Trente ans d'opensource ... pour en arriver là

J.-M Friedt, 14 avril 2024

1 Introduction

Été 2024 ... exactement 30 ans après la première installation de GNU/Linux sur un 80486 cadencé à 100 MHz, 80 disquettes copiées depuis un CD (distribution Slackware) dont je ne possédais pas le lecteur, avec évidemment la 79ème disquette défectueuse pour achever l'installation de X11 (aka XFree86 avant sa reprise en X.org en 1999). Peu importe, l'interface graphique ne sert à rien d'autre que consommer des ressources inutilement [1]. J'ai oublié la version du noyau (*kernel*), l'historique indique 1.1 mais je ne développais pas à ce niveau à cette époque. J'ai eu la chance de transiter de MS-DOS à GNU/Linux sans passer par l'étape MS-Windows, l'École Normale Supérieure de Lyon à laquelle j'accède en Septembre 1994 étant exclusivement munie de stations Sun Microsystems sous Solaris.

À partir de l'été 1995, le cher noyau 1.2.13 sur lequel nous avons fait nos première armes de pilotes (qui ne s'appelaient pas module à l'époque, cette fonctionnalité n'existait pas), où chaque modification au noyau prenait une nuit pour recompiler (Fig. 1). L'ajout des modules, autour de 1996, a considérablement facilité ces développements (version 2.0 du noyau).

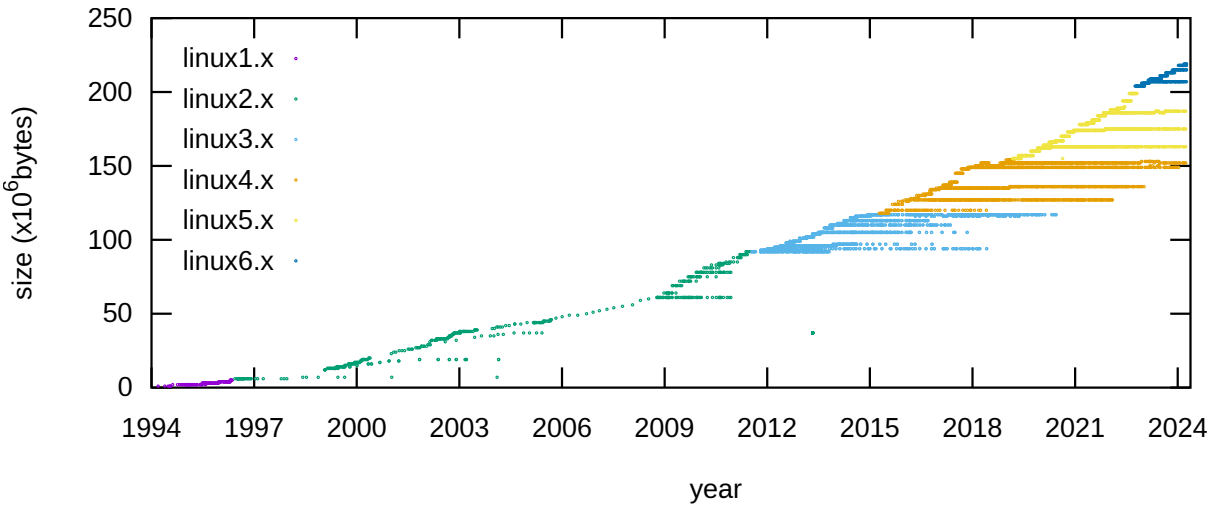


FIGURE 1 – Évolution au cours des années de la taille de l'archive `.tar.gz` du noyau Linux telle que documentée sur `kernel.org`. La métrique, certe discutable, met en évidence la difficulté croissante à aborder le code source d'un noyau supportant de plus en plus d'architectures de processeurs et de périphériques. Les lignes horizontales sont les sous-versions paires stables du noyau tandis que les versions impaires continuent à évoluer, nomenclature désormais abandonnée.

gnuplot

Il semble que `gnuplot` ne soit plus à la mode comme outil pour tracer des courbes. Pourtant il fait des miracles sur des fichiers dont le nombre de colonnes varie d'une ligne à l'autre (ce que déteste Octave), sait interpréter des valeurs en hexadécimal si on a pensé à les préfixer de `0x`, et les abscisses sous forme de date. En téléchargeant la liste des répertoires de `https://mirrors.edge.kernel.org/pub/linux/kernel/` dont chaque ligne est de la forme

```
linux-6.0.8.tar.gz 10-Nov-2022 17:29 204M
```

le script `gnuplot` pour tracer la Fig. 1 est de la forme

```
set size 1,1./2.
set xdata time
set timefmt "%d-%b-%Y" # format used to read input data
set xtics timedate     # controls interpretation of output format
```

```
set xtics format "%Y"  # format used for tic labels
plot 'linux.txt' u 2:($4/1e6) w d
set xlabel 'year'
set ylabel 'size (x10^6 bytes)'
```

pour analyser la deuxième colonne (`plot ... u 2:...`) comme un jour, mois sous forme de trois caractères, et année et ne retranscrire sur l'axe des abscisses que l'année. GNUPlot a surtout l'avantage de n'ouvrir le fichier pour en tracer le contenu qu'en lecture, et donc d'afficher alors que l'acquisition est en cours sans interférer avec l'écriture (sous Unix tout au moins).

Depuis, la philosophie prônée par Richard Stallman et formalisée par Éric Raymond [2] a été au cœur de nos développements, mettant à disposition de la communauté les outils développés dans les divers projets personnels et, quand les juristes l'autorisaient, professionnels. Cependant, après 30 ans d'utilisation et de développement d'outils opensource, force est de constater que les promesses du début ne sont pas tenues. Quelques retours d'expériences et réflexions sont proposées dans les paragraphes qui suivent.

