DigiNed

v 0.1.7(Août 2000)

par Henk de Groot, PE1DNN

Documentation Sysop
version française v 1.0

Un Programme de Digipeater APRS venu de Hollande!
Par PE1DNN

www.qsl.net/digi_ned/

Table des matières

Contenu de la distribution

Version: `DNxxxxxx.ZIP` (DN<type><version>.ZIP)

`DNEXExxx.ZIP`	- fichier exe DOS avec fichiers supplémentaires.
`DNSRCxxx.ZIP`	- source DOS, contient le fichier source .TGZ, wrappé, car le "stupide" MS-DOS ne comprend pas les noms de fichiers longs.
`digi_ned-x.x.x.tgz`	- fichier source pour LINUX (contenu dans `DNSRCxxx.ZIP`)

Introduction

Voici DIGI_NED, un digipeater évolué utilisant un fichier d'initialisation (.INI) et qui dispose de fonctions spéciales pour le trafic APRS. APRS est une marque enregistrée de Bob Bruninga, WB4APR. Sa page web peut être trouvée à "http://web.usna.navy.mil/~bruninga/aprs.html".

Pourquoi un nouveau programme de digipeater?
Les fonctions de ce digipeater sont globalement comparables a celles des autres digipeaters évolués, mais il existait encore de bonnes raisons d'en construire un nouveau.

1. Les digipeaters existants ne font pas exactement ce que nous voudrions qu'ils fassent.
2. Certains digipeaters s'arrêtent de fonctionner lorsqu'ils reçoivent des trames non conformes au protocole, notamment pour la plupart les trames contenant plus de 8 digipeaters à l'intérieur.
3. Nous aimerions disposer du code source de manière a pouvoir ajouter des options "hardware" par la suite, comme par exemple pouvoir y connecter une stations de radiogoniométrie.
4. Il serait intéressant et économique que le digipeater puisse tourner sur un 286 avec quelquechose de plus simple que BPQ.
5. Qu'il soit opérationnel et disponible 24h/24h et 7j/7j.
6. Pouvoir y ajouter des fonctions quand nous le voulons.

Le code source de ce digipeater est disponible suivant les termes de la "General Public License".

Voici les principales caractéristiques de DIGI_NED:

1. **Intelligent**: il est basé sur des règles de configuration stockées dans un fichier .ini (d'autres digipeaters sont basés sur ce système, mais ici nous avons la possibilité d'utiliser **WIDEn-N correctement** sans se perdre dans des "paths" trop longs. Le "swap" permet le traitement correct des WIDEn-N).
2. Le digipeating intelligent fonctionne également pour le **digipeating basé sur le SSID-destination** en utilisant la règle "`digito:`".
3. L'indicatif du digipeater n'est défini qu'a **un seul et unique emplacement** (à la fin de l'exemple) dans le fichier .ini; cela permet aisément de dupliquer ce fichier .ini.
4. Il digipeat les trames dont le **PID est AX.25, IP, ARP et NETROM**. Des connexions radio en **AX.25 et TCP/IP** peuvent également êtres établies via le digipeater.
5. **Substitution d'indicatif** (avec une option qui l'inhibe) et réactivité contre les boucles de paquets (looping); en fait le système vérifie son propre indicatif dans la liste "via" du digipeater et **mémorise le CRC** des premiers paquets reçus.
6. **Réponse avec balises aux requêtes ?APRS?**.
7. Supporte les **réponses automatiques** aux messages, comme ?ping?
8. Peut fonctionner comme un "**mini-serveur-wap**" disposant de messages facilement configurables et disposant d'une banque de données des sites intéressants de la région couverte.
9. Peut couvrir les "zones d'ombre".
10. **Digipeating préemptif**, il peut remplacer les digis momentanément Hors-Sercices / Évités.
11. Fonctionne aussi sous **LINUX**.

DIGI_NED peut être utilisé avec **tous les modems qui peuvent utiliser TFPCX** (vous vous demandez comment c'est possible...) ce qui inclut aussi le protocole **BPQ sur Ethernet,KISS, BayCom, YAM, SCC (PA0HZP, PE1PET, USCC)** etc... Sous LINUX DIGI_NED utilise l'interface du kernel; tout ce qui y est connecté, doit fonctionner.

La philosophie générale de DIGI_NED est de **permettre à l'utilisateur de créer sa propre configuration**. Nous avons essayé de faire de la configuration par défaut, une configuration la plus utile possible et celle s'adaptant au maximum d'utilisateurs; mais c'est en fait l'utilisateur qui, à la fin déterminera comment fonctionne son digipeater.

Ce document est la première tentative de création d'une documentation. Je n'ai en fait que réuni en un document toutes les notes que j'ai pu prendre auparavant, mais en Hollandais. A cause de l'intérêt international grandissant de DIGI_NED, j'ai laissé de côté la documentation en Hollandais pour écrire une documentation en Anglais. Les volontaires sont les bienvenus pour écrire les documentations dans d'autres langues. Ceci sera l'unique documentation que j'intégrerai dans les distributions, de manière à éviter de refaire la documentation à chaque révision. Ce document va s'agrandir au fur et à mesure que j'ajouterai de nouvelles fonctionnalités.

Je souhaiterai recevoir des suggestions et critiques pour apporter au fil des versions les améliorations ou corrections souhaitées. Ce DIGI est sous votre propre contrôle et peut être adapté à tous vos besoins. Je souhaite cependant vous rappeler que **je veux rester impérativement très proche de la spécification APRS!** Si vous rencontrez des violations de protocole dans le programme et le fichier .ini par défaut, faites-moi le savoir! C'est très important, car si mon fichier .ini par défaut est caduque, l'erreur va s'étendre dans le monde entier!

Très amicalement,
Henk.

Configuration de AX25_MAC

DIGI_NED utilise un programme résident en mémoire (TSR) qui se charge de piloter les modems, etc... Ce programme est AX25_MAC. MAC est l'abréviation de *Médium Access Control* qui est un acronyme courant en programmation utilisé pour designer la partie de code pilotant le "hardware". Vous aurez toujours besoins de démarrer ce TSR quand vous utiliserez la version DOS de DIGI_NED, sans ce TSR, le programme refusera de se lancer.

AX25_MAC est la couche MAC pour l'AX.25. Ce qu'il fait? Il récupère les trames de données brutes, y ajoute un CRC puis les transmet comme un paquet HDLC vers le "hardware". Les trames lues sont, quand le CRC est correct, passées en format brut au programme qui utilise la couche MAC. La couche MAC est très simple car elle transmet tout ce qui lui est passé.

AX25_MAC doit être chargé en mémoire en utilisant les paramètres utilisés pour le programme TFPCX. Dans le fichier 'run.bat' vous trouverez un exemple pour le modem Baycom 1k2 que j'utilise pour mes essais.

Voici les principaux paramètres:

Utilisation: AX25_MAC [-N] [<options de lancement> | -U]

```
<options générales>                                  <légende>
-N pas de messages                                   [] optionnel
-U déchargement                                      |  alternatif
                          x  hex digit
                          n  dec digit

<options de chargement>
-P<port>[:xxx:nn:nnnn]                 port packet [addr:IRQ:<clock>]
-Bnnnn[:nnnn ...]                      baud rate (1 number/port)

-F[file]     lecture du fichier ini
-D           debug mode
-C[xx]       affiche DCD [couleur]
-Ixx         interruption AX25_MAC
-L           interLock - un TX en même temps (uniquement ports half-duplex)
-BU[nnnn]    nombre de buffers (tampon)
<port>  COMn | LPTn | PARn | YAMn | BPQnn | KISSn | DSCC | OSCC | USCC
(n = 1-4, pour BPQ n=  60-80)
<clock> 0 = désactivé,    2 = hardclock    4 = PA0HZP port  (1 digit/
      1 = softclock  3 = DF9IC modem  5 = PA0HZP timer  channel)
```

Tout ceci peut être également affiché en lançant AX25_MAC de la façon suivante:

```
AX25_MAC -?
```

Les paramètres de configuration et de réglage pour le matériel utilisé sont les mêmes que pour TFPCX. Prenez la documentation de TFPCX273 pour en savoir plus. Vous pouvez trouvez la distribution à: "`http://www.microlet.com/tfpcx`" ou sur le site de NORD><LINK; de même que dans de nombreux autres endroits. Certains paramètres utilisés dans TFPCX ont disparus, ils ne sont d'aucun intérêt pour une couche MAC (il n'y a pas de couche protocole AX.25 dans AX25_MAC).

AX25_MAC peut gérer jusqu'à 8 ports. Le paramètre "`-L`" est utilisé différemment que décrit dans la documentation de TFPCX; il sert ici à s'assurer qu'un seul port transmet à la fois.

Les réglages des paramètres d'accès aux ports (TXDelay et autres) sont configurés au moyen du fichier AX25_MAC.INI. Au démarrage d'AX25_MAC l'option `-F` doit être spécifiée, autrement le fichier n'est pas lu et les valeurs par défaut seront utilisées.

