Telecom Bretagne
655 Avenue du Technopole
29200 Plouzané

Promotion 2015 / 2018

Rapport de simulation d'un système de transmission – SIT 213

Auteurs	Marc LEMAUVIEL – Johann CORCUFF-REBECCHI – Arouna KANE – Lim Kévin CHAO – Lahoucine AHMOUCHE
E-Mails	marc.lemauviel@telecom-bretagne.eu johann.corcuff-rebecchi@telecom-bretagne.eu arouna.kane@telecom-bretagne.eu lim.chao@telecom-bretagne.eu lahoucine.ahmouche@telecom-bretagne.eu
Destinataire	Éric COUSIN, Bruno FRACASSO, Julien MALLET, François-Xavier SOCHELEAU
E-Mail	eric.cousin@telecom-bretagne.eu bruno.fracasso@telecom-bretagne.eu julien.mallet@telecom-bretagne.eu fx.socheleau@telecom-bretagne.eu
Formation suivie	Ingénieur spécialisé en Informatique, Réseaux et Télécommunications
Établissement	Télécom Bretagne
Promotion	2018

Date	Version	Modifié par	Motifs
13/09/2016	1	LEMAUVIEL Marc	Création du document
06/10/2016	2	LEMAUVIEL Marc	MAJ – Organisation travail d'équipe
17/10/2016	3	LEMAUVIEL Marc	MAJ – Etape 2
01/11/2016	4	LEMAUVIEL Marc	MAJ – Etape 3

07/11/2016	5	LEMAUVIEL Marc	MAJ – Etape 4

Rapport de simulation d'un système de transmission – SIT 213

Étudiants : Marc LEMAUVIEL, Johann CORCUFF-REBECCHI, Arouna KANE, Lim Kévin CHAO, Lahoucine AHMOUCHE étudiants en Formation d'Ingénieur en Partenariat

Promotion : 2018

Établissement : Télécom Bretagne

Destinataires : Éric COUSIN, enseignant chercheur au département INFO

Bruno FRACASSO, enseignant chercheur au département OPTIQUE

Julien MALLET, enseignant chercheur au département INFO

François-Xavier SOCHELEAU, enseignant chercheur au département SIGNAL & COM

TABLE DES MATIERES

Introduction

L'objectif de ce projet est de nous faire étudier une chaine de transmission (représentée par la figure 1).

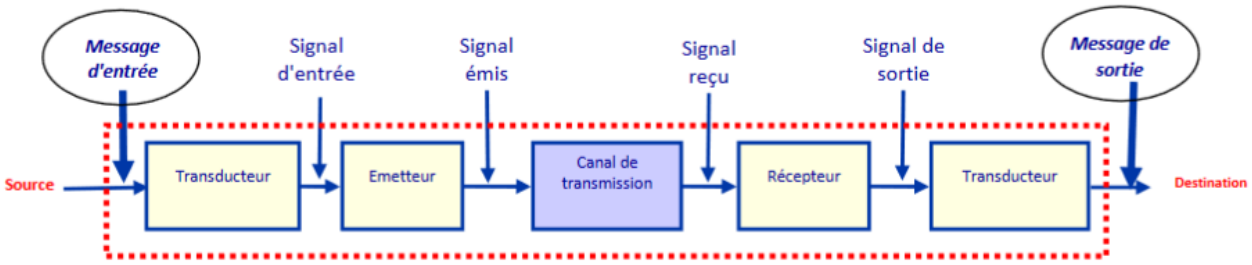

Figure 1 : Représentation d'une chaine de transmission

Pour effectuer cette étude, nous sommes composés d'un groupe de 5 personnes. L'avancée du projet se fait au travers de 6 étapes qui permettront de simuler pas à pas les différents blocs de la chaine de transmission.

L'étape 1 s'est déroulée entre le 13/09 et le 06/10. Elle consistait à réaliser une transmission élémentaire « back-to-back ».

