# An Introduction to FlexNet

Gunter Jost, DK7WJ
FlexNet Gruppe Darmstadt
Lichtenbergstrasse 77, D-64289 Darmstadt, Germany

Donald Rotolo, N2IRZ
Radio Amateur Telecommunications Society
PO Box 93, Park Ridge NJ 07656, USA

## Abstract

Features and operation of FlexNet packet networking software are discussed. Details of the software architecture, RMNC and MS-DOS hardware platforms, applications, user interface, adaptive parameters, and routing techniques are presented.

## Introduction

FlexNet is a flexible, modular and user-friendly software program for packet radio networking. First conceived in 1987, the software has undergone many changes and improvements since then, and is presently (July 1995) at version 3 -3. The most notable features of FlexNet are:

- Autorouter: Automatically routes all connect requests with minimal input required of the user.
- Adaptive Parameters: All network parameters (except TXDelay) are automatically set by the software, and adapt to changing channel conditions.
- Hop-to-hop Acknowledge: Each packet is ack'd directly by its neighbor, improving data transport reliability.
- Command Interpreter: The software interacts directly with the user, and has numerous internal applications providing detailed data concerning the configuration and status of the node.
- Remote Controllability: The node is completely remote-controllable, also allowing for remote tracing of a QSO with several filters.
- Modular and Portable Architecture: over 95 % is written in C, allowing wide portability across platforms.
- DAMA master and slave implemented. All channels may be master or slave, or mixed as desired.

FlexNet is presently the most widely used packet radio networking software in Germany, with over 500 installations. Due to German amateur radio regulations, each unattended station (node) must receive special permission to operate, and must use tightly coordinated dedicated point-to-point links. Despite creating problems and red tape, these limitations have forced the creation of a very high-quality network, where round-trip response times over hundreds of kilometers and tens of node sites are typically only a few seconds.

Although FlexNet is primarily used for networking, the PC version can also be used by users by simply eliminating the Node module. PC/FlexNet is thus a flexible yet powerful replacement for permanent TNC emulations (such as TFPCX). It is extremely simple to set up, since there are no hassles with parameters.

## Hardware

FlexNet is presently ported to two hardware platforms: The Rhein-Main Net Controller (RMNC) system and the Intel 80x86 processor running MS-DOS.

The RMNC platform is a 6809-based system using a 28530 SCC. Each channel control card (one for each radio/data channel) is on a standard Eurocard, and is plugged into a standard card cage with backplane bus. One master card can control up to 15 slave cards. Developed by the Frankfurt (Rhein-Main) Packet Radio Group, the RMNC remains the preferred platform for FlexNet because of its low cost and high performance. The software resides on a single EPROM.

An MS-DOS version has existed since 1990, but was not distributed, as it was used primarily as a test and development platform. In 1994, development of a stable version for MS-DOS was begun in earnest. FlexNet runs problem-free in an XT-class system, although faster processors are preferred. Several different channel drivers exist, allowing for some flexibility in I/O hardware.

The choice of MS-DOS was based primarily upon the large installed base. Admittedly, a better choice would be LinuX, and in fact the task of porting FlexNet over is already being worked upon.

## Architecture

In 1994, a discussion between the present FlexNet author (DK7WJ) and BayCom author Flori Radlherr DLSMBT yielded a new concept for FlexNet: Modularity. The major portions of the software were developed as separate modules, the most important being the channel drivers and applications. These modules can be used in combinations as desired, on either hardware platform, creating a very flexible system, and allowing a larger number of authors to contribute talent. In the future, a "developer's kit" will be available, further easing module development.

A number of driver modules presently exist:

- ◆•♍♍ Driver for all BayCom SCC cards. The card type is automatically detected.
- SER12 Driver for the serial BayCom AFSK modem.
- ☺☃♦♦ Driver for connection to a computer. This driver should not be used with a TNC, as the channel timing is not as precise as it should be, especially in the half-duplex mode. FIFO UARTs are supported, allowing data rates up to 115,200 baud on a 28606.
- 6PACK Driver for TNCs. 6PACK is a new protocol that overcomes timing problems with KISS and TNCs. This protocol can control up to 8 TNCs in a ring (without tokens!), however the data rate on the RS-232 side should be greater than the sum of the RF baud rates. Beta-testing is currently underway, involves replacing the TNC's EPROM.
- PAR96 Driver for the parallel version of the BayCom 9600 baud G3RUH-compatible modem
- IPXN Driver for Novell-IPX networks. Those that already have such a network running can link PCs using FlexNet. Traffic runs in the local net segment as broadcasts, and each FlexNet station monitors the net traffic.
- IPXPD The same as IPXN, except that the packet driver is replaced with the IPX protocol itself. To be compatible with an IPXN system on Ethernet, only used when a Novell installation is not available.
- IPPD AX/IP, a provision of IP to allow point-to-point connections over Ethernet. ARP is rudimentarily implemented. Not a solution for an intemet link, but functional. Eventually , this will be integrated into a TCP/IP module.
- VANESSA Driver for the VANESSA channel card. This card from the Swiss SEPRAN group can control 2 RF channels and has its own processor, significantly reducing the PC's workload.

Other drivers being developed include those for TMC320C25 DSP systems as well as Sound Blaster cards. Interested software authors are invited to contact DK7WJ for further information about developing additional drivers.

## Applications

The FlexNet system supports several applications, primarily to provide users and Sysops with information about the network or a node. These applications are mentioned in "User Interface" below. Due to the modular design of FlexNet, implementing new applications is relatively easy, and can be written by anyone having sufficient programming experience.

Three unique applications deserve special mention:

- TFEMU — Hostmode emulation. PC/FlexNet can be used with this application as a driver for various hostmode programs, such as BBS, DXCluster, terminal programs, etc.
- [illegible] — FlexNet can be used with this application to emulate an ethernet packet driver, allowing for example NCSA-Telnet or Winsocket applications to be worked via AX.25 Datagrams or Virtual Circuits.
- [illegible] — Remote DOS interface for service. This application allows full remote control of MS-DOS, even such local utilities such as formatting a disk.

## User Interface

FlexNet appears as a simple interactive application to the user. Various commands are used to set up connections, access applications, etc. The user connects to the node's user port and sends the desired command.

To initiate a connection, the user need merely specify the destination, although other options including manual routing, single-stepping from node to node and partial autorouting are possible.

The commands available to the user are summarized below. A complete description of all commands is given in the Sysop documentation, presently available in German and English.

**< A > ktuell (Current info)**
Displays a current-inteiest text uploaded by the sysop.

**<B>eacon**
Displays the Beacon text. This file shows which beacon is sent at what interval from which ports.

**< C > onverse**
Starts FlexNet Converse mode (if sent without a callsign following, otherwise see CONNECT below). 255 channels are available. Function and usage of the FlexNet Converse mode is similar to other popular conference systems.

**< C > onnect**
Starts a connect request to another station when followed by a callsign. A connect command can also be established by the user without being connected to the node, by specifying the entry and exit points of the network, and the destination. The node routes (see "Routing" below) SABM frames toward the destination, and sends the message "link setup.. ." to the user. If the connection is ack'd by the destination station, the user receives the message "*** COMKtd to < call > ". When unsuccessful, the message "*** failure with < call > " is sent to the user. If a DM is received from the destination, the message "*** busy from < call > " is sent to the user. The connect request can be broken by sending a < CR > or a new connect request. If a user attempts to connect to a station he is already connected to, he receives the message "*** can't connect twice". This command can also be used to connect to a different user port at a node site that has more than one user port. For example, the command "C-7" connects the user to the port with SSID 7, which is acknowledged with the message n*** <call > : ssid ok"
After using the Connect mode, if the other station disconnects, the user is reconnected to the last node connected to, which is announced with the message "*** reconnected to < call > ".
It is not permitted to manually route a connect request to form a loop, where the connection passes through the same node twice. If an attempt is made, the user is informed with "*** < call > loop detected", and reconnected to the previous node.

**<D > estinations**
Shows the automatically-generated destination table. Information includes the callsign, SSID range, measured ping time (1OOms steps) and, optionally, the entire route to the destination.
It should be noted that the destination list is nearly identical in every node in the network, since all updates are propagated throughout the network immediately.
The user can also selectively view the destination table, for example "D HB9" shows all destinations having callsigns beginning with HB9.