AX25_MAC.INI contient les paramètres qui existent également dans TFPCX, mais seulement les paramètres qui nous sont utiles pour la couche MAC y sont présents. D'autres paramètres qui existaient

dans TFPCX ont disparus.
Pour de plus amples informations reportez-vous au commentaire dans le fichier AX25_MAC.INI.

Voici un exemple qui montre comment configurer AX25_MAC avec un TNC en mode KISS. J'ai reçu plusieurs emails concernant ce sujet, et c'est pourquoi j'ai écris cette courte explication pour montrer la manière à suivre.

Voici ce que vous devez faire:

1. Passez le TNC en mode KISS (la façon de le faire dépend du type de TNC que vous avez).
2. Si nécessaire, modifiez le fichier AX25_MAC.INI pour fixer le TX-Delay de manière à ce qu'il corresponde à votre TRX.
3. Chargez AX25_MAC de la manière suivante:

```
AX25_MAC -PKISS1 -B9600 -L -F -C17 -BU50
```

(vous pouvez modifier le fichier 'run.bat' pour avoir un fichier "batch" de démarrage)

A ce moment vous avez un pilote qui doit fonctionner. Vous pouvez disposer de plusieurs ports; le paramètre '-C17' conditionne l'indicateur que vous pouvez voir dans le coin supérieur droit. Il doit suivre la réception des données. Si vous n'aimez pas ça, enlevez juste le paramètre '-C17'. Le paramètre '-B9600' définit la vitesse à 9600 bauds (par défaut en mode KISS), vous pouvez aussi le retirer. Le paramètre '-L' empêche les transmissions simultanées si vous avez plus d'un port, vous pouvez aussi l'enlever car avec un seul port il ne sert à rien.

Le '-F' force la lecture du fichier AX25_MAC.INI pour programmer le TX-delay, etc... Si vous l'oubliez, les valeurs par défaut seront utilisées.
'-BU50' définit que 50 trames est le maximum de trames pouvant êtres contenues dans le tampon du TSR.

Un démarrage très simple pourrait être:

```
AX25_MAC -PKISS1
```

Je sais, ça ne parait pas très simple au début, mais en utilisant AX25_MAC qui permet le fonctionnement de très nombreux "hardware" permet de rendre DIGI_NED indépendant du matériel.

AX25_MAC peut être déchargé de la mémoire en le relançant à nouveau mais cette fois-ci avec le l'option '-u'.

Maintenant le programme TSR fonctionne, nous pouvons donc passer à la configuration proprement dite de DIGI_NED.

Configuration de DIGI_NED

Une fois que AX25_MAC a été chargé en mémoire nous pouvons démarrer le programme DIGI_NED. Au lancement DIGI_NED lit le fichier DIGI_NED.INI. Un autre fichier peut être lu à sa place, il suffit de définir le nom du fichier au démarrage.

```
C:\APRS\> DIGI_NED MYDIGI.INI
```

La ligne ci-dessus charge MYDIGI.INI au lieu de DIGI_NED.INI. Quand aucun fichier n'est précisé, le programme lira le fichier DIGI_NED.INI se trouvant dans le même répertoire où se trouve le fichier .EXE.

Vous pouvez utiliser l'option '`-v`' pour verbose, ou '`-h`' pour l'aide.

Avant de démarrer DIGI_NED vous devez remplacer mon indicatif dans le fichier DIGI_NED avec l'indicatif que vous souhaitez utiliser pour le digipeater. L'indicatif ne se trouve qu'à un seul et unique endroit; à la fin du fichier DIGI_NED.INI vous trouverez:

```
digi_call: PE1DNN-2
```

Tous les champs DIGI_CALL se trouvant dans le fichier DIGI_NED.INI seront remplacés au lancement par l'indicatif défini à '`digi_call`', dans mon cas c'est PE1DNN-2.

En tant que sysop du digipeater vous devez fournir votre propre indicatif, de cette manière la requête `?id` retournera l'indicatif du sysop du digipeater. De surcroît le sysop pourra exécuter la commande `?exit` quand il le désire. Vous pouvez définir plusieurs sysops. Le premier qui est défini sera utilisé pour la requête `?id`, mais les autres seront capables d'utiliser la commande `?exit` également. Les autres indicatifs peuvent être, votre indicatif avec un SSID différent de même que les indicatifs d'autres co-sysops.

Le texte dans le fichier digibcon.ini sera transmis comme une balise. Actuellement il contient un message APRS avec la position du digipeater. La syntaxe de ce message et comment la définir peut être trouvée dans la spécification APRS. N'oubliez pas de changer la position, j'ai vu des stations très bizarres près de mon QTH ces derniers temps :-). Faites attention car certains caractères ne sont pas autorisés dans la spécification APRS!

Le texte dans le fichier digi_id.ini sera transmis comme identifiant de la station. C'est juste un message texte pour ceux qui écoutent la fréquence avec leur moniteur AX25 en fonction et qui peuvent se demander qu'elle est est la station qui émet toutes les données qu'ils voient à l'écran. N'oubliez pas de modifier ce fichier de manière à ce qu'il contienne votre indicatif.

Dans le fichier DIGI_NED.INI vous trouverez également une commande "`send:`" qui définit l'intervalle d'émission des balises, depuis quel fichier les textes de balises doivent être lus et sur quel port être transmis. De même les destinations et les indicatifs des digipeaters sont spécifiés ici. Quand le fichier de la balise est défini sans son chemin correspondant, alors DIGI_NED cherchera dans le répertoire où se trouve le programme.

Maintenant, voyons ce que fait le digipeater, et comment ?

Dans APRS le *digipeater-path* est constamment manipulé. Une station APRS utilise des indicatifs génériques dans la liste "*via*", comme 'RELAY', 'TRACE', 'WIDE', etc., qui sont interceptés par le digipeater, remplacés par le propre indicatif du digipeater puis retransmis. L'astuce est qu'une station APRS n'a pas besoin de connaître l'indicatif du digipeater qui couvre la zone où elle se trouve. La station a juste à envoyer sa trame avec 'WIDE' dans le chemin "*via*" et tout digipeater de la zone qui répond à l'indicatif 'WIDE' va intercepter la trame; il se peut que plusieurs digipeaters le fassent en même temps.

Le format WIDEn-N nécessite des traitements particuliers qui sont effectués par les digipeaters dits "intelligents" comme DIGI_NED. Quand une station commence avec un digipeater comme WIDE5-5 dans le path *via*, le premier digipeater intelligent qui intercepte cette trame va changer l'indicatif en WIDE5-4 dès qu'il va le retransmettre. Au second digipeater "intelligent", le path *via* va devenir WIDE5-3 et ainsi de suite jusqu'au 5ème digipeater où l'indicatif deviendra WIDE5-0 (en d'autres mots WIDE5).
Ensuite, le "digipeated bit" va être défini (visible dans la plupart des moniteurs par une indication '*') et l'indicatif suivant de la liste "*via*" va être activé.

TRACEn-N fonctionne de manière similaire. Il y a matière à discuter sur ce sujet mais c'est hors du cadre de cette documentation. Le fichier par défaut DIGI_NED permet un digipeating "intelligent" selon notre expérience et nos connaissances. Le comportement qui y est défini a été vérifié auprès du *SIG APRS (Special Interest Group)* à `http://www.tapr.org`.

Regardons un peu comment cela fonctionne en utilisant une règle du fichier DIGI_NED.INI:

```
digipeat: 1 relay 2
```

Cela veut dire: "*digipeat les paquets qui arrive via le port 1 et où le nom du digipeater qui va être activé est 'RELAY' et renvoie les via le port 2*".

Par exemple, une trame rentre via le port 1:

```
from 1: PE1DNN > APRS via PE1MEW-2*, RELAY, WIDE
```

Cette trame a déjà été répétée par PE1MEW-2, le digipeater suivant dans la liste via est RELAY et cela correspond avec l'indicatif dans la règle `digipeat:`;cette trame devrait alors partir par le port 2. Dans la règle de digipeating, rien n'est spécifié après le '2', c'est donc un digipeating 'normal'. Dans le cas présent l'indicatif est substitué par celui du digipeater, c.à.d PE1DNN-2. La trame transmise vers le port 2 devient:

```
to 2: PE1DNN > APRS via PE1MEW-2*, PE1DNN-2*, WIDE
```

C'est ceci qui va être retransmis. L'indication de port '`all`' indique '*depuis tous les ports*' et '*vers tous les ports*'.

Donc:

```
digipeat: all relay all
```

Quand on reçoit la trame par le port 1:

```
from 1: PE1DNN > APRS via PE1MEW-2*, RELAY, WIDE
```

elle est retransmise en suivant cette règle:

```
to 1: PE1DNN > APRS via PE1MEW-2*, PE1DNN-2*, WIDE
to 2: PE1DNN > APRS via PE1MEW-2*, PE1DNN-2*, WIDE
```

La transmission va se faire vers les deux ports de sortie quand il y en a deux (le nombre de ports est conditionné par ce qui est défini dans AX25_MAC).