Soyons néanmoins clairs dès le début de cette analyse : il ne s'agit pas de discuter logiciel propriétaire, produit par un mercenaire du code dont le seul but est de fournir dans les délais et le budget un produit plus ou moins médiocre pour lequel il n'a aucun intérêt, face au logiciel libre produit par un auteur qui a besoin de corriger une erreur ou de résoudre le problème qu'il se pose du mieux qu'il peut "*Every good work of software starts by scratching a developer's personal itch.*" [2, p.23]. La question qui se pose réellement est l'empilement de dépendances, et l'hypothèse que suffisamment d'yeux qui regardent le code rendent les erreurs "faciles" à corriger au plus vite, sous réserve que les yeux techniquement compétents soient encore disponibles compte tenue de la multitude des langages et infrastructures de développements mis en jeux, ne serait-ce que pour satisfaire les utilisateurs de logiciels libres sur systèmes d'exploitations propriétaires. Alors que les choses étaient simples entre `Makefile` et éventuellement `autoconf` de `M4`, nous subissons maintenant `cmake`, `ninja`, `meson`, `west`, `gradle` et multitude d'autres infrastructures de compilation multiplateformes, pour des langages qui tentent de remplacer le C avec des Rust ou Go dont on ne peut espérer que le même échec que Ada quand leur utilisation est imposés [3]. Entre développeurs et utilisateurs de systèmes compatibles POSIX uniquement, le problème serait sûrement plus simple à résoudre.

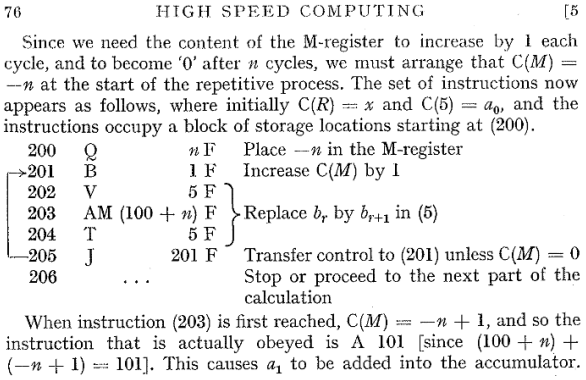
2 De l'électronique numérique programmable

Nous pouvons commencer par nous interroger comment nous en sommes venus à programmer des ordinateurs généralistes numériques respectant l'algèbre de Bool, d'abord en fournissant les instructions à l'unité arithmétique et logique (ALU) sous forme d'opcodes déterminant la séquence de portes logiques suivie par les opérandes, et plus tard avec des langages plus abstraits que seront C, bash, GNU Octave, Python ... et j'en oublie de nombreux autres aussi éphémères qu'inutiles. Cet environnement de travail ne doit pas faire oublier qu'il y a moins d'un siècle, le transistor était inconnu, la lampe à vide peu fiable, lente et gourmande en énergie.

Historiquement, le traitement du signal a commencé avec les ordinateurs mécaniques qui ont été longuement développés en assemblant engrenages, poulies et autres pièces mécaniques [4] pour que la sortie soit le résultat du traitement des entrées, par exemple en convertissant un mouvement circulaire en mouvement linéaire pour calculer une composante de Fourier, et en combinant ces pièces pour former une série de Fourier et donc résoudre un problème linéaire invariant en temps. Cependant, cette approche est source de nombre de problèmes : mauvaise reproductibilité des pièces dont la tolérance est dépendante de la dextérité de l'artisan qui les produit, usure des pièces mécaniques et donc dérive du résultat, frottements, et surtout le problème de l'impédance qui apparaît clairement avec l'exemple de contrôle d'une tourelle de tir de destroyer de plusieurs tonnes par une lunette de visée manipulée par un opérateur humain (bien incapable de déplacer les quelques tonnes) en tenant compte du mouvement relatif des deux navires pour que l'obus atteigne sa cible après un certain temps de vol[5, 6]. Cette illustration permet de bien comprendre pourquoi un amplificateur suiveur ne fournit en sortie que la même tension qu'en entrée, mais en absorbant un courant nul tout en étant capable de fournir un courant infini (ou presque, l'infini

étant assez grand). Le passage du mécanique à l'électronique analogique retira au moins les frottements et l'usure, mais reste encore la variabilité des valeurs des composants analogiques avec le temps ou avec leur environnement – le condensateur dont la valeur dépend de la permittivité relative change en fonction de l'humidité et de la température. Par ailleurs, deux circuits analogiques ne se comportent pas de la même façon compte tenu de la tolérance des valeurs des composants (résistance, condensateur, inductances, ...).

La transition à l'électronique numérique programmable avec un processeur généraliste [7] permettant d'exécuter n'importe quelle combinaison de ses instructions (Fig. 2) appliquées aux contenus des registres ouvre la perspective de reproduire un calcul et les résultats en partageant les exécutables, voir de partager les codes sources si l'interlocuteur est susceptible de modifier le comportement du programme. Le partage de code source ne devient cependant pertinent que lors du passage de l'assembleur, bijectif avec les opcodes qui configurent la séquence de portes logiques traversées par les données dans l'unité arithmétique et logique (ALU) – l'exécutable – à un langage de haut niveau tel que le C conçu spécifiquement pour rédiger Unix. Cette configuration de l'ALU se faisait historiquement en programmant les opcodes directement, puis grâce aux mnémoniques pour rendre le code lisible par un(e) humain(e), et finalement produits automatiquement par un compilateur[8]. En effet l'étape de compilation, qui fait passer du code C au code assembleur, n'est pas bijectif, et n'avoir que l'exécutable ne permet pas de remonter simplement au code source original en C, mais uniquement de retrouver par désassemblage le code assembleur, peu intuitif si un algorithme un tant soit peu complexe a été implémenté. L'analyse des séquences d'opcodes produits par `gcc` est un de mes passe-temps favoris tellement l'optimisation est devenue impressionnante. Partager le code source du langage de haut niveau devient donc indispensable pour permettre aux interlocuteurs de modifier les fonctions du programme, et cette nécessité devient d'autant plus claire que le niveau d'abstraction s'élève et que la représentation du problème s'éloigne du langage de la machine. Ghidra (`https://www.ghidra-sre.org/`) peut bien essayer d'imaginer quelle séquences d'instructions en C a produit une certaine séquence d'instructions en assembleur, mais l'abstraction s'arrête là : retrouver des objets et classes de C++ par exemple devient impossible juste avec la séquence d'opcodes.



