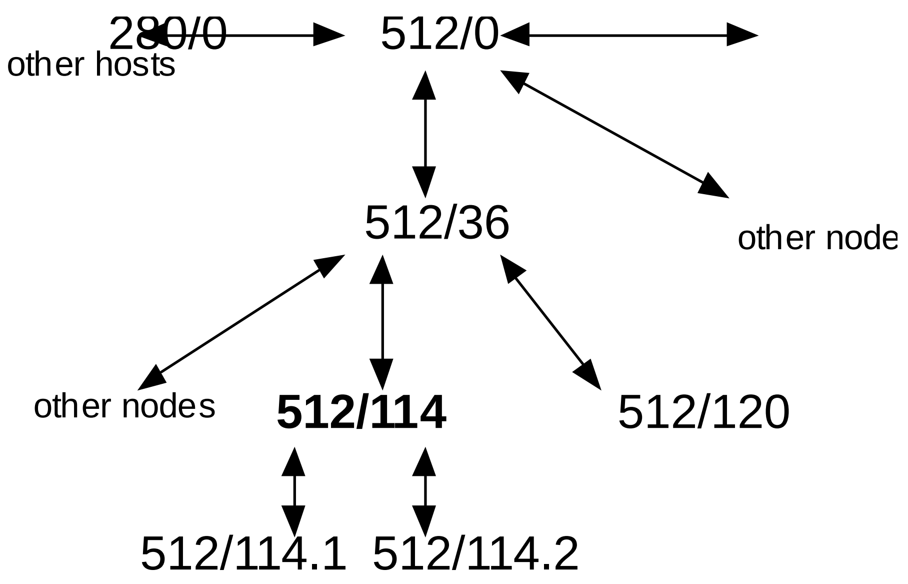
At the MacWorld in Boston, I was asked to write a manual explaining the finder points of file echo's.

We all know message echo's of course. We set them up with Tabby Maint, or, for those of us with more courage, with a simple text editor on a file called 'areas.bbs'. Let us look at a sample set-up and get familiar with the network I am going to use to explain the file echo thing.

This is it:



People who do business on a regular base with me know that I am node number 512/114. I put that node number in bold in the above picture. I have put the 'regular' mail flow in the picture with arrows. My node exchanges mail with two points (there are actually more, but that's not relevant to my discussion), and my hub, 512/36, where I get most of my echomail. I know that my hub, 512/36, gets most of his echomail from the network host, 512/0, and also serves 512/120 and some other nodes. Thus, whenever one of my points writes a message in the echo called 'TABBY', he sends it to 512/114, who sends it to the other point and to 512/36. 512/36 then sends it on to 512/0, 512/120 and any other nodes connected to him. This way, the message finds it way to the entire network.
There are a couple of questions a node must ask himself when processing a message:
1) who sent the message to me?
(after all, we don't want to send the message to him again)
2) who is connected to the echo that this message belongs to?
(after all, we don't want to send the message to any node not interested in TABBY)
3) who has already seen this message (through, perhaps, a different route)

Question three is a bit of a problem - it is only there to prevent duplicate messages from going out to the network. When a message from 512/114 reaches 512/36, 512/36 will usually need to send it to all nodes connected to 512/36 who are interested in TABBY except *to 512/114. Things could go wrong if, for example, 512/114 is connected to 512/120 as well for TABBY - that way, a circle would be created, and this can cause havoc on a network if the echomail processors would not detect it. Circular* set-ups for one echo are to be avoided!

Now, how does an echomail processor answers questions 1 through 3? Simple. They add information to the echomail message. Most systems do not show you this information, but some do. CounterPoint, for example, allows you to keep the 'SEEN-BY' lines, which is where the most important information is kept.