Vous pouvez effectuer toutes sortes de manipulations dans le *digipeater path*, comme le modifier complètement, ajouter des indicatifs à la liste *via*, etc... Vous pouvez aussi prendre en compte les digipeaters hors service ou créer un 'cluster' basique. Ce que vous pouvez faire est expliqué dans le commentaire du fichier DIGI_NED. Attention, il y a pas mal de possibilités. Votre imagination et vos besoins doivent faire le reste. Le verbose et le logging vous seront très utiles pour comprendre et analyser vos expérimentations.

Si vous voulez en savoir plus lisez ,également le chapitre concernant les FAQ (Questions Fréquemment Posées) à la fin de ce document.

Syntaxe des commandes

Voici une description sommaire de la syntaxe des commandes:

SEND	`send: <time> <to-ports> [<dest>] [<digis>] <filename>`
time	intervalle de temps en minutes
to-ports	<port>[,<port>]... ou "all" pour tous les ports
dest	destination call, par exemple DIGI_DEST, ID ou BEACON
digis	<digi>[,digi]... via digipeater calls for beacon
filename	nom du fichier contenant les données de la balise
digi	indicatif du digipeater, par exemple WIDE ou TRACE6-6
Port	1..(numéro du dernier port actif)
Exemple: `send: 20 all DIGI_DEST,WIDE,TRACE6-6 digibcon.ini`	

DIGIPEAT	`digipeat: <from-ports> <due-digis> <to-ports> [operation[n] [<digis>]]`
from-ports	<port>[,<port>]..., "all" pour tous les ports, "allbut" pour tous les ports sauf le port de réception
due-digis	<due>[,<due>]... liste des "due" digipeaters à intercepter
to-ports	<port>[,<port>]... ou "all" pour tous les ports
operation	add | replace | new | swap | hijack | erase
n	nombre d'indicatifs de digipeaters à marquer comme "utilisé,", 1 par défaut
digis	<digi>[,digi]... liste des indicatifs de digipeaters
port	1..(num,ro du dernier port actif)
due	indicatif du digipeater a activer, par exemple WIDE ou TRACE, supporte les caractères génériques ?, #, @ et *.
digi	digipeater pour opération, par exemple WIDE ou TRACE6-6.
Exemple: `digipeat: all wide7-7 all swap DIGI_CALL,wide7-6`	

DIGIEND	`digiend: <from-ports> <end-digis> <to-ports> [operation[n] [<digis>]]`
from-ports	<port>[,<port>]..., "all" pour tous les ports, "allbut" pour tous les ports sauf le port de réception
end-digis	<end>[,<end>]... liste des "derniers" digipeaters utilisés où une action doit être effectuée
to-ports	<port>[,<port>]... ou "all" pour tous les ports
operation	add | replace | new | swap | hijack | erase
n	nombre d'indicatifs de digipeaters à marquer comme
digis	<digi>[,digi]... list of digipeater calls
port	1..(numéro du dernier port actif)
end	digipeater dernièrement utilisé, par exemple WIDE ou TRACE, supporte les caractères génériques ?, #, @ et *.
digi	digipeater pour opération, par exemple WIDE ou TRACE6-6.
Exemple: `digiend: all wide*,trace* 2 add LOCAL`	

DIGITO	`digito: <from-ports> <destinations> <to-ports> <ssid> [operation[n] [<digis>]]`
from-ports	<port>[,<port>]..., "all" pour tous les ports, "allbut" pour tous les ports sauf le port de réception
destinations	<dest>[,<dest>]... liste des indicatifs destination intercepter
to-ports	<port>[,<port>]... ou "all" pour tous les ports
operation	add | replace | new | swap | hijack | erase
n	nombre de digipeaters à marquer "utilisés", défaut 1
digis	<digi>[,digi]... liste des digipeaters

port	1..(numéro du dernier port actif)
dest	indicatif destination, ex. APRS ou BEACON, supporte les caractères génériques ?, #, @ et *.
digi	paramètre opération du digipeater, par exemple WIDE ou TRACE6-6.
Exemple: `digito: 1 *-12 all 0 add WIDE`	

PREEMPT	`preempt: <from-ports> <on-digis> [<replace>]`
from-ports	<port>[,<port>]..., "all" pour tous les ports
on-digis	<find>[,<find>]... liste des digis à rechercher pour agir dessus, supporte les caractères génériques ?, #, @ et *.
find	digipeater à trouver dans la liste "via". Une fois trouvé, l'opération préemptive se déclenche.
Replace	digipeater de remplacement; va remplacer le digipeater trouvé. Si cela est non spécifié,, le digipeater est laissé inchangé.
Exemple: `preempt: all PE1DNN-2 WIDE`	

PREEMPT KEEP	`preempt_keep: <digis>`
digis	<keep>[,<keep>]... liste des digipeaters à garder lorsque l'opération préemptive se déclenche sur la trame, supporte les caractères génériques ?, #, @ et *.
keep	digipeater à conserver dans la liste "via" quand la trame subit l'opération préemptive.
Exemple: `preempt_keep: PA*,PE*,PD*,PI*`	

PREEMPT NEVER KEEP	`preempt_never_keep: <digis>`
digis	<keep>[,<keep>]... liste des digipeaters qui ne doivent jamais êtres gardés lors de l'opération préemptive sur la trame; supporte les caractères génériques ?, #, @ et *.
keep	digipeater … conserver dans la liste "via" lorsque la trame subit l'opération préemptive.
Exemple: `preempt_never_keep: RELAY*,WIDE*,TRACE*,GATE*`	

LOCAL	`local: <local-ports>`
local-ports	<port>[,<port>]... ou "all" pour tous les ports
Exemple: `local: 2`	

SIZE HEARD LIST	`size_heard_list: <nombre>`
number	nombre maximum d'indicatifs directs qui seront conservés dans la liste "mheard" (cumul de tous les ports)
Exemple: size_heard_list: 40	

KEEP TIME	`keep_time: <number>`
number	nombre de secondes à mémoriser pour les données déjà digipeatées
Exemple: `keep_time: 300`	

SHORT KEEP TIME	`short_keep_time: <number>`
number	nombre de secondes à mémoriser pour les données déjà digipeatées qui correspondent au "data_prefix:", voir ci-dessous
Exemple: `short_keep_time: 10`	

DATA PREFIX	`data_prefix: <char>[<char>]...`
char	caractère à comparer avec le premier caractère reçu pour déterminer combien de temps mémoriser cette donnée si elle est digipeatée.
Exemple: `data_prefix: :}?`	

MESSAGE FILE	`message_file: <nomdefichier>`
filename	nom du fichier où le programme doit lire les requêtes de messages ainsi que les réponses
Exemple: `message_file: digi_ned.mes`	

MESSAGE KEEP TIME	`message_keep_time: <nombre>`
number	nombre de secondes à mémoriser des messages envoyés au digipeater (requêtes) pour éviter les doubles réponses à la même requête.
Exemple: `message_keep_time: 900`	

MESSAGE PATH	`message_path: <to-ports> <digis>`
to-ports	<port>[,<port>]... ou "all" pour tous les ports
digis	<digi>[,digi]... via digipeater pour les réponses
digi	digipeater, par exemple WIDE ou TRACE6-6.
Port	1..(numéro du dernier port actif)
Exemple: `message_path: all TRACE,WIDE`	

MAX MSG HOPS	`max_msg_hops: <hopcount>`
hopcount	1..(nombre de hops)
Exemple: `max_msg_hops: 2`	

BLOCK	`block: <call>[,<call>]...`
call	indicatif source dont aucune trame n'est acceptée
Exemple: `block: N0CALL,NOCALL,MYCALL`	

MSG BLOCK	`msg_block: <call>[,<call>]...`
call	indicatif source dont aucune requête n'est acceptée
Exemple: `msg_block: N0CALL,NOCALL,MYCALL`	

DIGI OWNER	`digi_owner: <call>[,<call>]...`
call	indicatif du « propriétaire » du digi
Ceci doit être spécifié de façon permanente dans le fichier .ini et doit contenir au moins un indicatif.	
Exemple: `digi_owner: PE1DNN,PE1DNN-7,PE1MEW`	

ENABLE EXIT	`enable_exit: 0 | 1`
0	interdit la fonction d'arrêt à distance
1	autorise la fonction d'arrêt à distance (seulement pour le propriétaire du digi!)
Exemple: `enable_exit: 1`	

LOGFILE	`logfile: [<filename>]`
filename	nom du fichier pour écrire les données de log; si aucun nom n'est précisé il n'y a pas de logging
Exemple: `logfile: digi_ned.log`	
DIGI CALL	`digi_call: <call>`
call	indicatif du digipeater
Ceci doit être défini de façon permanente dans le fichier .ini.	
Exemple: `digi_call: PE1DNN-2`	

DIGI DEST	`digi_dest: <call>`
call	indicatif destination utilisé pour les balises et les messages.
Ceci doit être défini de façon permanente dans le fichier .ini.	
Exemple: `digi_dest: APZ017`	

Les Requêtes

Que peut-on donc faire avec les requêtes?

Vous pouvez posez des questions à DIGI_NED grâce aux messages APRS. C'est une fonction similaire aux Tiny-Web-Pages proposées par WB4APR.