L'étape 2 a été effectuée entre le 07/10 et le 17/10. Son but était d'effectuer une transmission non bruité d'un signal analogique.

Rédaction en cours, à compléter au fur et à mesure des séances.

ETAPE 1: TRANSMISSION « BACK-TO-BACK »

L'application réalisée pour l'étape 1 correspond à la figure 2 :

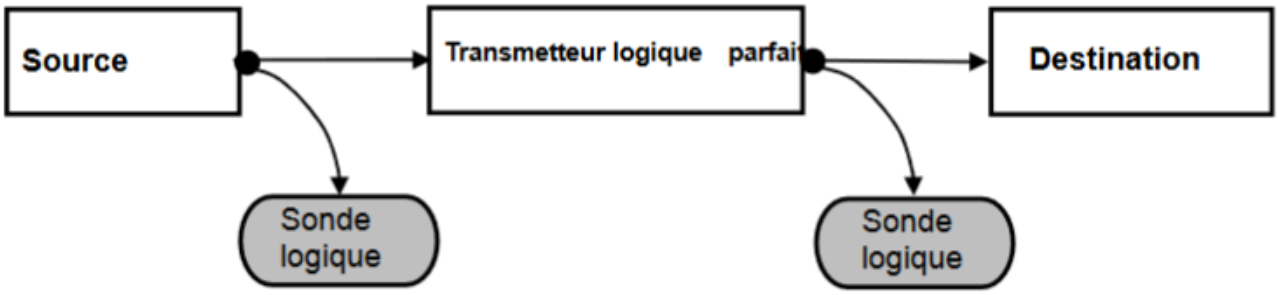

Figure 2 : Modélisation de la chaine de tranmission à l'étape 1

La source émet une séquence booléenne fixée ou aléatoire. Le transmetteur logique parfait se contente, à la réception d'un signal, de l'émettre tel quel vers les destinations qui lui sont connectées. La destination se contente de recevoir le signal du composant sur lequel elle est connectée. Des sondes logiques permettent de visualiser les signaux émis par la source et le transmetteur parfait. L'application principale calcule le taux d'erreur binaire (TEB) du système.

Le résultat de l'étape 1 est disponible sous forme du livrable 1 dans lequel figure les éléments suivants :

- Un fichier readme.txt qui donne la marche à suivre pour exécuter les différents programmes,
- 3 fichiers de scripts bash permettant d'exécuter les fichiers de code java,
- Les fichiers de code java réalisant les fonctionnalités du cahier des charges.

Afin de vérifier le fonctionnement de l'étape 1, nous avons réalisés des tests (illustrés par la figure 3). Ces derniers nous ont permis de mettre en évidence la conformité du travail effectué par rapport au cahier des charges. Ainsi, pour l'ensemble des transmissions simulée, nous obtenons un TEB de 0 (le transmetteur logique étant parfait et comme il n'y a aucune source d'erreur, comme du bruit par exemple, le résultat est cohérent).