Also, Tabby will figure out who else needs to read the message by looking in the 'areas.bbs' file in your Tabby folder. This file can be edited by Tabby Maint, and the relevant line in it looks something like this:

```
1           TABBY 512/36 1114/1 1114/2
```

(this means that any message sent in the TABBY area gets sent to 512/36 and to points 1 and 2 (1114 is my private net number - I am using groupmail in reality, but that would have confused this discussion))

This is not all there is to say about echo-mail, of course. Because echomail tends to be quite large, it is usually compressed into a single file, and this file is 'attached' to a small mail message and sent out. The small mail message gets interpreted by the other system - it refers to the large, compressed, bundle, which gets unpacked and processed.

How does all this show on your own system. Take a look at your Tabby Generic folder. It contains a certain number of files that are of interest to this discussion:

dup data base
This is where Tabby keeps track of a large number of messages. This data base is used to decide whether an incoming message is original, or whether it is already seen by this system. This way, duplicate messages are largely prevented from spreading further.

sendmail512/36.bbs
This file keeps the outgoing messages for system 512/36. If it gets too large, Tabby compresses this file into a file called 00DE0072.MO1 (or something similar), and creates a new, small sendmail512/36.bbs. Also, a file called **sendfiles512/36.bbs** is created. This does nothing but list the files to be sent to 512/36. It will list the 00DE0072.MO1 file.

Now wait a second, I hear you ask. This makes it really trivial to send additional files to 512/36 - just list them in the sendfiles512/36.bbs file! Correct. But there's a catch, of course.

Any file listed in the sendfiles512/36.bbs ends up in the folder with incoming mail on the 512/36 computer. If this would be a Tabby based system, it would end up in the Tabby Generic folder. But what next? We have seen that the echomail gets processed by Tabby because a) a reference to the 00DE0072.MO1 file is passed on, and b) all other information is kept with the messages. This means messages will be passed on to other systems as well. Any other file just sent by putting it's name into sendfiles512/36.bbs will sit there, forever (or until the sysop of 512/36 needs the diskspace more than he needs the file).

From the looks of it, it is therefore not possible to distribute a file through the network. My node 512/114, can send to 512/36 or to my points, and, by using direct routing, to other nodes as well, but I would really like to distribute files using the same network structure as echomail uses - after all, we are making those calls already...

Some years ago, a smart IBM based sysop figured out something to do just that. He figured that not only the file should be sent to the other system, but also a small control file containing all the information that the other system would need to figure out what to do with the file - this would allow for a lot of things; the description of the file could go in there, the SEEN-BY lines, which path the file has followed, etc. Also, it would allow files to belong to a certain 'area', just like echomail messages. You can recognise these small control files by their name. They start with 'TK', followed by 6 digits, and end with '.TIC'
(and, these files get interpreted by the Tick or MacTick programs)

Since they are made up of simple text lines, let's take a look at a sample TIC file:

```
Area TICKTEST
Origin 2:512/36
From 2:512/36
File TICKTEST.ARC
Desc Testfile for TickTest echo
CRC C4047ADF
Created by Hatch v2.00 (C) Copyright Barry Geller - 1988,1989,1990
Path 2:512/36 663786782 Sun Jan 13 17:13:02 1991 GMT
Seenby 2:512/36
Seenby 2:512/114
Pw blahblah
```