Dans DIGI_NED, le mécanisme des requêtes fonctionne avec des messages APRS standards. Pour commencer, DIGI_NED répond au message diffusé ?APRS?; DIGI_NED va transmettre toutes ses balises. Tous les autres messages doivent être adressés à DIGI_NED. La plupart des réponses doivent êtres reçues avec accusé de réception par la station effectuant la requête.

DIGI_NED répète les réponses jusqu'à 10 fois, en doublant l'intervalle à chaque tentative. Bien sur, ces retransmissions cessent dès qu'un accusé réception a été reçu.

Voici les commandes possibles:

`?help`	affiche un résumé de toutes les commandes
`?id`	affiche l'indicatif du digipeater et celui du sysop
`?ver`	affiche la version et la date de compilation
`?up`	affiche la date et l'heure du dernier redémarrage
`?type`	affiche le type DIGI_NED pour le digipeater
`?ports`	affiche le nombre de ports disponibles
`?aprsd`	affiche les stations entendues en direct; max 5; pas d'accusé réception
`?mheard`	affiche l'aide de la commande mheard
`?mheard 1`	affiche les stations entendues sur le port 1
`?mheard pe1dnn`	affiche quand pe1dnn a été entendu la dernière fois, avec le numéro du port
`?mh`	identique à mheard, juste une commande plus courte
`?ping?`	affiche le chemin de requête vers le digipeater, pas d'accusé réception
`?exit`	stoppe DIGI_NED à distance (seulement le sysop peut faire cela et cette fonction doit être autorisée dans le fichier digi_ned.ini)
`?exit 12`	identique à ?exit, mais ici exit-code 12 en plus
`?aprsm`	retransmet tous les messages accusés pour le demandeur
`?aprst`	identique à ?ping?

Le caractère '?' n'est pas nécessaire mais il est ici supporté pour garder une totale compatibilité avec la spécification APRS. Il peut y avoir bien d'autres messages. Tous les messages qui retournent un texte statique sont stockés dans un fichier que vous pouvez modifier selon vos besoins. Ce fichier est défini par "message_file:" dans digi_ned.ini, par défaut c'est digi_ned.mes. la structure du fichier digi_ned.mes est expliquée dans le fichier exemple livré avec DIGI_NED.

Notez que certaines commandes provoquent la transmission de balises ou la transmission d'objets au lieu de retourner un message.
Une commande comme "`?aprsm`" ne retourne rien si il n'y a aucun message en attente pour vous. "`?ping`" et "`?aprst`" envoient des messages qui ne nécessitent pas d'accusé réception. Si la réception échoue, vous ne verrez aucune réponse.

Une commande spécifique émanant d'un utilisateur est seulement acceptée une seule fois en "`message_keep_time:`" secondes, par défaut 900 secondes.
Quand un utilisateur envoi la même commande dans cet intervalle de temps DIGI_NED ne répondra pas, il accusera réception du message uniquement.

Ce la veut dire par exemple qu'un utilisateur ne peut envoyer deux commandes "`?info`" à DIGI_NED dans cet intervalle de temps; au deuxième envoi DIGI_NED ne répondra pas, mais accusera seulement réception du message. Si l'utilisateur essaye à nouveau la commande "`?info`" après 900 secondes alors il recevra une réponse normale à "`?info`". Après avoir envoyé une commande "`?info`" il peut envoyer une autre commande sans restriction, par exemple "`?up`" qui fonctionnera normalement. Une seconde commande "`?up`" ne renverra rien.

La raison de tout ceci est simplement d'éviter que deux systèmes de réponse automatiques ne commencent à se répondre mutuellement et de façon infinie. Avec ceci, il y aura qu'un seul "ping-pong". Si 15mn ne vous

semblent pas assez chez vous, vous pouvez augmenter la valeur de "`message_keep_time:`".

Il existe une façon de contourner une seule fois ceci. Si vous voulez réellement exécuter la même commande vous pouvez ajouter ou omettre le '?' en début de commande, le filtrage de doublon de DIGI_NED verra ici deux trames différentes, bien que la réponse soit identique avec ou sans ce caractère. Dans l'exemple précédent vous pouvez utiliser "info" et recevoir une réponse si la première requête a été faite avec "`?info`". Notez que le Kenwood TH-D7 supprime les réponses doublons, et vous ne pourrez pas voir la réponse si vous l'avez déjà reçue!

La commande `?exit` est faite pour être utilisée par le sysop du digipeater. La commande "`?exit`" a quelques restrictions; les deux conditions qui suivent doivent êtres vraies de manière à permettre à la commande de fonctionner:

1) La commande doit être validée dans DIGI_NED.INI
2) Le message doit être émis par le propriétaire du DIGI_NED

Quand une seule de ces conditions n'est pas vraie alors DIGI_NED enverra le texte d'aide par défaut.

Après réception et validation de la commande `?exit`, le digipeater va envoyer un message "shutdown" sans aucun digipeater dans le path (le destinataire doit être local et recevoir le digipeater en direct). Quand ce message a reçu un accusé réception le digipeater s'arrête. Le message "shutdown" est répété jusqu'à 3 fois; si un 'ack' n'a toujours pas été reçu après la troisième tentative alors le digipeater repasse en mode de fonctionnement normal comme s'il n'avait jamais reçu de commande `?exit`.

DIGI_NED utilise des nombres comme codes de sortie qui peuvent être utilisés dans un fichier .bat. Un exemple de ceci peut être trouvé dans le fichier:

"run.bat".

Les codes de sortie sont:

-1	erreur au démarrage (AX25_MAC non chargé, paramètres obligatoires dans DIGI_NED non renseignés!)
0	ok, sortie normale (par le clavier, ALT+X)
1	sortie avec code de test (seulement pour le debugage)
2	sortie télécommandée à distance par ?exit

On peut aussi éventuellement utiliser un nombre entre 0 et 255 avec la commande `?exit`; par exemple "`?exit 10`". Dans ce cas la valeur fournie sera utilisée comme code de sortie. De cette façon n'importe quel programme peut être lancé avec un fichier batch après avoir stoppé le digipeater.
Regardez dans "run.bat" il y a un exemple qui utilise les codes de sortie standards. On peut utiliser ceci pour lancer d'autres programmes comme NetCHL qui permet de faire la maintenance du système sur lequel tourne DIGI_NED et ajouter de nouveaux paramètres ou programmes.

Pour Linux cela fonctionne également, car sous Linux vous pouvez lancer plusieurs programmes en parallèle sans problèmes, même si ce n'est pas vraiment nécessaire, mais ça marche.

Les paths utilisés pour les réponses et les acks sont définis dans le fichier DIGI_NED.INI. Pour chaque port on peut définir un "`message_path:`"; 'all' peut être choisi pour utiliser le même path pour tous les ports. Les ports peuvent êtres aussi combinés; par ex. "`message_path: 1,3 WIDE, TRACE6-6`" défini le path pour les ports 1 et 3.

Définir un "`message_path:`" pour chaque port permet de transmettre des réponses à travers différents paths. Il est même possible de définir plus d'un path pour un port, cela se traduit clairement par beaucoup plus de trafic! Cela est nécessaire quand un digipeater du path à travers lequel la réponse doit être transmise n'est pas un digipeater APRS; l'indicatif exact de la station est ici nécessaire. Quand un message doit être émis à travers 2 digipeaters de ce type, 2 paths différent sont nécessaires.
Ceci s'oppose au concept du "digipeater générique", mais reste possible.

Les commandes-clavier de DIGI_NED

Quand DIGI_NED fonctionne vous pouvez utiliser la combinaison de touches ALT-X pour terminer le programme.

ALT-V active/désactive le mode 'verbose'. Vous pouvez passer en shell DOS avec ALT-D (uniquement si la variable d'environnement COMSPEC est définie correctement).

Au clavier vous pouvez aussi effectuer les mêmes requêtes que vous pouvez faire en utilisant les messages APRS. Ainsi, si vous tapez "`help`" vous obtiendrez le texte d'aide. Si vous êtes en mode *verbose* vous pouvez le désactiver avec ALT-V de manière à pouvoir visualiser les réponses. Quand vous avez terminé vous pouvez réactiver le mode verbose par ALT-V.

Quand vous effectuez le logging vers un fichier de log, vous pouvez le stopper avec ALT-L; Le fichier de log sera fermé pour vous permettre en passant en shell DOS de le détruire, ou encore le renommer sans problèmes.
En appuyant sur ALT-L à nouveau, le logging va recommencer, les nouvelles données seront ajoutées à celles existant déjà dans ce fichier. Quand vous arrêtez/relancez le logging, un marquage date/heure apparaît avec un séparateur dans ce fichier.

Sous Linux tout ceci ne marche pas, mais Linux n'a pas besoin de ceci car vous pouvez lancer ne nombreux processus en simultané et vous pouvez faire des requêtes au travers d'une connexions en boucle interne, par exemple avec XASTIR (c'est ce que je fais pour mes essais).

