
NOTE: See also arcnet-hardware.txt in this directory for jumper-setting and cabling information if you're like many of us and didn't happen to get a manual with your ARCnet card.

Since no one seems to listen to me otherwise, perhaps a poem will get your attention:

This driver's getting fat and beefy,
But my cat is still named Fifi.

Hmm, I think I'm allowed to call that a poem, even though it's only two lines. Hey, I'm in Computer Science, not English. Give me a break.

The point is: I REALLY REALLY REALLY REALLY REALLY want to hear from you if you test this and get it working. Or if you don't. Or anything.

ARCnet 0.32 ALPHA first made it into the Linux kernel 1.1.80 - this was nice, but after that even FEWER people started writing to me because they didn't even have to install the patch. <sigh>

Come on, be a sport! Send me a success report!

(hey, that was even better than my original poem... this is getting bad!)

WARNING:

If you don't e-mail me about your success/failure soon, I may be forced to start SINGING. And we don't want that, do we?

(You know, it might be argued that I'm pushing this point a little too much. If you think so, why not flame me in a quick little e-mail? Please also include the type of card(s) you're using, software, size of network, and whether it's working or not.)

My e-mail address is: apenwarr@worldvisions.ca

These are the ARCnet drivers for Linux.

This new release (2.91) has been put together by David Woodhouse <dwmw2@infradead.org>, in an attempt to tidy up the driver after adding support for yet another chipset. Now the generic support has been separated from the individual chipset drivers, and the source files aren't quite so packed with #ifdefs! I've changed this file a bit, but kept it in the first person from Avery, because I didn't want to completely rewrite it.

The previous release resulted from many months of on-and-off effort from me (Avery Pennarun), many bug reports/fixes and suggestions from others, and in

particular a lot of input and coding from Tomasz Motylewski. Starting with ARCnet 2.10 ALPHA, Tomasz's all-new-and-improved RFC1051 support has been included and seems to be working fine!

Where do I discuss these drivers?

Tomasz has been so kind as to set up a new and improved mailing list. Subscribe by sending a message with the BODY "subscribe linux-arcnet YOUR REAL NAME" to listserv@tichy.ch.uj.edu.pl. Then, to submit messages to the list, mail to linux-arcnet@tichy.ch.uj.edu.pl.

There are archives of the mailing list at:
http://tichy.ch.uj.edu.pl/lists/linux-arcnet

The people on linux-net@vger.kernel.org have also been known to be very helpful, especially when we're talking about ALPHA Linux kernels that may or may not work right in the first place.

Other Drivers and Info

You can try my ARCNET page on the World Wide Web at:
http://www.worldvisions.ca/~apenwarr/arcnet/

Also, SMC (one of the companies that makes ARCnet cards) has a WWW site you might be interested in, which includes several drivers for various cards including ARCnet. Try:
http://www.smc.com/

Performance Technologies makes various network software that supports ARCnet:
http://www.perftech.com/ or ftp to ftp.perftech.com.

Novell makes a networking stack for DOS which includes ARCnet drivers. Try FTPing to ftp.novell.com.

You can get the Crynwr packet driver collection (including arcether.com, the one you'll want to use with ARCnet cards) from oak.oakland.edu:/simtel/msdos/pktdrvr. It won't work perfectly on a 386+ without patches, though, and also doesn't like several cards. Fixed versions are available on my WWW page, or via e-mail if you don't have WWW access.

Installing the Driver

All you will need to do in order to install the driver is:

```
make config
        (be sure to choose ARCnet in the network devices
        and at least one chipset driver.)
make clean
make zImage
```

If you obtained this ARCnet package as an upgrade to the ARCnet driver in your current kernel, you will need to first copy arcnet.c over the one in the linux/drivers/net directory.

You will know the driver is installed properly if you get some ARCnet messages when you reboot into the new Linux kernel.