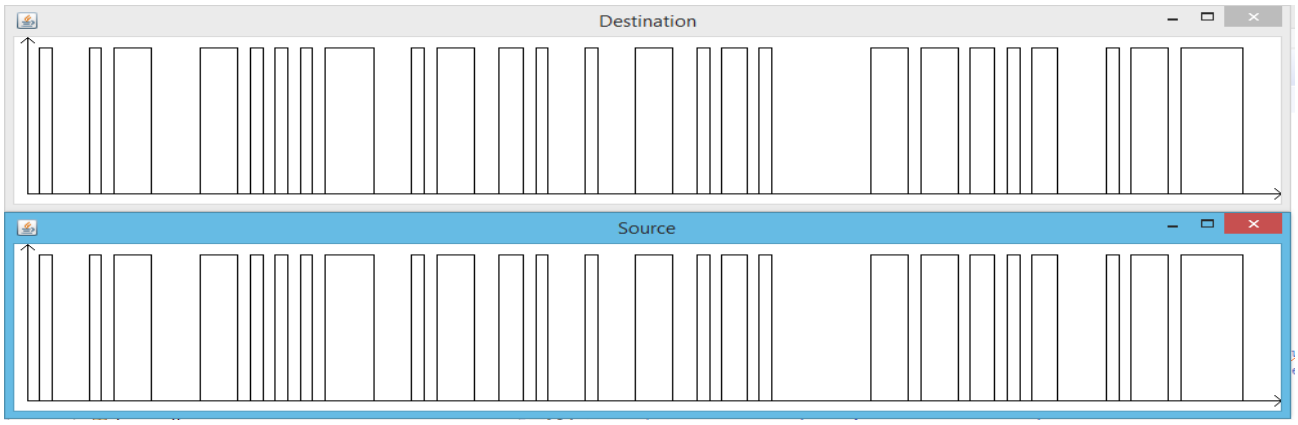

Figure 3 : Résultat de la simulation : java Simulateur -s => TEB : 0.0

Cette étape a été réalisé séparément par les 2 groupes (Johann et Marc d'un côté, Lim Kévin, Arouna et Lahoucine de l'autre). C'est pour cette raison que nous ne détaillerons pas plus la réalisation de cette étape puisque le livrable a déjà été rendu par chacun des 2 groupes.

A la suite de cette étape, les 2 groupes ont été fusionnés et une organisation de travail a été mise en place pour la suite du projet. Cette organisation est décrite dans la partie suivante.

ORGANISATION DU TRAVAIL D'EQUIPE

L'organisation est un point essentiel dans ce projet. Nous sommes un groupe de 5 et nous devons donc nous répartir les rôles pour que chacun soit efficace. Pour respecter au mieux les délais et les tâches nous avons réparti des responsabilités pour ce projet :

Chef de projet - Responsable livrables : Marc Lemauviel

Responsable développement : Johann Corcuff

Responsable télécom : Lahoucine Ahmouche

Responsable validation : Arouna Kane

Responsable qualité : Lim Kévin Chao

L'objectif principal est que chacun d'entre nous acquière des compétences dans les différents domaines du projet (Info et Telecom). Pour cela nous ferons régulièrement des séances de travail en commun pour que chacun puisse comprendre les différentes parties du projet. Nous réaliserons aussi des documentations (notamment de la javadoc) tout au long du projet pour les différentes étapes du projet.

Afin de garantir un bon niveau de qualité de notre travail et notamment pour notre code, nous appliquerons des tests pour nos itérations. La réalisation de ces derniers se fera au travers de JUnit qui est un outil simple à maitriser et qui nous permettra d'avancer rapidement sur nos tests. Nous mettrons également un point d'honneur sur la structure et la forme du code afin d'en faciliter la lecture autant que possible.

La rédaction du rapport sera la responsabilité d'une seule personne (le chef de projet) afin de garder la même ligne de conduite.

Le travail étant réalisé en groupe, la mise en commun du code est essentielle. Nous avons choisi d'utiliser un gestionnaire github pour mettre en commun nos différents travaux. Cela nous permettra de travailler en simultané sur des parties de code différentes.

BE – TEST ET VALIDATION

Les tests sont réalisés pour valider le contrat public des différentes classes et s'assurer du bon fonctionnement du programme au regard du cahier des charges conçu en amont du projet.

Les tests ont différents objectifs :

- Valider les fonctionnalités définies dans le cahier des charges établi par le client,
- Détecter de potentiels bugs et les corriger,
- S'assurer de la qualité du programme développé,

Quelles parties peut-on vraiment tester avec un programme de test ?

Les parties que l'on peut vraiment tester avec un programme de tests sont :

- Les contrats publics que sont censés remplir les différentes classes (tests unitaires),
- Le bon fonctionnement global du programme (tests d'intégrations).