Utilisation en « local »

Un digipeater sur le réseau APRS peut apparaître et disparaître. Des problèmes surgissent quand trop de digipeater non évolués se trouvent dans une zone peu étendue. Ceci arrive facilement dans les agglomérations à cause de la densité de population et le nombre élevé de zones radio non couvertes, la faute aux gros immeubles par exemple. Ces constructions bloquent l'accès direct au plus proche WIDE.

Une station dans cette zone a besoin d'un digipeater de voisinage qui va lui permettre de relayer ses trames vers le WIDE et inversement relayer les transmissions du WIDE vers cette station.

Les signaux émanant du WIDE qui sont répétés par ce relais intermédiaire n'ont pas besoin d'êtres émis une fois de plus par d'autres digipeaters; en effet d'autres digipeaters peuvent intercepter ces trames directement en provenance du WIDE. A l'opposé, les signaux émanant de la station locale doivent êtres réémis; au moins par le WIDE!

Avec DIGI_NED, un port "`local:`" peut être défini. Les données qui sont envoyées directement au digipeater seront digipeatées normalement sur le port local; il n'existe pas de différence avec les ports non-locaux. Les données qui ne sont pas envoyées en direct vers DIGI_NED mais qui passent d'abord à travers un autre digipeater, par exemple notre WIDE ci-dessus, seront transmises sur le port local avec une liste "*via*" tronquée; les digipeaters de la liste "*via*" qui ne sont pas "utilisés" sont effacés. De cette manière un autre digipeater ne traitera pas la trame digipeatée (à moins que les autres digipeaters n'aient une règle "`digiend:`" pour intercepter à nouveau les données).

Pour les digipeaters intelligents ceci n'est pas vraiment nécessaire; les données du WIDE seront aussi reçues par les autres digipeaters du voisinage et quand un digipeater retransmet un message les autres ne le retransmettront pas car la trame a déjà été traitée auparavant. Quand les digipeaters de la région ne sont pas intelligents ils vont répéter la trame une fois de plus. Ceci va provoquer du trafic supplémentaire et inutile. Dans ce cas le port de DIGI_NED sur lequel les données sont transmises peut être défini "`local:`", ainsi il n'y aura aucun indicatif de digipeater restant par lequel un digipeater de voisinage pourrait à nouveau réagir.

LINUX

La version pour LINUX est quasiment identique à la version DOS. Sous Linux AX25_MAC n'est pas nécessaire, car cette fonctionnalité est incluse dans le kernel Linux. Ici, il faut utiliser l'option '-p' pour définir quels ports AX25 doivent êtres utilisés. Les ports possibles sont définis dans le fichier `/etc/ax25/axports`, reportez-vous à AX25-HOWTO pour de plus amples informations. Vous devez compiler la version Linux vous-même. Le makefile est configuré pour utiliser les anciens outils-kernel pour les kernels 2.0.36. Pour l'utiliser avec les nouveaux outils-kernels vous devez modifier le Makefile; enlevez le '#' dans la définition de "`DEFS = -DNEW_AX25`".

Compiler DIGI_NED se fait de façon courante sous Linux:

"`make depend`"

suivi par

"`make`".

Vous obtenez alors un exécutable "digi_ned". Cet exécutable utilise comme la version DOS un fichier `digi_ned.ini`; le même fichier que sous DOS peut être utilisé. Les caractères '\' dans les chemins de fichier seront interprétés '/' sous Linux (et'/' sera interprété '\" sous DOS...).

Le port '1' est lié au premier port défini avec l'option '-p', le port '2' est lié au second port défini, etc... On peut utiliser jusqu'à 8 ports. Voici un exemple pour démarrer:

```
./digi_ned -p 1k2 -p lfbb -v
```

Dans cet exemple "`1k2`" sera le port 1 et "`lfbb`" le port 2. L'option '`-v`' permet l'affichage du *verbose*. DIGI_NED fonctionne uniquement avec les interfaces kernel; en les utilisant vous pouvez connecter quasiment n'importe quoi; lisez le fichier AX25-HOWTO pour savoir comment faire.

Notez bien que DIGI_NED doit tourner avec les privilèges 'root' pour être capable d'écouter le socket pour les paquets qui arrivent, tout comme 'listen', 'net2kiss' et d'autres programmes qui écoutent du trafic non-connecté sur un socket.

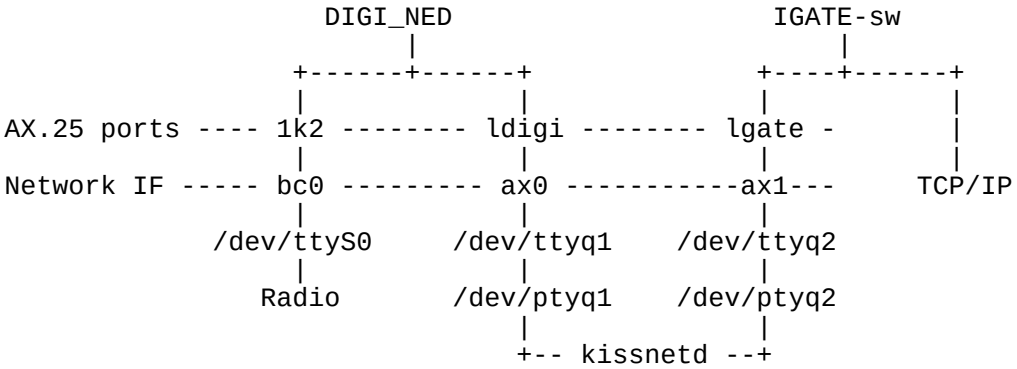
Pour sortir du programme sous Linux il faut envoyer le signal SIGINT au programme. Normalement ce signal est généré par Ctrl-C, mais cela n'est pas utile. Pour plus d'informations reportez-vous à l'aide de "`stty`". Un simple script de shell vous est fourni pour démarrer DIGI_NED en tâche de fond.

IGATE

Plusieurs personnes m'ont demandé si DIGI_NED pouvait fonctionner comme IGATE. Comme DIGI_NED est bâti pour DOS et que Internet avec DOS n'est pas facile à gérer, DIGI_NED ne peut fonctionner comme IGATE. Mais cela ne veut pas dire que vous ne pouvez pas utiliser DIGI_NED avec une IGATE sous Linux. Linux - comme tout Unix - suit la philosophie d'Unix, c.à.d avoir des petits blocs fonctionnels capables de fonctions simples très précises (je n'aime pas les gros programmes monolithique sous Linux pour cette raison).

En utilisant ces divers blocs fonctionnels vous pouvez aisément construire une connexion "loopback" avec des pseudo "`ttys`". D'un côté vous pouvez connecter DIGI_NED, de l'autre côté le programme IGATE.

Regardons le schéma ci-dessous pour mieux comprendre:



Voici comment tout ceci est configuré:

En premier nous avons le module KISS chargé (kissnetd créé des appareils KISS):

```
echo 'Starting AX25 loopback drivers'
modprobe mkiss
```

Ensuite on démarre "kissnetd" sur 2 pseudo ttys en tâche de fond:

```
/usr/sbin/kissnetd /dev/ptyq1 /dev/ptyq2 &
sleep 4
```

On attend que "`kissnetd`" soit démarré, qui prend l'autre extrémité des pseudo ttys - qui fonctionne en KISS - et créé deux interfaces réseau qui sont liées à deux ports ax25. Dans ma configuration cela crée les interfaces `ax0` et `ax1`.

```
/usr/sbin/kissattach /dev/ttyq1 -l ldigi
/usr/sbin/kissattach /dev/ttyq2 -l lgate
```

Maintenant je configuré ax0 et ax1 et les déclare "up".

```
      /sbin/ifconfig ax0 10.0.0.1 netmask 255.255.255.255 \ mtu 256 hw ax25
PE1DNN-7 up
      /sbin/ifconfig ax1 10.0.0.2 netmask 255.255.255.255 \ mtu 256 hw ax25
PE1DNN-8 up
```

J'ai donc maintenant 2 ports ax25, "`ldigi`" et "`lgate`" qui sont connectés l'un à l'autre. Vous pouvez maintenant lancer DIGI_NED sur "`ldigi`" et vous avez le programme IGATE qui prend les trames sur "`lgate`" et qui les envoie sur l'Internet. Les trames venant d'Internet sont envoyées sur "`lgate`" et entrent dans DIGI_NED via "`ldigi`".
DIGI_NED va digipeater les trames du "`1k2`" (c.à.d de la radio via un modem Baycom) vers "`ldigi`" et les trames venant de "`ldigi`" seront digipeatées par DIGI_NED vers "`1k2`" et transmises de cette manière. Cela fonctionne parfaitement.

Le fichier exemple `/etc/ax25/axports` pour cette configuration est:

```
# name      callsign    speed packlen     window      description
1k2         PE1DNN-7    0     250         4           BayCom sur Radio
ldigi       PE1DNN-8    0     250         4           Entrée pour DIGI_NED
lgate       PE1DNN-9    0     250         4           Entrée pour IGATE
```

Il y a tant de choses possibles sous Linux, et c'est pourquoi je ne crois pas qu'il soit nécessaire de construire un IGATE dans DIGI_NED. Il existe de nombreux blocs fonctionnels qui permettent d'accomplir ce que vous souhaitez. J'espère que ceci vous permettra de comprendre comment configurer DIGI_NED pour qu'il fonctionne avec un programme IGATE.