76 HIGH SPEED COMPUTING [5

Since we need the content of the M-register to increase by 1 each cycle, and to become '0' after n cycles, we must arrange that $C(M) = -n$ at the start of the repetitive process. The set of instructions now appears as follows, where initially $C(R) = x$ and $C(5) = a_0$, and the instructions occupy a block of storage locations starting at (200).

200	Q	n F	Place $-n$ in the M-register
201	B	1 F	Increase $C(M)$ by 1
202	V	5 F	Replace b_r by b_{r+1} in (5)
203	AM $(100 + n)$ F		
204	T	5 F	
205	J	201 F	Transfer control to (201) unless $C(M) = 0$
206	...		Stop or proceed to the next part of the calculation

When instruction (203) is first reached, $C(M) = -n + 1$, and so the instruction that is actually obeyed is A 101 [since $(100 + n) + (-n + 1) = 101$]. This causes a_1 to be added into the accumulator.

Figure 2: Tant que le programme est écrit en langage machine, le concept d'opensource n'a pas de sens puisque code source et exécutable sont équivalents, à une traduction bijective près des opcodes en mnémoniques. Ce n'est que l'utilisation de langages de haut niveau d'abstraction qui rend l'opensource nécessaire. Figure extraite de [7]

3 Pérennité des applications et des formats de données

Au milieu des années 1990, Éric Raymond nous convainc que le développement en bazar au lieu de la cathédrale proposée par Richard Stallman lorsqu'il développe les premiers outils GNU et en particulier Emacs. Cela était probablement valable avant que le bazar ne s'installe dans la tour de Babel. En effet, alors que "*A programmer could easily hold the entire logical structure of C in his head (unlike most other languages before or since) rather than needing to refer constantly to manuals*" (p. 9), il devient aujourd'hui difficile de maîtriser l'ensemble des outils et langages impliqués dans un projet de grande envergure tel que KiCAD ou GNU Radio. À titre de démonstration du nombre délirant de dépendances dans la distribution Debian GNU/Linux pour ces outils,

```
$ debtree gcc-avr | cut -d\- -f1 | grep \" | sort | uniq   | wc -l
6
$ debtree gimp | cut -d\- -f1 | grep \" | sort | uniq   | wc -l
190
$ debtree kicad | cut -d\- -f1 | grep \" | sort | uniq   | wc -l
207
$ debtree inkscape | cut -d\- -f1 | grep \" | sort | uniq   | wc -l
247
```

```
$ debtree gnuradio | cut -d\- -f1 | grep \" | sort | uniq   | wc -l
764
```

indique que le compilateur GCC à destination des cibles AVR dépend de 6 autre paquets alors que GNU Radio dépend de 764 autres paquets (nous avons validé manuellement la sortie de cette commande pour `gcc-avr` (Fig. 3) et laissons au lecteur le soin de valider pour les autres cas !). Une seule de ces dépendances échoue, pour un bug, un changement d'API ou une incompatibilité avec le compilateur en cours d'utilisation, et c'est tout GNU Radio qui échoue à fonctionner. Autant dire que les chances de recompiler aujourd'hui, à l'époque de gcc-13 et de Pybind pour GNU Radio 3.10, une version de GNU Radio 3.7 s'appuyant sur SWIG, sont absolument nulles (déjà qu'à l'époque la tâche n'était pas triviale sans l'aide de PyBOMBS ... qui n'est plus maintenu).

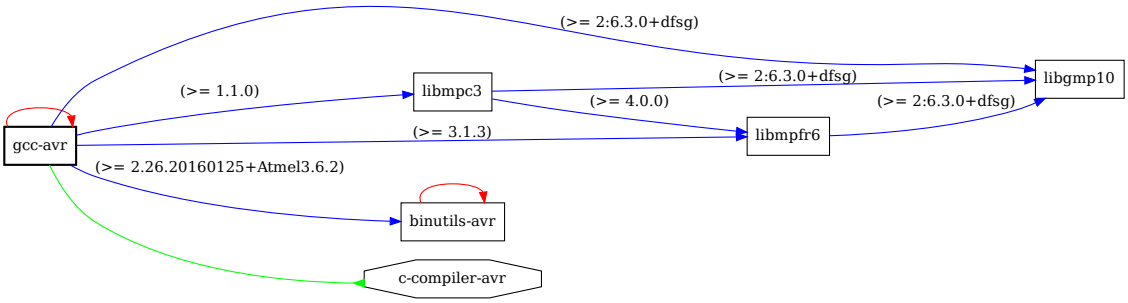


FIGURE 3 – Arbre des dépendances de `gcc-avr`.

