

CS 425 Program 3: Routing

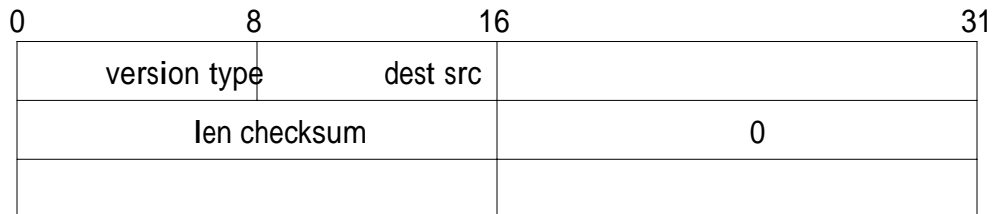
Due: April 29, 2007, 11:59pm

Introduction

For this assignment you will implement a router that routes packets using the Arizona Internet Protocol (AIP). AIP packets are encapsulated in UDP datagrams, but a router can only exchange AIP packets to those routers to which it is directly connected (its *neighbors*). A configuration file for each router specifies the links to its neighbors and their costs. The routers use the Arizona Distance Vector Protocol (ADVP) to exchange routing information and construct their forwarding tables.

AIP

The Arizona Internet Protocol is a simple protocol that connects nodes using UDP datagrams. Each AIP packet must fit in a single datagram and has the following header:



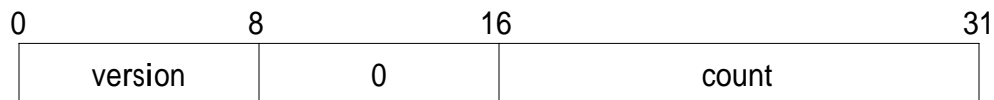
This header is defined in `aip.h`. All fields are in network byte-order. The *version* field indicates the version of the AIP protocol (currently 1). The *type* field indicates the type of packet, either `AIP_TYPE_DATAGRAM` or `AIP_TYPE_ADVP`. The former value indicates that it is a standard AIP datagram packet that should be forwarded to its destination, the latter indicates that it is an ADVP packet used to create the forwarding tables on the router. More on these in the following section. The *dest* field contains the AIP address of the destination. Each AIP node has a unique AIP address. The *src* field contains the AIP address of the source. The *len* field is the length of the entire AIP packet (including the header) in bytes. The *checksum* field contains the Internet checksum (pg. 91 of the textbook) of the entire AIP packet including the header. The last field must be zero and simply makes the header a whole number of 32-bit words.

AIP Datagrams

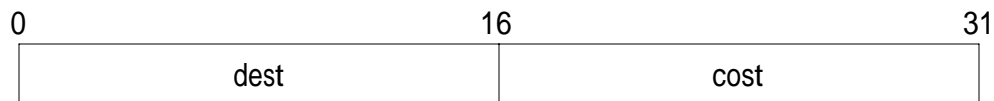
AIP datagrams are forwarded by the AIP routers to their destinations. If the destination is a router itself, the destination router simply writes the body of the datagram to stdout. If the destination is not found in a router's forwarding table the router prints an error message and drops the packet. The router also prints an error message and drops the packet any errors are found in the header (e.g. the checksum is incorrect).

ADVP

The Arizona Distance Vector Protocol is used by the AIP routers to construct their forwarding tables. Each AIP routers sends ADVP packets to its neighbors indicating the cost of reaching various AIP addresses through it. An ADVP packet begins with an AIP header (with type AIP_TYPE_ADVP), followed by an ADVP header:



The ADVP header is defined in `advp.h`. All fields are in network byte-order. The *version* field indicates the version of the ADVP protocol (currently 1), and the *count* field is the number of ADVP entries following the header. Each entry has the format:



The fields of an ADVP entry are also in network byte-order. The *dest* and *cost* fields are the AIP address of the destination that can be reached through the AIP router sending this ADVP packet and the cost of doing so.

Each AIP router sends an ADVP packet to each neighbor router every 10 seconds as well as every time its forwarding table changes. If a router hasn't received an ADVP packet from a neighbor in at least 30 seconds it declares the neighbor dead. As a result, the cost of any route through that neighbor are is set to infinity (`ADVP_INFINITE_COST` defined in `advp.h`) so that packets are no longer routed to the neighbor. When the neighbor eventually recovers it will send an ADVP packet to the router, causing it to recalculate the cost of reaching various AIP addresses through the neighbor and perhaps start forwarding packets to it again.

