIP?

7.5. gwdb State function

This function is implemented by K-STAR (and LBL KIP?), and returns FastPath specific data structures;

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

7.6. gwdb Stats function

This function is implemented only by LBL KIP?

7.7. gwdb Frame function

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

7.8. 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	data / E*NN*
G	set the value of the CPU registers	OK / E*NN*
m*AA..AA*,*LLLL*	Read *LLLL* bytes at address *AA..AA*	data / E*NN*
M*AA..AA*,*LLLL*	Write *LLLL* bytes at address *AA..AA*	OK / E*NN*
c[*AA..AA*]	Continue [at address *AA..AA*]	S*NN*
s	Step one instruction	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 Un*x `signal` number *NN* and ''E*NN*'' represents an error condition. All returns marked ''data'' are streams of hex.

8. DDP/IP encapsulation

DDP/IP encapsulation consists of 3 services; Dynamic address assignment, Address resolution, and the actual DDP/IP encapsulation.

8.1. Encapsulation

The IP Datagram is encapsulated in DDP packet using type 22.

8.2. Dynamic Address Assignment

TBS.

8.2.1. IPGP Protocol

TBS.

8.2.1.1. packet format

TBS.

8.2.1.2. functions

TBS.

Note:
reference new apple-ip IETF work here.

8.2.2. KIP address confirm behavior

TBS.

8.2.3. KIP reboot behavior

TBS.

8.3. Address Resolution

KIP uses AppleTalk NBP lookups to achieve IP address resolution. This permits routing of IP packets within an AppleTalk zone, without requiring any changes to intermediate DDP routers.

Each IP host (ie; Macintosh) must NBP register a tuple of type ‘‘IPADDRESS’’ and with the object name set to the dotted decimal representation of the host’s IP address.

KIP BEHAVIOR TBS HERE.

Another method was previously used for DDP/IP IP Address Resolution. Work done in the summer of 1984 at Dartmouth by Mark Sherman [Sherman86] treated the LocalTalk as an IP (sub)net, and used ARP [RFC-826] over LocalTalk using DDP type 23. This method was also used by early Kinetics software. This approach limits the IP (subnet) to a single LocalTalk wire, and required that all hosts be statically configured.

Note:
An alternative which never appears to have been used would be to to use the LAP node number for the IP address host part to provide a dynamic IP address!

9. Stanford KIP revision history

Below is an edited revision history for KIP;

9.1. 10/86

Improved IP address management. Centralized configuration & 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)

9.2. 02/87

Bug fixes. *(Croft)*

9.3. 09/87

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

9.4. 01/88

Ethertalk support. Magic zone name ‘‘ALL’’. ATP ZIP support rewritten. DDP Echo support. atalkad configuration aids. *(Kim*&

9.5. 06/88

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

10. NIC Assigned UDP/DDP ports:

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

Port	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, only NBP and ECHO (both opened by atis) are used by CAP or KIP, as RTMP is never sent and ZIP is sent encapsulated in `aaZONEQ` packets.

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;

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

Kinetics is a registered trademark of Novell.

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

LocalTalk is a trademark 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]
‘‘A Network Package for the Macintosh use the DoD Internet Protocols’’, Sherman, M. Computer Network Laboratory, Department of Mathematics and Computer Science, Dartmouth College, Hanover, NH Technical Report PCS-TR86-124

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

13. Author’s Address:

Philip Budne
Shiva Corporation
1 Cambridge Center
Cambridge, MA 02142

Phone: 617-252-6300

EMail: phil@Shiva.COM