À titre d'illustration, plus ludique qu'instructive, en Fig. 3 le graphique en `dot` de la sortie de `debtree gcc-avr | dot -Tpdf > gcc-avr.pdf`, le compilateur C à destination des microcontrôleurs AVR (maintenant Microchip) en interface en ligne de commande, et en Fig. 4 le résultat appliqué au paquet `kicad` pour l'outil de conception de circuits imprimés avec son interface graphique et sa console de programmation en Python, qui laisse rêveur (noter que les sources de KiCAD sur `gitlab` ne compilent pas en l'état en Debian SID, un changement d'API de OpenGL, toujours la faute de ces maudites interfaces graphiques). Nous épargnons le graphique pour GNU Radio dont le calcul prend près de 5 minutes sur les 4 cœurs d'un processeur i5 cadencé à 2,7 GHz. Par ailleurs, alors que Éric Raymond a bien identifié que "*Most developers (including me) used to believe this was bad policy for larger than trivial projects, because early versions are almost by definition buggy versions and you* **don't want to wear out the patience of your users**. *This belief reinforced the general commitment to a cathedral-building style of development. If the overriding objective was for users to see as few bugs as possible, why then you'd only release a version every six months, and work like a dog on debugging between releases*", l'intégration et le développement continus (CI/CD) se traduisent invariablement par un code qui certes compile, mais dont la validité est laissée au soin de l'utilisateur. Probablement un des rares bénéfices d'enseigner est qu'année après année il faut vérifier que les fonctionnalités attendues des logiciels qui appuient les sujets enseignés restent disponibles, voir corriger leur disparition lorsqu'elles sont constatées (e.g. `https://gitlab.com/kicad/code/kicad/-/issues/17227`). La cours effrénée des évolutions par une multitude de développeurs aux objectifs antagonistes rend cependant la tâche lourde, voir épuisante, quand chaque nouvelle séance de travaux pratiques risque d'exposer une nouvelle déficience des dernières versions de logiciels, avec un délai souvent insuffisant pour en corriger les dysfonctionnements qui s'accumulent pour l'année suivante.

Cette pléthore de dépendances est prise en charge par les distributions binaires de GNU/Linux, mais chaque développeur les subit quotidiennement lorsqu'il tente de cross-compiler à destination d'une cible d'architecture autre que l'hôte sur lequel il compile. Ainsi, maintenir un environnement de compilation fonctionnel de GNU Radio dans Buildroot est une activité presque à plein temps tant le nombre de dépendances est important et la vitesse de mise à jour des versions effreinée. Étant donné qu'il est probablement impossible de valider toutes les combinaisons possibles de paquets de Buildroot, chaque nouvelle version casse une des dépendances de GNU Radio et donc cette application – pour 2024.02 c'est `python-numpy` qui ne compile plus.