There are four chipset options:

1. Standard ARCnet COM90xx chipset.

This is the normal ARCnet card, which you've probably got. This is the only chipset driver which will autoprobe if not told where the card is.
It following options on the command line:
com90xx=[<io>[,<irq>[,<shmem>]]][,<name>] | <name>

If you load the chipset support as a module, the options are:
io=<io> irq=<irq> shmem=<shmem> device=<name>

To disable the autoprobe, just specify "com90xx=" on the kernel command line.
To specify the name alone, but allow autoprobe, just put "com90xx=<name>"

2. ARCnet COM20020 chipset.

This is the new chipset from SMC with support for promiscuous mode (packet sniffing), extra diagnostic information, etc. Unfortunately, there is no sensible method of autoprobing for these cards. You must specify the I/O address on the kernel command line.
The command line options are:
com20020=<io>[,<irq>[,<node_ID>[,backplane[,CKP[,timeout]]]]][,name]

If you load the chipset support as a module, the options are:
io=<io> irq=<irq> node=<node_ID> backplane=<backplane> clock=<CKP>
timeout=<timeout> device=<name>

The COM20020 chipset allows you to set the node ID in software, overriding the default which is still set in DIP switches on the card. If you don't have the COM20020 data sheets, and you don't know what the other three options refer to, then they won't interest you - forget them.

3. ARCnet COM90xx chipset in IO-mapped mode.

This will also work with the normal ARCnet cards, but doesn't use the shared memory. It performs less well than the above driver, but is provided in case you have a card which doesn't support shared memory, or (strangely) in case you have so many ARCnet cards in your machine that you run out of shmem slots. If you don't give the IO address on the kernel command line, then the driver will not find the card.
The command line options are:
com90io=<io>[,<irq>][,<name>]

If you load the chipset support as a module, the options are:
io=<io> irq=<irq> device=<name>

4. ARCnet RIM I cards.

These are COM90xx chips which are _completely_ memory mapped. The support for these is not tested. If you have one, please mail the author with a success report. All options must be specified, except the device name.
Command line options:
 arcrimi=<shmem>,<irq>,<node_ID>[,<name>]

If you load the chipset support as a module, the options are:
 shmem=<shmem> irq=<irq> node=<node_ID> device=<name>

Loadable Module Support

Configure and rebuild Linux. When asked, answer 'm' to "Generic ARCnet support" and to support for your ARCnet chipset if you want to use the loadable module. You can also say 'y' to "Generic ARCnet support" and 'm' to the chipset support if you wish.

```
make config
make clean
make zImage
make modules
```

If you're using a loadable module, you need to use insmod to load it, and you can specify various characteristics of your card on the command line. (In recent versions of the driver, autoprobing is much more reliable and works as a module, so most of this is now unnecessary.)

For example:

```
cd /usr/src/linux/modules
insmod arcnet.o
insmod com90xx.o
insmod com20020.o io=0x2e0 device=eth1
```

Using the Driver

If you build your kernel with ARCnet COM90xx support included, it should probe for your card automatically when you boot. If you use a different chipset driver complied into the kernel, you must give the necessary options on the kernel command line, as detailed above.

Go read the NET-2-HOWTO and ETHERNET-HOWTO for Linux; they should be available where you picked up this driver. Think of your ARCnet as a souped-up (or down, as the case may be) Ethernet card.

By the way, be sure to change all references from "eth0" to "arc0" in the HOWTOs. Remember that ARCnet isn't a "true" Ethernet, and the device name is DIFFERENT.

Multiple Cards in One Computer

Linux has pretty good support for this now, but since I've been busy, the ARCnet driver has somewhat suffered in this respect. COM90xx support, if compiled into the kernel, will (try to) autodetect all the installed cards.

If you have other cards, with support compiled into the kernel, then you can just repeat the options on the kernel command line, e.g.:
LILO: linux com20020=0x2e0 com20020=0x380 com90io=0x260

If you have the chipset support built as a loadable module, then you need to do something like this:
```
insmod -o arc0 com90xx
insmod -o arc1 com20020 io=0x2e0
insmod -o arc2 com90xx
```
The ARCnet drivers will now sort out their names automatically.

How do I get it to work with...?

NFS: Should be fine linux->linux, just pretend you're using Ethernet cards. oak.oakland.edu:/simtel/msdos/nfs has some nice DOS clients. There is also a DOS-based NFS server called SOSS. It doesn't multitask quite the way Linux does (actually, it doesn't multitask AT ALL) but you never know what you might need.

With AmiTCP (and possibly others), you may need to set the following options in your Amiga nfstab: MD 1024 MR 1024 MW 1024
(Thanks to Christian Gottschling <ferksy@indigo.tng.oche.de> for this.)