The first line tells us what area the file belongs to. The area TICKTEST is one I set up with my host and one of my points to test the MacTick version under development.
The 'Origin' tells us which system has generated the file. The file may have been generated by some Australian system, tranferred through the USA, and several European countries, and still I would be able to tell which system generated this file.
The 'From' line tells me which system actually sent it to ME. In this case, the origin and from lines tell me the same thing - I am very close to the creator of the file - just one call away,.
The 'File' line tells me the name of the file. This means that I know which file all this actually refers to and it will sit in my Generic folder with this name.
The 'Desc' line tells me what this file actually is.
The 'CRC' tells me what a 32-bits Cyclic Redundancy Check should be for the file. If I test the file with the CRC method, the result should equal what this line reads - this way I have a way to make sure that the TickTest.ARC file is the same that 512/36 sent me.
The 'Created' line tells me which program originally created the TIC file.
The 'Path' line may occur many, many times in the file - every system that processes the file adds one of these lines - this way the file can be traced back to it's origin! The somewhat long number in the line is the date in Unix format - the number of seconds since Jan 1, 1970. This way I can pin-point to the second when each system processed the file.
The seenby-lines tell me which systems already 'saw' this file. 'But...' I hear you say, isn't this the same information that's in the path lines? No. Take a look back at the picture and imagine that 512/36 sends a file to 512/0, 512/120, and 512/114. Then, the path lines would only mention 512/36, but the seen-by lines would mention all the systems. After all, only 512/36 processed the file, but everybody has 'seen' it. This would prevent my system from sending it to 512/0 as well.
The 'Pw' line mentions a password, that let's me make sure it was actually 512/36 who sent it, and not some bogus system (note that 'blahblah' is NOT the actual password I use).

Now, all we have to do is list the TIC file and the actual file in the sendfiles512/36.bbs file, and that would do it. Fortunately, all this is automated in the MacTick and Hatch applications, and the sysop never needs to know about sendfiles.bbs files and the such.

Now, how to set up?

Let's say 512/0 wants to distribute a file called NETWORK.512 on a weekly basis, listing all nodes in the 512 network. This may sound very familiar, since most networks have this thing. 512/0 would tell everybody that there's a new file echo called NETWORK512, and that he will start distributing files in it. Next, he would set up his own system to send files to 512/36 and all other hubs. He would also set it up disallowing 512/36 and other hubs to send files in this echo. After all, he is the only one who is authorised to create these network files.

512/36 on the other hand, would set up his NETWORK512 echo to connect to 512/0, 512/120, and 512/114. He would disallow all nodes except 512/0 to send in this echo (fortunately, both IBM and Mac versions of Tick allow for this kind of security).

512/114 would set up his system to connect to 512/36 and all points, again only allowing 512/36 to send in this echo, since 512/36 is, from the point of view of 512/114, the only node allowed to send files in this echo.

That's basically all there is to setting up the echo (apart form the fact that all sysops will very likely set it up to 'overwrite' duplicate files. Since the file has the same name every week, there will be only the latest version online at any time).

There's one other catch here. Macintosh files differ quite a bit from IBM based files. They are actually TWO files, a data fork and a resource fork. Way back in 1984 somebody smart figured out a way to concatenate these two files to one file (with a small header that told receiving Mac's how to recreate the original dual file), thus allowing Dos

systems to get the file as well. It's called MacBinary. There's only one problem here - the file get's renamed on conversion.
The conversion from a Mac file to a 'dos' file is done either when Tabby needs to send it through a telephone line, or much earlier by any application that can create this 'dos' like file. BinHatch does just that. This way, the file name will still match what's in the TIC file when it reaches the other system. The conversion back to a Mac file is done by ArcMail Extract, so when we know that a file is hatched (created) in the MacBinary format, we want to make sure that MacTick get's hold of it before ArcMail Extract does, because ArcMail Extract will rename the file BACK to the macintosh name, again breaking the link to the TIC file. This means that the files will end up in MacBinary format in the file area's, but this is OK, since virtually every telecom package on the Mac speaks MacBinary. There's an extension for Stuffit Deluxe as well, and many other programs know about it too.
The careful reader will have noticed that it makes a difference wether the file is hatched in MacBinary format, or Tabby does the conversion. In the first case, the name in the TIC file will match the converted file name, and every system will be able to process the file as long as they do so before ArcMail Extract runs. In the second case, the file name will match the name after ArcMail Extract is done with it, and IBM based systems will never be able to process the file at all, since the TIC file never points to the right file name for them.

Luckily, the creators of the Macintosh Shareware Distribution Network, which is the most important network for Macintosh files, set the standard to hatch in Macbinary format (forcing me to rewrite Hatch, but that's another story).

-John W. Sinteur
(with thanks to the 512 network for the example)