FAQ

QUESTION	Ma balise s'affiche au mauvais endroit, c'est un bug?
REPONSE	DIGI_NED ne fait absolument rien sur le texte de la balise, il ne fait que l'envoyer. Peut-être avez-vous mal entré la position - les degrés et minutes sont en base 60, mais les secondes sont en base 100. C'est à dire, si votre position est 40°29'15 N, 105°04'54 O elle devra être rentrée `4029.25` et `10504.90` dans le fichier de balise. Pour cette position ça donne quelquechose comme: `=4029.25NN10504.90E#PHG2110/DIGI_NED: N0ABC-2`

QUESTION	Dans le fichier .ini je vois la ligne: `digipeat: all DIGI_CALL all` Dois-je remplacer '`DIGI_CALL`' par l'indicatif de ma station DIGI_NED ou est-ce que le programme s'en occupe?

REPONSE	C'est le programme qui s'en charge, c'est remplacé par ce qui est défini par:
	
	`digi_call: n0abc-2`
	
	Ce qui équivaut à:
	
	`digipeat: all n0abc-2 all`

QUESTION	Voici un fragment de capture d'écran de DIGI_NED.
	
	*****début*****
	`from:1 KF0ZH > APR819 via GATE* FTCOL* UIv PID=F0 68 bytes`
	`@291427z4447.06N/09329.48W_267/002g001t054r000p000....h97b10`
	`173dU2k`
	`==========`
	`DigiEnd: Digipeater last done 'FTCOL', hop 2`
	`No matching rule: ignore frame`
	*****fin*****
	
	FTCOL est un digipeater local avec comme indicatif tactique FTCOL,
	l'autre digi est CSU (Colorado State University).
	
	Apparemment DIGI_NED semble ne pas traiter les paquets qui ont FTCOL
	dans le path, est-ce correct?

REPONSE	Voir page suivante

REPONSE	Comme vous pouvez remarquer par le '* ' après chaque indicatif de digipeater, tous les digipeaters tout au long du path ont déjà traité le paquet. Le paquet est maintenant au bout de son trajet.
	
	Dans ce cas DIGI_NED va chercher une règle de type 'digiend:'. DigiEnd est utilisé pour intercepter les paquets qui ont des digipeaters non encore utilisés. La règle s'applique sur le dernier digipeater de la chaîne. Dans ce cas il n'y a plus de digipeater non-utilisé (ils ont tous un marqueur '*') et DIGI_NED va chercher une règle avec un indicatif qui correspond à 'FTCOL'. Il n'en trouve aucune, car il n'y en a pas. Il n'y a seulement que quelques paquets dans le fichier .ini par défaut qui sont traités à nouveau après avoir atteint leur destination finale.
	J'ai un exemple dans le fichier .ini que j'utilise pour ,mettre les paquets vers une autre QRG.
	
	Le voici:
	
	`digiend: all wide*,trace* 2 add LOCAL`
	`digiend: 1 pe1mew-2 2`
	
	La première ligne intercepte tous les paquets reçus sur n'importe quel port où le dernier indicatif est WIDE ou TRACE ou WIDEn-N ou TRACEn-N (le '*' est un caractère générique). J'y ajoute l'opération 'add' qui veut dire que l'on ajoute en premier lieu l'indicatif du digipeater, dans notre cas c'est LOCAL qui est ajouté au path des digipeaters. Ce paquet modifié est envoyé à travers le port 2, qui est dans mon cas une QRG locale sur 70cm.
	
	La seconde ligne intercepte tous les paquets reçus sur le port 1 et qui finissent leur trajet à notre digipeater local. Il n'y a pas d'opération supplémentaire; donc on exécute le comportement par défaut qui ajoute le propre indicatif du digipeater et envoie les paquets à travers le port 2.
	
	Donc, si vous voulez intercepter les paquets qui finissent à FTCOL et les répéter, vous devez ajouter une règle. Normalement les paquets qui ont terminé leur trajet doivent être laissés de côté, mais une raison de les traiter à nouveau serait par exemple de couvrir une zone mal couverte, ou les envoyer vers une autre QRG.
	
	Voici un exemple qui effectue ce genre d'opération:
	
	`digiend: 1 FTCOL 1`
	
	Ici on prend les paquets sur le port 1 où le dernier digipeater est FTCOL, on ajoute l'indicatif de votre digipeater (comportement par défaut) et on les renvoie par le port 1.
	
	Voici le genre de trame que vous devriez voir:
	
	`to:1 KF0ZH > APR819 via GATE* FTCOL* N0EQ-2* UIv PID=F0 68 bytes`
	`@291427z4447.06N/09329.48W_267/002g001t054r000p000....h97b10173dU2k`
	
	Vous pouvez faire la même chose pour CSU:
	
	`digiend: 1 CSU 1`
	
	Bien évidemment, vous devez jugez par vous-même si ces paquets doivent être répétés et s'ils ne provoquent pas des boucles.

QUESTION	Est-ce que je comprends bien ce qui suit: `digipeat: all DIGI_CALL all` Tous les paquets reçus par mon DIGI_NED seront digipeatés et auront l'indicatif de ma station DIGI_NED inséré dans les paquets. Est-ce correct?

REPONSE	Non. Supposons que "`digi_call`" est défini: N0ABS-2.
	
	La règle dit: *Les paquets de tous les ports où le digipeater de la liste via qui doit être utilisé est N0ABC-2, seront renvoyés à nouveau vers tous les ports*. Il n'y a pas d'autre règle donc nous ferons un digipeating normal. L'indicatif du digipeater dans le path via qui doit être utilisé est remplacé par l'indicatif du digipeater (dans ce cas N0ABC-2 est remplacé par N0ABC-2 - pas d'effet) puis ensuite le paquet modifié est envoyé à travers tous les ports (vous pouvez avoir jusqu'à 8 ports).
	
	Vous pouvez aussi explicitement prendre les paquets d'un port et les renvoyer à travers un autre ports. Exemple:
	
	`digipeat: 1 relay 2`
	
	Ce qui veut dire: les paquets reçus par le port 1 où 'relay' est le digipeater suivant dans le path via doivent être renvoyés à travers le port 2. Puisqu'il n'y a pas d'action supplémentaire, c'est un digipeating normal. Cela veut dire que, 'RELAY' est changé en '`N0ABC-2`' et le paquet modifié est renvoyé à travers le port 2.
	
	Si vous recevez ce qui suit sur le port 1:
	
	`W0ABC>APRS,GATE*,RELAY,WIDE`
	
	Alors le second indicatif 'RELAY' est le prochain digipeater a entrer en action (l'indicatif GATE est marqué d'un '*', donc déjà passé). Relay est modifié en votre propre indicatif, soit:
	
	`W0ABC>APRS,GATE*,N0ABC-2*,WIDE`
	
	soit le path transmis sur le port 2. Notez que votre propre indicatif a un '*' car ce 'hop' dans la liste via est maintenant traité. Le prochain digipeater qui entrera en action sera 'WIDE'.

QUESTION	Comment puis-je empêcher la transmission de 'paths' incorrects? Supposons que je reçoive un paquet d'une station dont le chemin UI est RELAY, RELAY, RELAY, WIDE, WIDE, WIDE, c'est à dire clairement un path incorrect. Qu'est-ce que je vais modifier (et comment) dans le fichier ini pour convertir ceci en un path nettement plus acceptable?

REPONSE	Cela dépend du chemin déjà parcouru par le paquet dans la chaîne, donc l'indicatif du digipeater suivant le symbole '*'. Si il n'y a pas marqueur '*' sur n'importe lequel des indicatifs vous êtes le premier à le recevoir. Supposons que ce soit effectivement le cas.
	
	Dans le fichier digi_ned.ini par défaut, "RELAY" n'effectue qu'un digipeating normal. C'est ce qui se passe dans votre cas:
	
	entrée:
	`PE1DNN>APRS,RELAY,RELAY,RELAY,WIDE,WIDE,WIDE`
	règle:
	`digipeat all RELAY all`
	sortie:
	`PE1DNN>APRS,PE1DNN-2*,RELAY,RELAY,WIDE,WIDE,WIDE`
	
	Maintenant vous pouvez rendre ceci plus correct si vous avez des stations qui maîtrisent mal le concept des paths dans votre région. Vous savez que 'RELAY' doit seulement être utilisé une et une seule fois dans un path: en tant que premier digipeater de la liste. Maintenant vous pouvez forcer un path sur toutes les stations qui utilisent "RELAY" en redéfinissant le path dans le digi.
	
	entrée:
	`PE1DNN>APRS,RELAY,RELAY,RELAY,WIDE,WIDE,WIDE`
	règle:
	`digipeat all RELAY all new WIDE3-3`
	sortie:
	`PE1DNN>APRS,PE1DNN-2*,WIDE3-3`
	
	Désormais toutes les stations qui sont répétées à travers vous en utilisant "RELAY" auront le même path, qui est ici votre propre indicatif suivi de WIDE3-3. Donc peu importe ce que les utilisateurs configurent, car après avoir été réémis par votre digipeater, le path sera "réparé".
	