**<F>ind**
**This comman**d, one of the most powerful in FlexNet, sends a request to all nodes in the network (can be limited by the sysop, see also Setsearch) to find a particular station. Each node sends an UNproto frame, addressed to the station to be found. If the station hears this frame, it will send a DM frame. When this occurs, a message is sent back to the user exactly where the desired station can be found. If there is no response, the user receives no message. Since UNproto frames can get lost, the user should send the command a few times to be sure. Naturally, the AX.25 protocol requires that the correct SSID be used.

**<H>elp**
Sends a help file uploaded by the sysop.

**<I>nfo**
Sends an information file uploaded by the sysop

**<L>inks**
Lists the Sysop-defined link table, showing all ports, neighboring nodes, and their status.

**<LO>cal**
Sends the node's connect text file. Always sent when connecting directly to the user port, this cornman d allows the text to be viewed remotely.

**<M>heard**
Displays up to 200 heard callsigns with time stamp and channel number.

**<MY>call**
Displays the node's callsign and SSID range.

**< P > arameters**
Displays a detailed list of all ports at the node, their parameters, various link statistics and their status.

**<Q>uit**
Disconnects the user from the node, after the node sends "73! ".

**< S > etsearch**
Shows which nodes will transmit the Find message.

**<U>sers**
Lists all users of the node, including those just passing through. Shows QSO number (internally generated), connection status, data frames unack'd, port, and callsigns involved.

**< IO > (Input/Ouptut)**
Shows status of each of the 16 binary inputs and outputs (RMNC version).

**Sysop Commands:**
The following additional or expanded commands are available exclusively to the Sysop:

| | |
|---|---|
| <L>inks | Set links list |
| < M > ycall | Set callsign and SSID range |
| <P>arms | Set various parameters |
| <IO> | Read all input lines and/or set all output lines |
| <CA> | Send a CALibrate signal |
| < K > ill | Kill a specific QSO |
| <MO>de | Set HDLC parameters |

| | |
|---|---|
| <SY>sop | Sysop authentication |
| <W>rite | Upload various text files |
| <T>race | Monitor all activity on a channel remotely |
| < RESET > | Cold reset |
| < RESTART > | Warm reset |

## Connection Phases

There are four possible phases for any given connection or QSO: Link Setup; Information Transfer; Link Failure; and Disconnect. Each phase is discussed briefly:

### 1. Link Setup

During a connection setup, the connect (SABM) frames are simply routed through the network (see "Routing Methods", below), without being directly acknowledged (No UA Frame). Thus, a connection can only be made when the destination station is actually available, for it is the only station that will acknowledge the SABM frame. Of course, the user can connect to any node instead of a user station, and then attempt to manually route the remainder of the path from there. However, the fully automatic routing method is significantly easier for automated stations, such as BBS store-and-forward operations.

When the called station acknowledges the connection with a UA frame, it is passed back (again without any hop-by-hop ack) to the calling station. At this point, there are two QSOs in the QSO list on every node involved: one from calling station to destination station, and one in the reverse direction. Once the calling station receives the destination station's UA frame, the link is established (as a Virtual Circuit) and information transfer can begin.

The reason for this method, instead of the more conventional direct acknowledge for SABM frames, is that very few network resources are wasted on connect requests that cannot be established, yet good connections are established fairly quickly. In the FlexNet network installed in Europe, typical response times from end to end are on the order of a few seconds, so timeouts, lost packets and other problems are rare.

### 2. Information transfer

During the information transfer phase of a QSO a 'Hop-by-hop Acknowledge' is used. Each I-Frame (packet frame containing some information) from one or the other station is directly acknowledged by the node, and then passed along to the next node along the path (also directly ack'd). Retries caused by whatever reason on a given link are thus only retries across one link and not across the entire path.

Once the path is setup, it is completely transparent to the data flowing through it. Each frame passes through the network completely unchanged. Thus, data frames with other Protocol IDs (PIDs), such as TCP/IP, are carried across the network without any problems.

### 3. Link failure

If, during information transfer, the path is broken for some reason, the end nodes report the link failure to both stations with the message "*** DBOxxx --> Link Failure" and the session is disconnected.