Probably these refer to maximum NFS data/read/write block sizes. I don't know why the defaults on the Amiga didn't work; write to me if you know more.

DOS: If you're using the freeware arcether.com, you might want to install the driver patch from my web page. It helps with PC/TCP, and also can get arcether to load if it timed out too quickly during initialization. In fact, if you use it on a 386+ you REALLY need the patch, really.

Windows: See DOS :) Trumpet Winsock works fine with either the Novell or Arcether client, assuming you remember to load winpkt of course.

LAN Manager and Windows for Workgroups: These programs use protocols that are incompatible with the Internet standard. They try to pretend the cards are Ethernet, and confuse everyone else on the network.

However, v2.00 and higher of the Linux ARCnet driver supports this protocol via the 'arc0e' device. See the section on "Multiprotocol Support" for more information.

Using the freeware Samba server and clients for Linux, you can now interface quite nicely with TCP/IP-based WfWg or Lan Manager networks.

Windows 95: Tools are included with Win95 that let you use either the LANMAN

style network drivers (NDIS) or Novell drivers (ODI) to handle your ARCnet packets. If you use ODI, you'll need to use the 'arc0' device with Linux. If you use NDIS, then try the 'arc0e' device. See the "Multiprotocol Support" section below if you need arc0e, you're completely insane, and/or you need to build some kind of hybrid network that uses both encapsulation types.

OS/2: I've been told it works under Warp Connect with an ARCnet driver from SMC. You need to use the 'arc0e' interface for this. If you get the SMC driver to work with the TCP/IP stuff included in the "normal" Warp Bonus Pack, let me know.

ftp.microsoft.com also has a freeware "Lan Manager for OS/2" client which should use the same protocol as WfWg does. I had no luck installing it under Warp, however. Please mail me with any results.

NetBSD/AmiTCP: These use an old version of the Internet standard ARCnet protocol (RFC1051) which is compatible with the Linux driver v2.10 ALPHA and above using the arc0s device. (See "Multiprotocol ARCnet" below.) ** Newer versions of NetBSD apparently support RFC1201.

Using Multiprotocol ARCnet

The ARCnet driver v2.10 ALPHA supports three protocols, each on its own "virtual network device":

arc0 - RFC1201 protocol, the official Internet standard which just happens to be 100% compatible with Novell's TRXNET driver. Version 1.00 of the ARCnet driver supported _only_ this protocol. arc0 is the fastest of the three protocols (for whatever reason), and allows larger packets to be used because it supports RFC1201 "packet splitting" operations. Unless you have a specific need to use a different protocol, I strongly suggest that you stick with this one.

arc0e - "Ethernet-Encapsulation" which sends packets over ARCnet that are actually a lot like Ethernet packets, including the 6-byte hardware addresses. This protocol is compatible with Microsoft's NDIS ARCnet driver, like the one in WfWg and LANMAN. Because the MTU of 493 is actually smaller than the one "required" by TCP/IP (576), there is a chance that some network operations will not function properly. The Linux TCP/IP layer can compensate in most cases, however, by automatically fragmenting the TCP/IP packets to make them fit. arc0e also works slightly more slowly than arc0, for reasons yet to be determined. (Probably it's the smaller MTU that does it.)

arc0s - The "[s]imple" RFC1051 protocol is the "previous" Internet standard that is completely incompatible with the new standard. Some software today, however, continues to support the old standard (and only the old standard) including NetBSD and AmiTCP. RFC1051 also does not support RFC1201's packet splitting, and the MTU of 507 is still

smaller than the Internet "requirement," so it's quite possible that you may run into problems. It's also slower than RFC1201 by about 25%, for the same reason as arc0e.

The arc0s support was contributed by Tomasz Motylewski and modified somewhat by me. Bugs are probably my fault.

You can choose not to compile arc0e and arc0s into the driver if you want - this will save you a bit of memory and avoid confusion when eg. trying to use the "NFS-root" stuff in recent Linux kernels.

The arc0e and arc0s devices are created automatically when you first ifconfig the arc0 device. To actually use them, though, you need to also ifconfig the other virtual devices you need. There are a number of ways you can set up your network then:

1. Single Protocol.

   This is the simplest way to configure your network: use just one of the two available protocols. As mentioned above, it's a good idea to use only arc0 unless you have a good reason (like some other software, ie. WfWg, that only works with arc0e).