Router Configuration

A router is configured via a configuration file such as the following:

```
version = 1; // current config version is 1
port = 0; // listen on port 0 (let OS choose)
aip = 1; // this router's AIP address

links { // links to neighbors
  link0 {
    aip = 2; // neighbor's AIP address
    cost = 100; // cost of link to neighbor
  }
  link1 {
    aip = 3;
    cost = 200;
  }
}
```

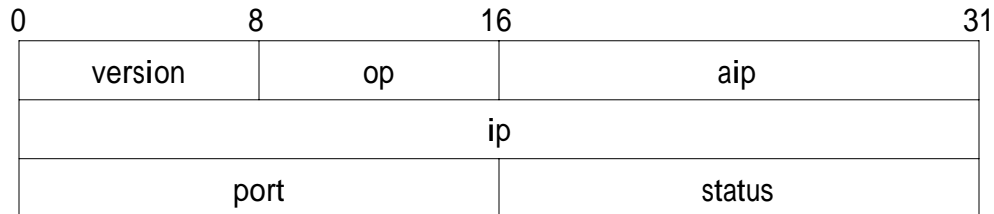
The files `config.h` and `config.c` provide routines for processing the configuration file, based on the `libconfig` library. Note that the IP address and port of a neighbor are not specified. Doing so would require you to specify the port numbers in advance which would be problematic because you can't guarantee that some of the ports aren't already in use. Letting the OS choose doesn't work either. The problem is solved using Arizona ARP (AARP) as described in the next section.

AARP

AARP is used to find the IP address and port for a given AIP address. The real ARP protocol is based on broadcast; that's not feasible for this project so instead AARP is based on a server. The AARP server (which we provide) maps AIP addresses to IP addresses and ports. After a router reads its configuration file it contacts the AARP server to set its AIP address, IP address, and port. When a router needs to know the IP address and port of one of its neighbors it contacts the AARP server and provides the AIP address and receives the IP address and port. The AARP server may not know the answer because the other router hasn't started yet. In this case it will return a "not found" error and the requesting router should treat its neighbor as down.

Note that because AIP is based on IP the routers are actually fully-connected so that any router could use AARP to find the IP address and port of any other router, and then communicate directly. This is not allowed. Routers may only communicate with their directly-connected neighbors.

An AARP packet is encapsulated in a UDP datagram and has the following format (it does not have an AIP header):



The packet is defined in `aarp.h`. The fields are in network byte-order. The *version* field is the version of the AARP protocol (currently 1). The *op* field is one of `AARP_OP_SET`, `AARP_OP_GET`, or `AARP_OP_REPLY`. The *ip* and *port* fields contains the IP address and port associated with the AIP address. If the *op* field is `AARP_OP_REPLY` then the *status* field is one of `AARP_STATUS_SUCCESS`, `AARP_STATUS_FAILURE`, or `AARP_STATUS_NOTFOUND`; otherwise it is 0.

An AIP router uses `AARP_OP_SET` to set its IP address and port. The *ip* field is ignored by the AARP server and the IP address is obtained from the underlying UDP datagram (it isn't easy for an AIP router to get its IP address since a single computer can have multiple IP addresses). The *port* field is used, and should be set by the router using `getsockname()`. The AARP server will respond with a reply that has the *op* field set to `AARP_OP_REPLY` and the *status* field set accordingly.

An AIP router uses `AARP_OP_GET` to get the AIP address of another router. The AARP server will respond with a reply that has the router's IP address and port set if the *status* is `AARP_OP_SUCCESS`. `AARP_STATUS_NOTFOUND` will be returned if the AARP server simply doesn't have an entry for the requested AIP address; `AARP_STATUS_FAILURE` is returned if an error occurred.

AIP Sender

We've provided you with a simple application called `aip_send` that reads data from `stdin` and sends it to the specified AIP address using AIP datagrams. This will help you test your AIP router.

Command Syntax

```
aip_router [-a name:port] [-c configFile] [-v]
```

-a specifies the host name and port of the AARP server.

-c specifies the configuration file.

-v produces verbose output on stderr. Your AIP router should produce the same output as the provided one.

Your router should print a message to stderr whenever it forwards a packet or determines a neighbor is up or down. It should also print additional messages if the -v option is specified. Use the provided aip_router for guidance on which messages to print and when.

```
aarpd [-p port] [-v]
```

-p specifies the port on which the AARP server (daemon) listens. If this option is not specified the server will let the OS choose a port.

-v produces verbose output.

```
aip_send [-a name:port] [-d dest] [-s src] [-r router] [-v]
```

-a specifies the host name and port of the AARP server.

-d specifies the destination AIP address.

-s specifies the source AIP address.

-r specifies the AIP router to which aip_send should connect.

-v produces verbose output.

Implementation Notes

You are only responsible for implementing aip_router. The aip_send and aarpd programs are provided for you.

Do not modify any of the header files we provide. Use additional header files as necessary.

Use SIGALRM to implement the periodic ADVP packets. Your code should block this signal except

when `recvfrom()` is called to prevent race conditions.

You can implement `aip_router` without using `malloc()`. Constants are provided in the various header files to allow you to allocate data structures statically.

Makefile

You are to provide a working Makefile that builds `aip_router` simply by typing “make”. Typing "make clean" should remove any intermediate object files. You are welcome to have any additional targets you find useful.

Testing

We will make binaries `aip_router`, `aip_send`, and `arpd` available so you can test your programs. We will also provide the source code for `arpd` since it isn't integral to the assignment and having the source might be useful. The compilation and testing of the program is to be done on the fdXX(01-08) machines.

Turnin

For the turnin, you will need to submit all the files that make up your `aip_router`. You may, if helpful to us in grading, turn in a README file to help us understand your code. Use the "turnin" program on `lectura` to turn in your assignment under the name “`cs425prog3`”.