### 4. Disconnect

If station A sends a disconnect frame, it is immediately ack'd by the local node. The node then attempts to sever the connection to station B. If there is no data remaining 'in the pipeline' for station B, the disconnect occurs immediately. If data remains, it is first passed to station B and then the link is disconnected. If station B had sent data to station A, it is lost.

## Routing Methods

To connect it is only necessary for the user to specify the point of entry to the network and the destination, although any connection may be manually routed if desired. There are four methods available to automatically route the users connect request:

1. Routing by destination table
2. Routing by link table
3. Routing by Heard list
4. Routing by SSID

Routing is only performed for the Connect (SABM) frame. Once the Virtual Circuit is set up (the UA of the SABM is received by the calling station), all subsequent frames take the exact same path. If a failure occurs, the link is tom down after informing both ends of the failure. A failed link must be reestablished manually.

### 1. Routing by destination table

The first method is based upon routing by callsign information. The node looks at the destination callsign and compares it with a table of calls and routes that has been stored in a destination table by the autorouter. If a match is found, the call is passed along to the specified neighbor.

***(NOIIT: All node callsigns begin with DBO... J***

Example: DBOODW has the following links:
1: DBOKT
2: DBOAAC
3: DBOIE
and knows the following destinations: DBOEQ, DBOZDF, etc.

The frame
< fm DG3FBL to DK7WJ via DBOODW,DBOZDF >
is expanded to:
< fm DG3FBL to DK7WJ via DBOODW,DBOAAC,DBOZDF >

This is because DBOODW knows that DBOAAC (which it has a direct link to, on Port 2) is the next (and best choice) node along the path to DBOZDF. Thus, the frame is sent via Port 2, despite the fact that there is no direct link to DBOZDF. The 'best' path is always selected for any given destination, as determined by the average 'ping' time for a given path.

The node sends out test frames ("Pings") to each neighbor, and then stores the round-trip response time. Using the ping information, the destination tables are constantly updated in real time. If a destination becomes available (or unavailable), that information is instantly broadcast throughout the network (limited only by the propagation time through the network), thus all destination tables are nearly identical. The destination table is built automatically and cannot be changed by the operators. It is thus assured that only accessible destinations are forwarded, and instantly deleted when they (or their path) disappear.

### 2. Routing by link table

If the autorouter does not fmd an entry in the destination table, it then goes to the Sysop-defined link table. If a pre-defined route is found here, the frame is sent to the specified neighbor (on a specified port) for handling. The link table normally contains only the direct neighbors, that is, those nodes having a direct link with the node.

Example: DBOODW has the following links:
1: DBOKT
2: DBOAAC
3: DBOIE
4: DBOAIS

The frame
<fm DG3FBL to DK7WJ via DBOODW,DBOAIS>
would be routed to Port 4 so long as no entry exists in the destination table, because DBOAIS is known in the link table. Hopefully, DBOAIS would have a route to DBOODW.

**3. Routing by Heard list**
The node stores the last 200 directly heard callsigns in an internal list. This list remains in memory until a RESET occurs. An entry to this list occurs only when a station has an active connection, or when it is found using the 'Seek' command. At first, the node uses the SSID of the desired station to establish a connection but, if there is no response, then the SSID is ignored. It is therefore without problems possible to be in STBY on various ports with various SSIDs, so that one can always be found on the correct port. Incoming connect requests or also UNproto frames are routed with this list.

**4. Routing by SSID**
A fmal chance for the node to automatically route frames is based upon the SSID. Specific channels can be arranged by SSID (this can also be done by Port number). This is shown with the PARMS command. To use the SSID routing, the user specifies the SSID (Port) that the frame should be sent from.

Example: DBOODW has the following arrangement of SSID versus Channel numbers: Channel 1 has SSID 0, Channel 5 has SSID 12, Channel 6 has SSID 3, etc. If a connect frame arrives that has specified an SSID for DBOODW, the frame is routed directly to the channel specified by the SSID:

The frame < fm DG3FBL to DK7WJ via DBOODW-12> is routed to channel 5, which has the SSID of 12, so long as no other way to DK7WJ is known. The following frame is sent out from channel 5: < fm DG3FBL to DK7WJ v DBOODW-3* > , which clearly shown where the frame came from (in this case, channel 6 has the SSID 3). This also carries the entry node information. This turnaround of SSIDs is also notable, in that every frame shows where it came from.