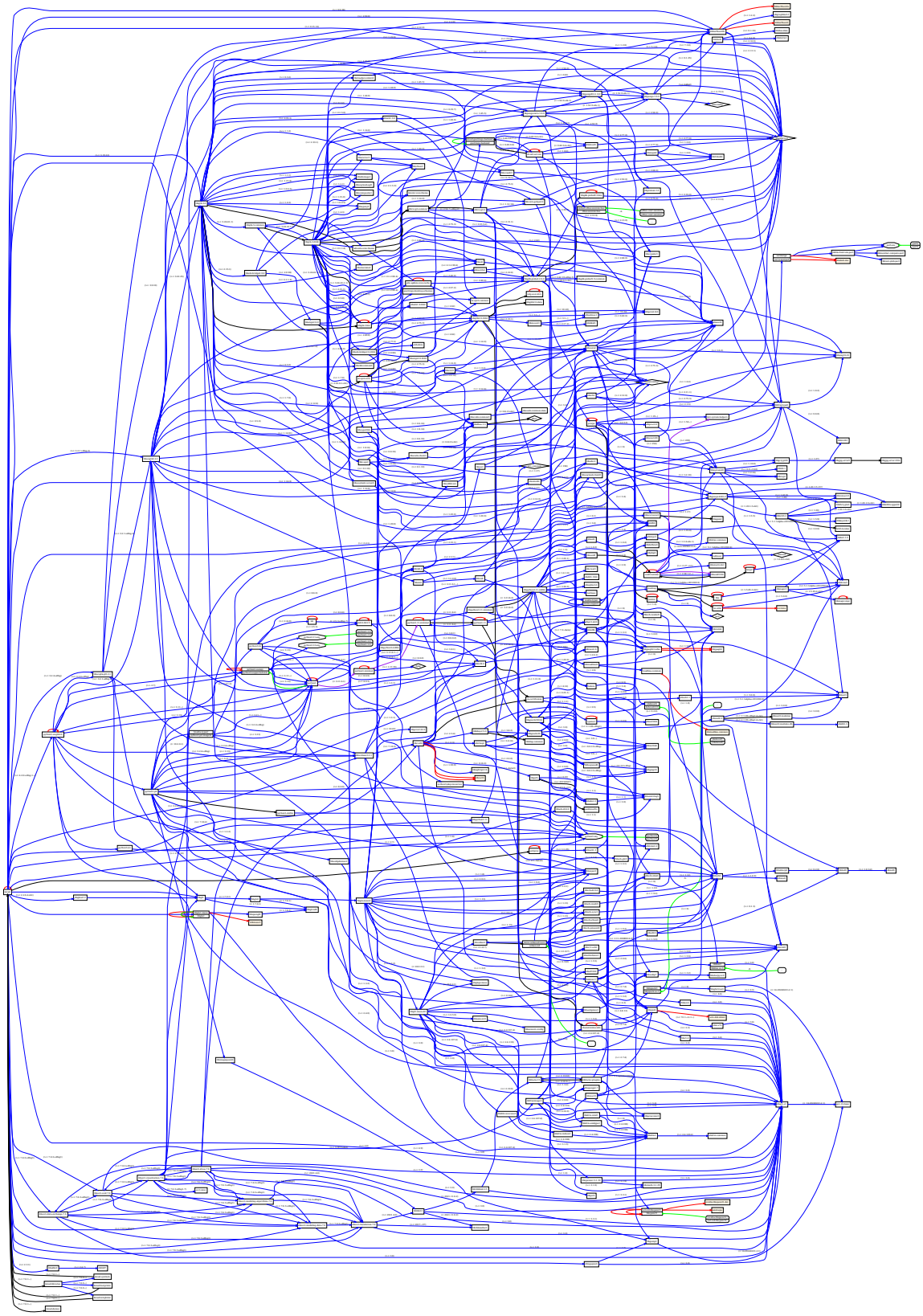


FIGURE 4 – Arbre des dépendances du paquet KiCAD dans Debian GNU/Linux SID.

Le langage C semble un compromis optimal entre niveau d'abstraction élevé (boucles, branchements) pour un nombre réduit d'instructions et néanmoins accès (via les pointeurs) aux adresses mémoires reliant électronique et logique au travers des registres de configuration des périphériques, reliant variables abstraites et fonctions physique. L'évolution fait cependant croître cette abstraction vers des structures qui se multiplient sans logique apparente – la difficulté à manipuler des programmes objets sous `vi` écrits en Python ou C++ illustrent la nécessité d'une assistance (IDE) pour naviguer dans la multitude des classe sans logique entre elles autre que celle de leurs concepteurs – et maintenant ces niveaux d'abstraction deviennent encore plus contraignants avec les nouvelles extensions de C++, voir de nouveaux langages qui vont inciter à reprendre tous les codes (Go, Rust ...). Après avoir vu passer Awk, Perl, Ruby et finalement Python, ou Ada, Pascal, CAML pour les voir tous mourir plus vite que nous n'avons eu de temps à les appréhender, pourquoi continuer à construire cette tour de Babel aussi incohérente qu'éphémère. Pire, le code n'est plus écrit mais généré : Qemu pour STM32 (`https://github.com/beckus/qemu_stm32`) va mourir parce-que l'analyse de la liste des périphériques des microcontroleurs pour produire la configuration de Qemu est écrite en Python 2.7 obsolète, et je viens de me heurter aux passages de paramètres dans `gnss-sdr` qui utilise `protobuf` (`https://protobuf.dev/`) pour produire automatiquement ses interfaces d'écriture et de lecture des données échangées par UDP. Encore une dépendance qui induira la mort du logiciel quand elle sera brisée, qui a déjà pu arriver puisque `https://gnss-sdr.org/docs/sp-blocks/monitor/` explique que la transition a déjà du être faite depuis *Originally Boost.Serialization was used but in release v0.0.11 it was deprecated in favor of Protocol Buffers.* Les dernières extensions de C++, `https://isocpp.org/files/papers/P1673R13.html`, semblent ne contenir la lettre "C" que par compatibilité historique mais la syntaxe n'a plus de rapport avec la langage original.

4 Pourquoi écrire du logiciel libre ?

Éric Raymond (p.30) argumente qu'une motivation des développeurs de logiciels libres est la reconnaissance des pairs pour leur contribution, "*stimulated by the prospect of having an ego-satisfying piece of the action, rewarded by the sight of constant improvement in their work*" mais cette satisfaction est désormais concurrencée par l'addiction aux réseaux sociaux dont le retour nécessite un investissement intellectuel et technique bien moindre. Alors que la liste de diffusion de fetchmail a atteint à son apogée 287 interlocuteurs (p.38), n'importe quel compte de réseau social compte un nombre de participants en milliers, avec une exposition plus satisfaisante à court terme que les efforts nécessaires à un *pull request* pertinent. La retour instantané des réseaux sociaux pour une contribution aux performances techniques ou intellectuelles nulles rend la recherche d'une solution techniquement subtile ou complexe – et donc gourmande en temps – à un problème logiciel peu attractive. D'un autre côté, la stratégie de contribution proposée par `git` me laisse quelque peu sceptique : `forker` un dépôt entier pour y modifier quelques lignes paraît discutable, mais surtout la forte propension à ne jamais `merger` les contributions induit une multiplicité de branches divergentes dans lesquelles le novice ne peut se sortir. À titre d'exemple, `rtl-sdr`, le pilote GNU Radio de l'interface matérielle RTL-SDR faible coût pour le traitement de radio logicielle, possède (`https://github.com/osmocom/rtl-sdr`) 282 forks dont l'objectif ou la contribution ne sont pas clairement identifiées et encore moins fusionnées au projet initial. Même au sein de White Rabbit, avec un groupe restreint de développeurs expérimentés du CERN,
`git clone https://ohwr.org/project/wr-cores && cd wr-cores && git branch -a | wc -l`
indique 172 branches, impossible à démêler sans être sur site (et même en y étant ...), dans la même veine que les 335 branches de `https://github.com/GSI-CS-CO/bel_projects`. L'alternative de diffuser des `patchs` par liste de diffusion ne semble pas bien meilleure que par `github` où ma proposition de correctif d'un composant (`https://gitlab.com/kicad/libraries/kicad-symbols/-/merge_requests/3939`) a nécessité de dupliquer toutes la bibliothèque pour corriger une paire de lignes afin de finalement respecter les consignes de modifications, puisque la liste de diffusion de Buildroot croule sous des dizaines de correctifs quotidiens dont il est difficile de voir comment la cohérence peut être validée en temps réel. Bien entendu chaque nouvelle version de Buildroot est l'occasion de constater que GNU Radio ne compile plus, mais je dois être le seul à m'obstiner à vouloir cross-compiler une infrastructure aussi complexe vers des systèmes embarqués, et par Buildroot qui plus est.