	Réparer les paths ainsi est quelquechose qui doit seulement être fait à la source. Quand vous recevez cette trame:
	
	entrée:
	`PE1DNN>APRS,RELAY,RELAY,RELAY,WIDE,WIDE*,WIDE`
	
	Vous ne pouvez plus rien faire pour la réparer, car elle a déjà presque atteint la fin du trajet.
	
	Ce que vous pouvez faire est par exemple éviter les longs paths de digipeaters en remplaçant TRACEn-N avec WIDEn-N.
	
	entrée:
	`PE1DNN>APRS,RELAY*,PE1MEW-2,TRACE5-4`
	règle:
	`digipeat: all TRACE5-4 all swap0 wide5-3`
	sortie:
	`PE1DNN>APRS,RELAY*,PE1MEW-2,WIDE5-3`
	
	Maintenant il n'y a plus de tracing, en maintenant le path court et en l'empêchant de grandir plus.
	
	Voilà les choses que vous pouvez faire. Vous ne pouvez pas écrire des règles qui disent par exemple:
	'*si RELAY apparaît plus d'1 fois ...*',
	S'il existe un besoin d'effectuer cela je pourrai peut-être l'ajouter, mais cela rendra le système beaucoup plus complexe et difficile pour comprendre son fonctionnement.
	
	Actuellement, et il est important de le comprendre, DIGI_NED agit sur:
	
	le répéteur ciblé grâce aux règles '`digipeat:`'.
	'`digiend:`' agit sur le dernier digipeater du path quand ils ont tous été utilisés
	'`digito:`' agit sur l'indicatif de destination quand il n'y a plus de digipeater dans la chaîne.

QUESTION	Par défaut, en utilisant le(s) fichier(s) ini fourni(s), est-ce que DIGI_NED digipeat tous les paquets?
REPONSE	En fait je pense (et espère) que j'ai couvert tous les indicatifs génériques de l'APRS. Si vous voulez digipeater simplement tout, une règle simple serait: `digipeat: all * all` Mais ceci effectue la substitution d'indicatif sur tous et ce n'est plus du digipeating intelligent.

QUESTION	Est-ce que je peux changer l'adresse "`digi_dest: APZ016`" ?
REPONSE	Oui, sans aucun problème. J'ai choisi celle-ci car elle est en accord avec la spécification APRS, c.à.d le "Numéro de version Logiciel APRS" pour les programmes en cours d'expérimentation (programme qui n'a pas encore reçu de préfixe). Le 017 indique version 0.1.7, et qui sera bien sur différent dans les versions à venir. Vous pouvez la modifier en "APRS" si vous le souhaitez. Pour d'obscures raisons, certaines stations locales ici préfèrent avoir leur suffixe suivre le préfixe AP, pour moi ce serait "APDNN". Je ne vois pas d'intérêt particulier à cela, mais je ne vous empêche pas de la faire.

QUESTION	Peut-on faire de la maintenance à distance sur DIGI_NED?
REPONSE	En tant que sysop vous pouvez envoyer une commande ?exit pour arrêter DIGI_NED (si cette fonctionnalité a été activée dans le fichier digi_ned.ini). Sur notre digipeater local nous démarrons NetCHL après avoir stoppé DIGI_NED. Avec TCP/IP nous pouvons télécharger les nouvelles versions, les nouvelles configurations, etc... Après avoir arrêté NetCHL le digipeater est redémarré. Cela permet la maintenance du digipeater quand vous ne pouvez pas y accéder physiquement aisément. L'utilisation d'outils externes comme NetCHL vous permet plus de contrôle et de pouvoir sur le système, plus que je n'ai pu programmer dans le digipeater lui-même. Sous Linux vous pouvez bien sur faire beaucoup plus car vous pouvez lancer plusieurs programmes en parallèle.

QUESTION	Peut-on se ?ping? soit-même ou forcer l'affichage d'une balise?
REPONSE	Oui, vous pouvez envoyer `?APRS?` en tant que message qui se comporte de la même manière que le broadcast `?APRS?` en envoyant toutes les balises. Si vous utilisez la version DOS vous pouvez taper `?aprs?<enter>` au clavier pour forcer une balise.

QUESTION	Puis-je utiliser les requêtes localement avec le clavier?
REPONSE	Oui pour la version DOS, pas sous Linux. Vous pouvez entrer n'importe quelle requête par le clavier en DOS. DIGI_NED utilisera une source "KEYBOARD" virtuelle et une destination virtuelle "SCREEN". Si vous ne pouvez voir ce que vous tapez ou ne voyez pas les réponses c'est que le logging verbose fait défiler le texte hors de l'écran; vous pouvez utiliser ALT-V pour stopper les messages verbose. Les requêtes au clavier n'entraînent pas de transmissions radio, sauf `?APRS?` qui provoque l'émission des balises sur le premier port. La version Linux n'a pas besoin de tout cela, vous pouvez utilisez un loopback local et XASTIR pour effectuer une requête vers le digi. Linux est un environnement multitâches/ multi-utilisateurs où vous pouvez lancer d'autres programmes quand DIGI_NED fonctionne.

QUESTION	Puis-je créer de nouvelles requêtes/réponses?
REPONSE	Oui, vous pouvez ajouter/modifier le fichier "digi_ned.mes" (ou tout autre fichier défini avec "`message_file:`" dans digi_ned.ini). Quand je veux ajouter une requête "`?bla`" qui répond "`Hello, World!`" je dois ajouter ce qui suit dans le fichier: `?bla`

```
Hello, World!
```

C'est aussi simple que cela. Si le message "`?bla`" ou "`bla`" est envoyé vers votre DIGI_NED alors DIGI_NED répondra à l'expéditeur du message avec "`Hello, World!`".

Quand je veux utiliser un raccourci pour ma requête je peux assigner des alias pour les commandes, séparés par le symbole "|". Supposons que je veuille utiliser le raccourci 'b' à la place de 'bla' alors je dois modifier ce que j'ai entré précédemment en:

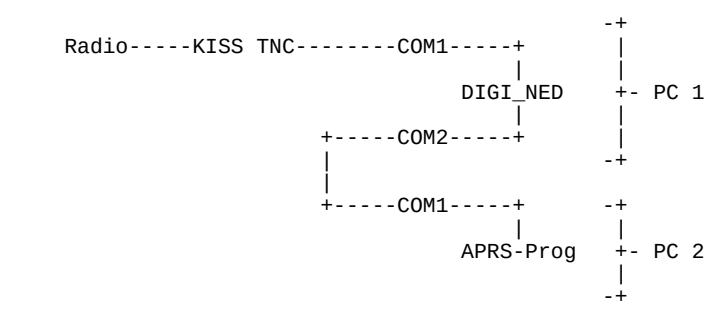
```
?bla|b
Hello, World!
```

Cette requête répond maintenant à `'?bla', 'bla', '?b'` et `'b'` et répond à l'expéditeur de la requête par "`Hello, World!`".

QUESTION	Peut-on utiliser un second port pour injecter les données reçues vers un autre programme APRS. C'est à dire, peut-on injecter les données de DIGI_NED vers WinAPRS et voir le programme WinAPRS fonctionner comme un second terminal? C'est un peu comme utiliser DIGI_NED en tant que hub ou comme routeur. Le(s) différents programmes vont tous attendre les données de DIGI_NED et tour à tour lui réinjecter leur données pour les répéter vers le TNC pour retransmission.
REPONSE	Oui vous le pouvez, le sysop de notre digipeater local le fait. La manière la plus simple de le faire est d'utiliser KISS sur le second port. Dans AX25_MAC utilisez l'option `-PKISS2` pour créer un lien KISS à 9600 bauds sur le COM-2. Laissez le programme APRS choisi se connecter sur ce port en mode KISS via un câble null-modem (câble croisé). Vous pouvez aussi étendre tout ceci, si vous ajoutez le paramètre `-PKISS3` en ligne de commande d'AX25_MAC; vous créez alors un second port KISS sur le COM-3, etc... Maintenant nous avons: `AX25_MAC -PKISS1 -PKISS2 -PKISS3 -F -C17 -BU50` Le premier port sera un port KISS sur le COM-1 qui est connecté à un TNC en mode KISS, le second port est un port KISS sur le COM-2 et le troisième port est un port KISS sur le port COM-3. Vous pouvez aussi connecter un deuxième TNC KISS à l'un de ces ports pour créer un digi cross-band. Si le programme que vous souhaitez utiliser supporte les trames BPQ sur Ethernet vous pouvez également l'utiliser. Du côté de DIGI_NED vous avez besoin d'un pilote packet compatible FTP pour votre carte réseau et utiliser l'option '`-PBPQ`' dans AX25_MAC pour créer un port BPQ sur Ethernet. Sous Linux vous pouvez utiliser un loopback local comme décrit pour connecter un iGATE à DIGI_NED. Au lieu d'un programme IGATE vous pouvez lancer par exemple Xastir de l'autre côté de cette boucle interne.