One of the most important features of FlexNet is that all connections are reversible, that is, you are always provided sufficient info to have the same connection in the reverse direction.
This routing method is naturally used above all for user access.

If all of these routing methods fail, the connect frame is lost. If this occurs when the user is connected to the node, the message "*** < callsign > : can't route" is sent to the user. If the user issued a connect command when disconnected from the node, the TNC will eventually reach the RETry limit.

## Adaptive Parameters

All level 1 and level 2 parameters within the network (including user access channels) are either self-adjusting according to the current channel conditions or fixed in the software. The only exception to this is TXDelay, which is set by the sysop.

A RETry limit of 10 is used, to ensure a quick recovery for links subjected to temporary interference. Regular polling is used to verify that links are operational, as well as to determine the actual one-way or round trip data transfer time. The polling interval, controlled by FRAck (Tl), varies according to channel conditions. If FRAck is small, polling occurs no sooner than every 90 seconds, lengthening when FRAck becomes larger. Each node polls only its neighbors. The round-trip turnaround time is used to adjust FRAck (Tl) to the time it takes for the neighboring station to acknowledge I-frames, and is also used (along with other factors) by the autorouter to determine the best (fastest) path to other stations.

MAXframe is regulated according to the capability of the link. When the other station receives all frames without problems, MAXframe runs as high as 7. At lk2 baud, 7 frames in a row are seldom sent, since the maximum key-down time is limited to about 12 seconds. When multiple streams are active, the 12 seconds is divided up and shared. When the other station begins missing frames (for example by sending reject frames), MAXframe is reduced. In this case, the throughput is actually increased with a lower MAXframe. It should be noted that evaluation checks the link quality TO the other station, not FROM the other station, and that the value changes as necessary. A poll, in which all frames are acknowledged, or with lost acknowledgements, don't reduce MAXframe.

P-Persistence is the critical parameter when many users are on the channel simultaneously. A fixed parameter is, at best, a compromise between collision potential and transmit potential. FlexNet notes the number of users on the channel, as well as the data rate and other factors, and adjusts P-Persistence to offer the best

performance at all times. Aggressive stations no longer have the advantage of faster performance at the expense of weaker stations, yet fast downloads remain feasible when channel activity is low. The improvement over a fixed parameter is most notable on duplex user ports (the most common in Germany), but improvement is noticeable on simplex channels as well.

Concerning TXDelay, if the node hears more than 1OOms of flags from.a user station, which indicates an excessive TXDelay setting in the users TNC, a polite message is sent informing the user of this fact, and then disconnects. Thus, a user running excessive TXDelay cannot use the network until the problem is corrected.

It would be highly desirable if user software were to also follow the trend towards full adaptation to the channel conditions. It is clear that a computer can adjust to prevailing conditions faster and more accurately than a human, not to mention the fact than most users are totally baffled by TNC parameters anyway. Other problems also are caused by using a single fixed parameter to control a constantly changing channel, which would be resolved with adaptive user software.

PC/FlexNet can be installed as user software, to take full advantage of adaptive parameter adjustment, by simply omitting the Node module. It becomes a powerful yet flexible replacement for permanent TNC emulations like TFPCX - simply use FlexNet with TFEMU and channel drivers as needed. Setup is simple, with only one parameter (TXDelay) requiring operator input.

## Interoperability with TheNET

It is relatively easy and painless to include TheNET network nodes into a FlexNet network. Each TheNET node that interfaces directly with a FlexNet node appears in the FlexNet destination table. The interface must be single-stepped manually, but no serious problems occur as TheNET users are used to single-stepping ;-)

## For Further Information

This overview is necessarily brief. Complete details are available with the software (as an ASCII file) or in separate printed documentation available from the author. At this time (July 1995) the documentation is available in German as well as English.

## References:

G. Jost, DK7WJ and J. Sonnabend, DG3FBL, *"FkxNet, the European So&on"*, Proceedings of the 9th ARRL Computer Networking Conference, ~~127-133, 1990.

G. Jost, DK7WJ, *"FlexNet Sysop Manual* v3.3", 2nd edition, 1995.