J'ai passé mes années de lycée à retranscrire les cours manuscrits en version numérique sous WordPerfect 5.1 sous MS-DOS (quand on s'en sert tous les soirs, on se rappelle de la version, acquise piratée

à Singapour en même temps qu'une copie de LISP qui aura bien moins servi). Il est assez amusant de se rappeler que comme sous LaTeX, l'utilisation de clavier qwerty sans caractères français n'empêchait pas d'ajouter les accent en mode commande. Il est évident qu'aujourd'hui tout ce travail est perdu, car même en supposant que je retrouve un lecteur de disquette 3.5", je n'ai aucune idée comment lire ces fichiers au format propriétaire, quand je peux encore compiler la thèse rédigée en 2000, déjà sous LaTeX2e puisque la transition de LaTeX2.09 s'est effectuée fin des années 1990 sans trop de douleur, principalement l'entête `\documentstyle` devenu `\documentclass`. Néanmoins, la persistance des formats de fichiers devrait être un pré-requis incontournable pour utiliser un outil capable de produire ces fichiers, et l'incapacité de MS-Office et LibreOffice d'échanger des fichiers XML décrivant le format de leurs fichiers ne présage pas d'une interopérabilité sur le long terme, ni même le court, quelle qu'en soit la cause. La conclusion est évidemment valable pour tout format numérique, que ce soient les images (qui sait décoder à la main une image JPEG ? [9] peut être autant qui savent décrypter les hiéroglyphes), les circuits imprimés ou les pièces mécaniques que même des instituts de recherche publics aussi glorieux que le CERN s'obstinent à produire avec des outils (et formats de fichiers) propriétaires malgré tous leurs efforts vers des solutions libres.

5 Du traitement décentralisé de données : le "cloud"

Peut être que la mort du logiciel libre viendra du traitement décentralisé de l'information avec des infrastructures logicielles que l'utilisateur ne maîtrise plus, le "cloud". Dans ce contexte, le traitement de signaux n'est plus un développement mais un service requis sur un ordinateur dont on ne maîtrise plus ni l'installation, ni les bibliothèques et donc les algorithmes exécutés. Il ne faut bien entendu pas s'y tromper, le nuage n'est que déporter vers quelqu'un d'autre le travail de gestion de l'infrastructure informatique, et donc en perdre la maîtrise : comme le dit Richard Stallman, "It's stupidity. It's worse than stupidity : it's a marketing hype campaign"[10]. Les autocollants produits par la FSF Europe affirmant "*There is no cloud – just other people's computers*" oublient de mentionner que ces autres personnes ne sont pas forcément des amis du logiciel libre, et que déporter les calculs enfreint les libertés fondamentales du libriste de consulter le code, le modifier et le redistribuer.

L'industrie logicielle explique [11] que depuis une dizaine d'années les "web apps" dominent devant les applications locales, en prenant pour exemple Office 365 ou les outils de partage en ligne de documents de Google. Il me semble inimaginable qu'un défenseur des libertés associées aux outils FOSS puisse utiliser aucune de ces infrastructures logicielles dépendantes d'une connexion internet, faisant circuler des documents sur un réseau dont la sécurité n'est pas contrôlée, pour les faire traiter sur un serveur distant dont la confidentialité n'est aucunement garantie. Seuls les auteurs de logiciels propriétaires, qui ont réussi à standardiser un modèle de location de service au lieu d'une acquisition de logiciel, peuvent promouvoir un tel mode de traitement décentralisé de l'information dont l'utilisateur perd le contrôle. Pourtant, j'ai vu des mes propres yeux une étudiante se connecter via son browser web à `octave-online.net` alors qu'elle travaillait sur un ordinateur sous GNU/Linux avec une installation locale de GNU Octave munie de toutes ses extensions (`packages`). Comment argumenter dans ces conditions de bénéficier d'un logiciel opensource en consultant l'implémentation de fonctions un peu complexes, par exemple `xcorr.m` dans `/usr/share/octave/packages/signal-${version}/xcorr.m` pour démontrer que GNU/Octave utilise efficacement le passage dans le domaine de Fourier pour calculer la corrélation et pas l'expression dans le domaine temporel de complexité quadratique avec la longueur du vecteur ?

Cependant, cette quête de la connaissance nécessite un minimum de bagage scientifique et technique pour pouvoir se poser des questions. Eric Raymond relie clairement capacité de création et outils lorsqu'il affirme [2, p.14] "*the fact that non-Unix operating systems don't come bundled with development tools meant that very little source was passed over them. Thus, no tradition of collaborative hacking developed.*" Une fois acquis que la connaissance est inutile car reportée sur Google, et maintenant la réflexion est dévolue à OpenAI, le sens premier du logiciel libre de partager l'expérience de l'auteur du logiciel au travers de la lecture de son code source devient caduque. Que le patron de la société NVidia explique que les générateurs automatiques de langage seront le prochain mode de programmation et qu'il devient inutile d'apprendre à coder se comprend, son objectif est de vendre ses circuits intégrés, mais que cet argumentaire soit repris par des chroniqueurs de la chaîne nationale publique radiophonique d'information semble dramatique en terme d'avenir de la formation technique. Ce mode de programmation ne répond ni à mon expérience de tentative de génération de code ni même de tentative de traduction d'un langage

à l'autre, même pour des langages aussi proches que Python et Octave, et les résultats aux examens d'utilisateurs de ces outils ne semble pas indiquer un meilleur succès des étudiants dans l'usage de cet outil. En tout état de cause le logiciel libre ne semble pas être intégré dans une telle réflexion. Cependant, délocaliser la connaissance et la réflexion a une conséquence majeure : la connexion constante et systématique, sans laquelle les outils deviennent inaccessibles et ses utilisateurs démunis.