Comment améliorer la testabilité ?

L'idéal est de coder les tests avant l'implémentation des différentes classes. Cela nécessite d'avoir une architecture où les contrats publics des différentes classes sont déjà définis. Dans la réalité, il se peut que l'architecture évolue ce qui peut amener à modifier les tests précédemment réalisés.

Comment faire du test de non régression à chaque nouvelle itération du projet ?

Pour cela il faut tester les parties du code les plus stables ainsi que les tests sur le fonctionnement général du programme qui ne sont pas censés changer au cours des itérations. Si du code mutable est mélangé avec du code stable il est bon de mettre une couche d'abstraction entre les deux. Pour cela on peut utiliser une interface, une classe abstraite ou des design patterns comme le pattern delegation.

Quelle confiance accorder aux résultats de simulation, et comment accroître cette confiance ?

La confiance accordée aux résultats de simulation dépend de la manière dont ils ont été réalisés. Si c'est le même développeur qui code à la fois les tests et la fonctionnalité correspondante, on pourra alors lui accorder une confiance moindre que si ces deux parties avaient été réalisées par deux personnes différentes (regards croisés). C'est pour cela que les entreprises possèdent des personnes spécialisées dans les tests et validation des programmes qui sont indépendantes des équipes de développement afin d'accroître cette confiance dans les résultats.

Comment comparer des simulations lorsqu'elles reposent sur des comportements aléatoires (message, bruit, …) ?

Pour tester les simulations malgré des comportements aléatoires, on peut utiliser des outils statistiques permettant de mesurer un degré de corrélation suffisant entre une entrée et la sortie attendue (calcule du Taux d'Erreur Binaire, TEB). La prédétermination théorique du TEB (en utilisant MATLAB par exemple) et sa comparaison avec le TEB calculé dans le programme est un bon moyen de vérifier la véracité du code réalisé.

ETAPE 2 : TRANSMISSION ANALOGIQUE (NON BRUITEE)

La seconde étape du projet reprend le travail effectué lors de l'étape 1. Cependant nous y ajoutons un émetteur, un récepteur et changeons le canal de transmission (analogique) comme indiqué dans la figure 4 :

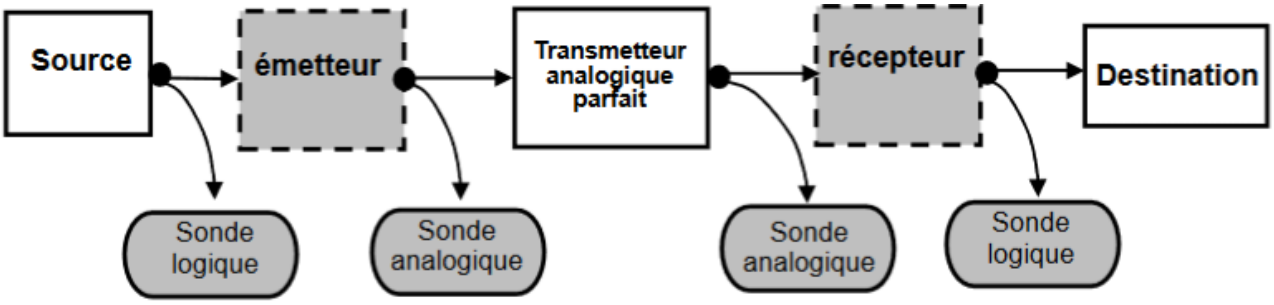

Figure 4 : Modélisation de la chaine de transmission à l'étape 2

L'ajout de l'émetteur et du récepteur permet de réaliser la conversion « logique => analogique » puis « analogique => logique ». Grâce à cela, le signal peut être émis au travers d'un canal analogique (dans notre cas, c'est le transmetteur analogique parfait).