   If you need only arc0, then the following commands should get you going:

```
ifconfig arc0 MY.IP.ADD.RESS
route add MY.IP.ADD.RESS arc0
route add -net SUB.NET.ADD.RESS arc0
[add other local routes here]
```

   If you need arc0e (and only arc0e), it's a little different:

```
ifconfig arc0 MY.IP.ADD.RESS
ifconfig arc0e MY.IP.ADD.RESS
route add MY.IP.ADD.RESS arc0e
route add -net SUB.NET.ADD.RESS arc0e
```

   arc0s works much the same way as arc0e.

2. More than one protocol on the same wire.

   Now things start getting confusing. To even try it, you may need to be partly crazy. Here's what *I* did. :) Note that I don't include arc0s in my home network; I don't have any NetBSD or AmiTCP computers, so I only use arc0s during limited testing.

   I have three computers on my home network; two Linux boxes (which prefer RFC1201 protocol, for reasons listed above), and one XT that can't run Linux but runs the free Microsoft LANMAN Client instead.

   Worse, one of the Linux computers (freedom) also has a modem and acts as a router to my Internet provider. The other Linux box (insight) also has its own IP address and needs to use freedom as its default gateway. The XT (patience), however, does not have its own Internet IP address and so I assigned it one on a "private subnet" (as defined by RFC1597).

To start with, take a simple network with just insight and freedom. Insight needs to:

- talk to freedom via RFC1201 (arc0) protocol, because I like it more and it's faster.
- use freedom as its Internet gateway.

That's pretty easy to do. Set up insight like this:

```
ifconfig arc0 insight
route add insight arc0
route add freedom arc0  /* I would use the subnet here (like I said
                           to to in "single protocol" above),
                           but the rest of the subnet
                           unfortunately lies across the PPP
                           link on freedom, which confuses
                           things. */
route add default gw freedom
```

And freedom gets configured like so:

```
ifconfig arc0 freedom
route add freedom arc0
route add insight arc0
/* and default gateway is configured by pppd */
```

Great, now insight talks to freedom directly on arc0, and sends packets to the Internet through freedom. If you didn't know how to do the above, you should probably stop reading this section now because it only gets worse.

Now, how do I add patience into the network? It will be using LANMAN Client, which means I need the arc0e device. It needs to be able to talk to both insight and freedom, and also use freedom as a gateway to the Internet. (Recall that patience has a "private IP address" which won't work on the Internet; that's okay, I configured Linux IP masquerading on freedom for this subnet).

So patience (necessarily; I don't have another IP number from my provider) has an IP address on a different subnet than freedom and insight, but needs to use freedom as an Internet gateway. Worse, most DOS networking programs, including LANMAN, have braindead networking schemes that rely completely on the netmask and a 'default gateway' to determine how to route packets. This means that to get to freedom or insight, patience WILL send through its default gateway, regardless of the fact that both freedom and insight (courtesy of the arc0e device) could understand a direct transmission.

I compensate by giving freedom an extra IP address - aliased 'gatekeeper' - that is on my private subnet, the same subnet that patience is on. I then define gatekeeper to be the default gateway for patience.

To configure freedom (in addition to the commands above):

```
ifconfig arc0e gatekeeper
route add gatekeeper arc0e
route add patience arc0e
```

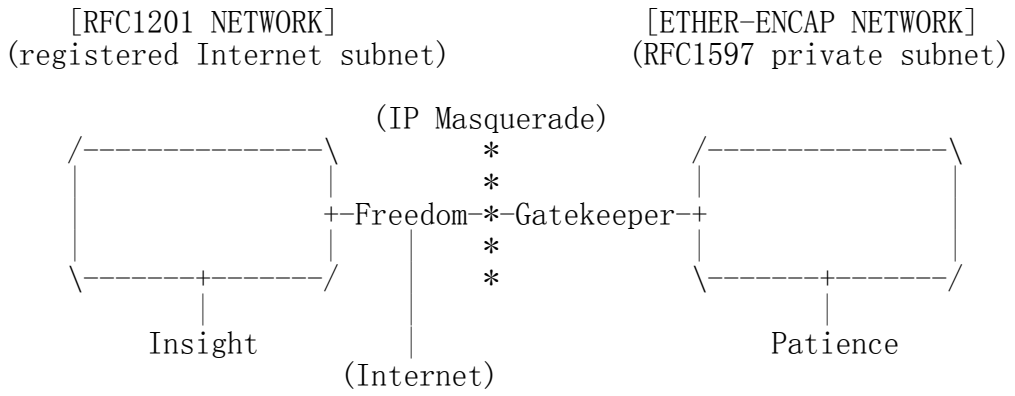
This way, freedom will send all packets for patience through arc0e, giving its IP address as gatekeeper (on the private subnet). When it