6 De la connectivité globale : de GSM et Iridium à Starlink

Le traitement déporté des informations acquises par les systèmes embarqués impose une connectivité réseau pour transmettre les informations acquises vers les centres de traitements. Aussi fragile qui puisse sembler cette approche, elle est par exemple au cœur d'un réseau LoRa dans lequel les capteurs nommés *endpoints* transmettent sur une portée de quelques kilomètres à dizaines de kilomètres leurs mesures vers des *gateways* qui se chargent ensuite de router les informations au travers d'un réseau, filaire ou non, vers un *network server*. Tant que cette communication se limite à l'observation de données environnementales sans grande importance, la perte de connectivité n'a que peu d'impact, mais qu'en sera-t-il lorsque des données critiques seront acquises de la sorte (consommation d'électricité ou d'eau, au hasard). Cette omniprésence des communications sans fil s'est posée au cours des années 90 avec les deux options des relais terrestres et satellitaires. En effet, [12] enseigne que la négociation sur l'attribution des bandes radiofréquences entre l'Europe et les États-Unis a porté sur un échange dans lequel le premier obtient les ressources nécessaires à déployer les réseaux terrestres GSM tandis que le second se voit attribuer les ressources pour déployer le premier réseau satellitaire, Iridium. Alors qu'il pourrait sembler évident aujourd'hui que le réseau terrestre s'est imposé et que le réseau satellitaire ne répond qu'aux besoins de marchés de niche où les réseaux terrestres n'accèdent pas (militaires, aventuriers, exploitations minières et pétrolières), l'histoire se redessine avec les lancements de constellations massives telles que Starlink, OneWeb et autres essaims de "petits" satellites relais. Néanmoins une originalité majeure d'Iridium – maintenue dans la nouvelle mouture d'Iridium Next – est la communication *entre satellites*, argument majeur pour une utilisation militaire puisque l'origine de l'appel ne peut pas être triangulée par analyse du signal reçu au sol (contrairement aux appels vers un satellite géostationnaire qui ne fait que relayer l'appel, voir par exemple `https://www.kratosdefense.com/products/space/signals/rf-management/satid` selon une retransmission qualifiée de *bent pipe*). Néanmoins, l'utilisation de satellites pour relayer les signaux téléphoniques et de connexion aux réseaux informatiques rend les radiotélescopes sourds aux murmures des émetteurs en espace lointain. La robustesse d'Iridium a été mise à rude épreuve tel que le décrit l'extrait accompagnant [12], démontrant comment une dépendance excessive en un tel réseau sans redondance peut s'avérer dangereux.

L'évolution des systèmes embarqués tend donc ici encore à décentraliser les calculs vers des ordinateurs puissants dont le code n'est pas nécessairement maîtrisé, imposant le bon fonctionnement du réseau de communication pour transmettre les mesures et recevoir les commandes. Les réseaux de plus en plus rapide et de plus en plus denses (5G, 6G ...) n'ont pas vocation à faire communiquer des humains mais des machines. La science fiction nous rattrapera peut être, alors que Bruce Sterling prédisait dans sa fiction *Heavy Weather* (Bantam USA) de 1994 en p.175 "*You see, Jane, there are many places in America where human beings just can't live anymore, but that's not true for our communications technologles. The machines are literally everywhere. In the U.S.- even Alaska !-there's not one square meter left that's not in a satellite footprint, or a radio-navigation triangulation area, or a cellular link, or in packet range of nemode sites or of wireless cable TV.... 'Wireless cable,' that's a nasty little oxymoron, isn't it ?" Leo shook his head. "It took a truly warped society to invent that terminology...."*". Quelle dommage que le roman poursuive avec une solution technologique erronée " *Leo seemed lost for a moment, then recovered himself. "Except, Jane, not here, and not now ! For one shining moment, not here, not around us ! Because we are inside the F-6 ! The most intense, thorough, widespread devastation that the national communications infrastructure has suffered in modern times. Bigger than a hurricane. Bigger than earthquakes. Far bigger than arson and sabotage, because arson and sabotage on this huge scale would be far too risky, and far too much hard work. And yet here we are, you see ? In the silence ! And no one can overhear us ! No one can monitor us ! Not a soul."* puisque les tornades ne sont pas nécessairement associées aux précipitations qui seules pourraient perturber une liaison micro-onde satellitaire[13], la destruction des infrastructures de communication au sol n'ayant pas d'importance si effectivement le relais de communication devient spatial. "*But for a little while, a brilliant, perfect silence, and in that moment all things are possible.*

Everything is possible ! Even freedom.".

7 Conclusion

L'infrastructure des logiciels libres fournit un environnement idéal pour apprendre (et enseigner) et des outils robustes que chacun peut corriger et améliorer à son gré (et ses compétences). Cependant, une intrication croissantes des diverses bibliothèques rend la pérennité incertaine, tandis que le nombre croissant d'outils impliqués dans un développement de logiciel un tant soit peu complexe rend les contributions externes de plus en plus difficiles tant le nombre de langages et de préceptes à maîtriser deviennent importants. Alors que l'évolution vers cette infrastructure s'est faite petit à petit au cours des décennies, le passage vers un traitement de l'information décentralisé ("cloud") dans lequel l'utilisateur n'a plus la maîtrise des outils qu'il utilise change le paradigme et interroge sur l'avenir d'un modèle opensource et la contribution des développeurs. Par ailleurs alors que d'un côté le développement de systèmes embarqués ("edge") n'a jamais été aussi aisée avec pléthore de documentations, de datasheets disponibles sur le web et d'environnements de développements libres, la complexité croissantes des interfaces de communication (RS232 remplacé par USB, ISA remplacé par PCIe, VGA remplacé par HDMI ...) les rend moins abordables pour le débutant.