QUESTION	Peut-on lancer un shell DOS sans quitter DIGI_NED; je souhaite effectuer des changements dans le fichier balise.
REPONSE	Oui, c'est possible dans la version DOS (sous Linux vous pouvez toujours démarrer un autre shell ou lancer DIGI_NED en tâche de fond ou utiliser Ctrl-Z pour arrêter un programme, etc ...). Si vous tapez ALT-D un interpréteur DOS est démarré. Comme DIGI_NED est un "petit" programme vous aurez assez de mémoire disponible pour lancer un travail d'édition. DIGI_NED utilise la variable d'environnement COMSPEC pour savoir quel interpréteur lancer. Si COMSPEC n'est pas spécifié la fonction échouera. Avant de démarrer DIGI_NED vous pouvez faire la chose suivante depuis le prompt DOS: `C:> set COMSPEC=C:\COMMAND.COM` COMSPEC indique maintenant que le processeur de commandes DOS est en fonction C:\COMMAND.COM. Évidemment COMMAND.COM doit être présent, autrement la fonction ne se lancera pas.

QUESTION	Puis-je construire un petit réseau KISS avec DIGI_NED et y connecter un programme APRS? (suite)

REPONSE	Oui. Vous pouvez faire quelquechose comme ci-dessous:
	 
	
	Sur le PC 1 vous devez faire la chose suivante:
	1) Passer le TNC en mode KISS.
	2) Si besoin, adapter le TX-Delay pour votre TRX.
	3) Charger AX25_MAC de la manière suivante:
	`AX25_MAC -PKISS1 -PKISS2 -F -C17 -BU50`
	-PKISS1 est un port KISS sur COM1
	Première définition; sera la port 1
	-PKISS2 est un port KISS sur COM2
	Seconde définition; sera la port 2
	-F force AX25_MAC à lire le fichier AX25_MAC.INI
	-C17 est une indication visuelle, vous pouvez l'omettre
	-BU50 donne à la couche Mac 50 trames de buffer AX.25,
	par défaut c'est 100, vous pouvez donc l'omettre aussi.
	Cela va charger le pilote avec 2 ports KISS, un sur COM1 et un sur COM2. Je n'ai pas spécifié de vitesse, c'est pas défaut 9600 baud.
	Pour plus d'informations des paramètres de la ligne de commande reportez-vous à la documentation de TFPCX à: http://www.microlet.com/tfpcx
	Ensuite vous devez configurer DIGI_NED:
	4) Changez le paramètre "digi_call:" dans digi_ned.ini. Modifiez digibcon.ini et digi_id.ini pour entrer votre indicatif et votre position.
	5) Démarrer DIGI_NED et il devrait commencer à digipeater! Si vous utilisez l'option '-v' en ligne de commande de DIGI_NED vous devrez voir les données reçues et les actions effectuées par DIGI_NED sur ces trames.
	Les trames reçues par DIGI_NED seront digipeatées via COM1 vers la radio et via le COM2 vers le second PC connecté. Vous devrez peut-être ajouter une règle 'digiend:' pour forwarder les paquets terminant leur trajet vers COM2. Normalement un paquet avec aucun digipeater non-utilisé n'est pas répété, donc il risque de ne pas arriver sur COM2. Avec 'digiend:' vous pouvez digipeater les paquets qui sont normalement "au bout du trajet" de manière à les voir à l'écran de votre programme APRS. La configuration par défaut contient déjà deux lignes pour cela.
	La première ligne est:
	`digiend: all wide*,trace* 2 add LOCAL`
	Ici on digipeat tous les paquets où le dernier indicatif de digipeater était WIDE...ou TRACE...une fois de plus, on ajoute un autre indicatif de digipeater 'LOCAL'. Le premier 'all' veut dire "provenant de tous les ports", le '2' veut dire "vers le port 2". Donc les paquets reçus sur le COM1 sont renvoyés via le COM2. Ceci a été établi pour une utilisation en cross-band, vous aurez peut-être besoin de faire des modifications, comme:
	`digiend: 1 wide*,trace* 2`
	Maintenant seulement les paquets reçus sur le port 1 sont envoyés vers le port 2. Si votre digipeater local effectue la substitution d'indicatif alors vous aurez peut-être envie d'ajouter cet indicatif.
	La seconde règle 'digiend:' le fait pour moi. 'PE1MEW-2' est notre digipeater local. La règle ressemble à ceci:
	`digiend: 1 pe1mew-2 2`
	Cela veut dire que tous les paquets reçus sur le port 1 (le port est associé avec le premier paramètre -P d'AX25_MAC, soit le port KISS sur le COM1 dans ce cas) sont renvoyés via le port 2, qui est le second port KISS.
	6) Configurez votre programme APRS sur le deuxième PC pour qu'il utilise un TNC en mode KISS.
	Maintenant les trames venant de votre programme APRS doivent passer au travers du DIGI vers le TNC connecté, sur le COM1. Les trames qui apparaissent sur le COM1 sont répétées sur le COM2 et finissent leur parcours sur l'écran de votre programme APRS.

QUESTION	Qu'en est-il de l'idée de Bob Bruninga concernant le ZIP-lan; puis-je faire cela?

REPONSE	Je pense que vous pouvez utiliser la configuration ZIP-lan sans modifications avec DIGI_NED en maitre.
	(pour toute information concernant ZIP-lan reportez-vous à la distribution de DosAPRS)
	
	Il y a trois différences importantes:
	
	1) L'échange de paquets entre les stations est effectué en KISS au lieu de texte brut.
	
	2) Les paquets de la station esclave sont digipeatés vers les TNCs, ce qui veut dire que les stations doivent avoir un vrai indicatif et non pas un pseudo indicatif. Vous pouvez utiliser des pseudo- indicatifs pour bloquer les transmissions des esclaves si vous le voulez, vous devrez alors ajouter les indicatifs à la liste 'block'.
	Vous pouvez aussi configurer un esclave 'NOCALL', c'est déjà dans la liste. Pour les stations dont les transmissions vous intéressent vous pouvez utiliser un SSID distinct.
	
	Une autre option est d'utiliser des indicatifs spécifiques de digipeaters. Par exemple:
	
	`PE1DNN>APRS via ZIPLAN`
	
	Si il n'existe pas de règle dans DIGI_NED pour 'ZIPLAN' alors la trame ne sera pas digipeatée. Vous pouvez en faire une autre:
	
	`PE1DNN>APRS via ZIPTNC`
	
	Et ajouter la règle suivante au fichier DIGI_NED:
	
	`digipeat: 1 ZIPTNC 1 new WIDE,WIDE2-2`
	
	Les paquets reçus sur le port 1 où le prochain digi qui doit être utilisé est ZIPTNC, sont renvoyés à travers le port 1 avec un nouveau path WIDE,WIDE2-2 (exemple seulement!).
	
	3) Les paquets provenant des esclaves ne sont pas envoyés comme des messages 'tierce partie', DIGI_NED empêche les boucles en ne répétant pas les paquets qui ont déjà étés transmis en cherchant son propre indicatif dans la liste des digipeaters et en mémorisant les CRCs des paquets de données déjà répétés. C'est suffisant pour éviter les boucles. DIGI_NED ne répète jamais ses propres paquets!
	
	La configuration de DIGI_NED est identique à ce que nous avons vu plus haut, tout comme les questions se rapportant au réseau KISS. Cette fois-ci vous n'avez besoin que d'un port KISS. Vous pouvez aussi utiliser 2 ports, un avec le TNC et un avec le ZIP-Lan. De cette manière vous pouvez blinder encore plus les esclaves, vous pouvez définir que toutes les transmissions des esclaves ne sont pas digipeatées à moins que ce soit via 'ZIPTNC'. Avec une solutions câblée les esclaves peuvent toujours utiliser 'WIDE' comme destination et seront répétés.
	
	Évidemment, vous devez prendre conscience que les collisions sur le ZIP-lan endommagent les paquets, mais c'est identique en KISS que cela l'est pour le texte pur, c'est la simplicité du ZIP-lan. Il subsiste une chance que ces paquets endommagés soient transmis!
	
	Vous devez faire des essais avec cela, mais en ayant testé le système précédemment je ne vois pas de raisons pour que cela ne marche pas.
	Je serai très intéressé de connaître les résultats de vos expériences à ce sujet.

QUESTION	Je crois avoir trouvé un bug, que dois-je faire?
REPONSE	Faites-le moi savoir à pe1dnn@amsat.org Lisez également le fichier "BUGS.TXT".

CONCLUSION

Je reconnais que cette documentation est loin d'être terminée, mais au moins c'est déjà un début. J'espère progressivement la structurer en un véritable manuel de référence, l'étoffer et la présenter dans un format plus agréable à lire.

73,

Henk.

Traduction française, mise en page et création du fichier PDF: Jean-philippe Aumaitre - F5SMZ

Pour me joindre:
f5smz@qsl.net ou F5SMZ@LX0PAC.LUX.EU