talks to insight or the Internet, it will use its "freedom" Internet IP address.

You will notice that we haven't configured the arc0e device on insight. This would work, but is not really necessary, and would require me to assign insight another special IP number from my private subnet. Since both insight and patience are using freedom as their default gateway, the two can already talk to each other.

It's quite fortunate that I set things up like this the first time (cough cough) because it's really handy when I boot insight into DOS. There, it runs the Novell ODI protocol stack, which only works with RFC1201 ARCnet. In this mode it would be impossible for insight to communicate directly with patience, since the Novell stack is incompatible with Microsoft's Ethernet-Encap. Without changing any settings on freedom or patience, I simply set freedom as the default gateway for insight (now in DOS, remember) and all the forwarding happens "automagically" between the two hosts that would normally not be able to communicate at all.

For those who like diagrams, I have created two "virtual subnets" on the same physical ARCnet wire. You can picture it like this:



It works: what now?

Send mail describing your setup, preferably including driver version, kernel version, ARCnet card model, CPU type, number of systems on your network, and list of software in use to me at the following address:

apenwarr@worldvisions.ca

I do send (sometimes automated) replies to all messages I receive. My email can be weird (and also usually gets forwarded all over the place along the way to me), so if you don't get a reply within a reasonable time, please resend.

It doesn't work: what now?

Do the same as above, but also include the output of the ifconfig and route

commands, as well as any pertinent log entries (ie. anything that starts with "arcnet:" and has shown up since the last reboot) in your mail.

If you want to try fixing it yourself (I strongly recommend that you mail me about the problem first, since it might already have been solved) you may want to try some of the debug levels available. For heavy testing on D_DURING or more, it would be a REALLY good idea to kill your klogd daemon first! D_DURING displays 4-5 lines for each packet sent or received. D_TX, D_RX, and D_SKB actually DISPLAY each packet as it is sent or received, which is obviously quite big.

Starting with v2.40 ALPHA, the autoprobe routines have changed significantly. In particular, they won't tell you why the card was not found unless you turn on the D_INIT_REASONS debugging flag.

Once the driver is running, you can run the arcdump shell script (available from me or in the full ARCnet package, if you have it) as root to list the contents of the arcnet buffers at any time. To make any sense at all out of this, you should grab the pertinent RFCs. (some are listed near the top of arcnet.c). arcdump assumes your card is at 0xD0000. If it isn't, edit the script.

Buffers 0 and 1 are used for receiving, and Buffers 2 and 3 are for sending. Ping-pong buffers are implemented both ways.

If your debug level includes D_DURING and you did NOT define SLOW_XMIT_COPY, the buffers are cleared to a constant value of 0x42 every time the card is reset (which should only happen when you do an ifconfig up, or when Linux decides that the driver is broken). During a transmit, unused parts of the buffer will be cleared to 0x42 as well. This is to make it easier to figure out which bytes are being used by a packet.

You can change the debug level without recompiling the kernel by typing:

```
        ifconfig arc0 down metric 1xxx
        /etc/rc.d/rc.inet1
```

where "xxx" is the debug level you want. For example, "metric 1015" would put you at debug level 15. Debug level 7 is currently the default.

Note that the debug level is (starting with v1.90 ALPHA) a binary combination of different debug flags; so debug level 7 is really 1+2+4 or D_NORMAL+D_EXTRA+D_INIT. To include D_DURING, you would add 16 to this, resulting in debug level 23.

If you don't understand that, you probably don't want to know anyway. E-mail me about your problem.

I want to send money: what now?

Go take a nap or something. You'll feel better in the morning.