Max Planck a écrit en 1950 dans sa biographie que "*A new scientific truth does not triumph by convincing its opponents and making them see the light, but rather because its opponents eventually die and a new generation grows up that is familiar with it ...",* principe repris et formalisé par Thomas Kuhn dans son paradigme de la révolution scientifique. L'industrie du logiciel et l'évolution du concept de logiciel libre ne sont certainement pas étrangers à ce principe. C'est ce que nous constatons dans le monde des radioamateurs avec la transition vers la radio logicielle, et que nous avions déjà trouvé dans les ouvrages édités par Jim William, dieu de l'électronique analogique chez Linear Technology, quand *Analog Circuit Design – Art, Science, and Personalities* (Elevier, 1991) contient au chapitre 6 le texte de S. Wilensky intitulé *Reflections of a Dinosaur*, dont un extrait est *"The explosive growth of digital technology is the cataclysmic event that has threatened the analog designer with extinction. The linear circuit engineer has been added to the list of endangered species. For the past twenty years the focus of the engineering curriculum has shifted priority from analog to digital technology. The result of this shift is that only a small fraction of recently trained engineers have the analog design skills necessary to attack "real world" problems. The microprocessor has revolutionized the area of measurement and control, but the transducers used to measure and control temperature, pressure, and displacement are analog instruments. Until sensors and actuators are available that can convert a physical parameter such as temperature directly to digital information, the analog designer will still be in demand."* Espérons que les auteurs de logiciels libres seront encore sollicités eux aussi tandis que le paradigme de l'enseignement de la programmation dérive vers... je ne sais quoi.

Références

[1] N. Stephenson, *In the beginning... was the command line*, Avon Books (New York, 1999) même si son auteur a renié son texte par la suite.

[2] E.S Raymond, *The Cathedral and the Bazaar – Musings on Linux and Open Source by an Accidental Revolutionary*, O'Reilly (2001) à `https://monoskop.org/images/e/e0/Raymond_Eric_S_The_Cathedral_and_the_Bazaar_rev_ed.pdf`

[3] White House Office of the National Cyber Director, *Back to the building blocks : a path toward secure and measurable software* (Fev. 2024) à `https://www.whitehouse.gov/wp-content/uploads/2024/02/Final-ONCD-Technical-Report.pdf`

[4] A.B. Clymer, *The mechanical analog computers of Hannibal Ford and William Newell*, IEEE Annals of the History of Computing **15**(2) 19–34 (1993) à `https://web.mit.edu/STS.035/www/PDFs/Newell.pdf`

[5] B. Stuart, *A history of control engineering, 1930–1955* **47** IET (1993)

[6] D.A. Mindell, *Digital Apollo : Human and machine in spaceflight*, MIT Press (2011)

[7] W.H. Hollingdale, *High speed computing – methods and applications*, The English Universities Press Ltd. (1959)

[8] X. Leroy, *Le logiciel, entre l'esprit et la matière*, leçon inaugurale du collège de France à `https://www.college-de-france.fr/fr/agenda/lecon-inaugurale/le-logiciel-entre-esprit-et-la-matiere-0` (2018)

[9] J.-M Friedt, *Décodage d'images numériques issues de satellites météorologiques en orbite basse : le protocole LRPT de Meteor-M2* (en 3 parties), GNU/Linux Magazine France (2019)

[10] *Cloud computing is a trap, warns GNU founder Richard Stallman*, The Guardian (29 Sep. 2008), à `https://www.theguardian.com/technology/2008/sep/29/cloud.computing.richard.stallman`

[11] Amazon Web Services, *What is Cloud Native ?* à `https://aws.amazon.com/what-is/cloud-native/`

[12] J. Bloom, *Eccentric Orbits : The Iridium Story*, Atlantic Monthly Press (2016) dont un extrait est "*The Iridium system could handle 98,586 calls at a time—the math had been worked out by Ken Peterson long ago—and the company barely had 140,000 phones in service. Could there be a glitch in the pattern of cell reuse ? Could the phased-array antennas be duplicating the same call or failing to pass it off to the next satellite ? The technician dug deeper into the numbers scrolling rapidly across his screen, zeroed in on the heavy traffic, and then realized all the calls were originating in the same fifty-thousand-square-mile zone. It was possible to overload one small part of the system if, by some odd set of circumstances, massive numbers of people were all sending signals to the same satellite.*

And that's what was happening. It was a holiday in the United States, and there were 146,000 American troops in Iraq, and there was only one phone they could all depend on. Technically you weren't supposed to use an Iridium phone for a "morale call," but when you absolutely, positively had to get your call through at a certain time to a certain person, every soldier, sailor, Marine, and airman knew there was only one solution. The only situation that could max out the Iridium system was Mother's Day in a combat zone.

Mother's Day 2004 was not only a turning point for Iridium—around that time the company reached cash-flow break-even and started making money—but it was the first time the rest of the world realized there was a phone that worked everywhere on the surface of the planet. For cutting-edge professionals, there were actually three events that made Iridium famous. The first was 9/11. The second was the war in Afghanistan, even more than Iraq, since the terrain was so rugged and the infrastructure so primitive. And the third game-changing event was Hurricane Katrina, which was not the first natural disaster where relief workers used the Iridium phone as a lifeline, but was definitely the highest profile.

[13] G Maral & M Bousquet, *Satellite Communications Systems*, Wiley (1986) indique jusqu'à 10 dB/km d'atténuation pour les plus fortes précipitations de 150 mm/h aux fréquences descendantes de Starlink de 10–12 GHz