Cela correspond à ce qu'il se passe dans la réalité. Nous disposons d'un signal numérique (composé de 0 et de 1 logiques) que nous souhaitons transmettre au travers une chaine de transmission. Pour ce faire, nous convertissons ce message logique en un signal analogique grâce à l'émetteur. L'intérêt de cette pratique est de pouvoir adapter le signal au canal de transmission (hertzien, guidé optique, guidé électrique, …). Au bout du canal de transmission, le récepteur reçoit le signal analogique puis le converti en signal numérique (le but étant que le message en sortie soit identique à celui en entrée).

La transmission de l'information se fait au travers de l'utilisation de l'un de ces 3 signaux dont les caractéristiques sont décrites ci-après :

- NRZ : forme d'onde rectangulaire,
- NRZT : forme d'onde trapézoïdale (temps de montée ou de descente à 1/3 du temps bit),
- RZ : forme d'onde impulsionnelle (amplitude min sur le premier et dernier tiers du temps bit, impulsionnelle sur le tiers central avec un max au milieu du temps bit égal à l'amplitude max).

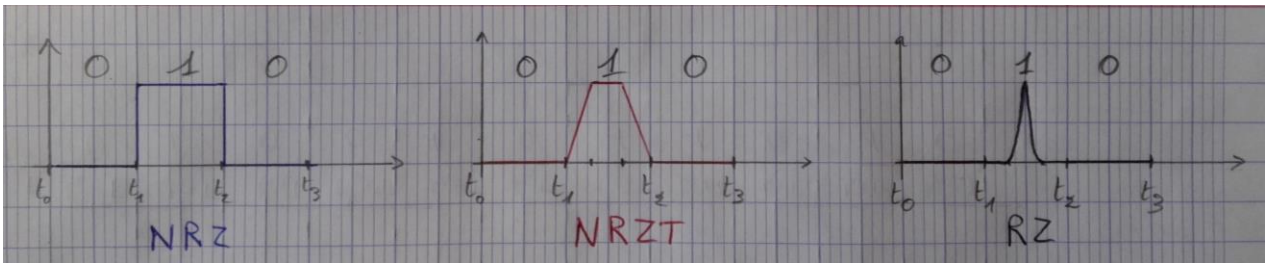

Figure 5 : Illustrations des signaux NRZ, NRZT et RZ

La réalisation de l'étape 2 s'est déroulée lors de sessions de partages sur notre temps de travail personnel. Afin d'optimiser notre gestion des ressources, nous avons mis en place un dépôt Github. Grâce à cela, chacun de nous pouvait travailler sur le projet indépendamment des autres personnes. Toutefois, nous avons dû gérer l'encodage des fichiers java sous les différents OS (Windows et Linux) ainsi que la présence de fichiers liés aux configurations individuelles de nos IDE.

D'un point de vue implémentation, nous sommes tout d'abord partis sur une première architecture où nous utilisions l'héritage afin de spécialiser les émetteurs et les récepteurs selon le type de signal transmis (NRZ, NRZT ou RZ). Cela nous a amené à avoir un grand nombre de classes et une relative redondance du code. C'est pourquoi nous avons opté pour une seconde architecture utilisant le pattern delegation. Nous avons ainsi créé le package « codeur » qui contient des encodeurs et des décodeurs pour chacun des types de signaux. Il suffisait ensuite de passer en argument l'encodeur (respectivement le décodeur) désiré à l'émetteur (au récepteur). Ainsi nous n'avions plus qu'une seule classe émetteur (et récepteur) ce qui nous a permis de factoriser le code. Néanmoins, nous avons dû modifier nos tests unitaires afin de les adapter à cette nouvelle architecture.

Afin de reconnaitre quel est le symbole reçu par le récepteur, nous avons effectué une opération de comparaison entre un seuil et la moyenne des échantillons pour un même temps bit.

Le seuil est déterminé en réalisant l'opération suivante : $Seuil = \frac{Moyenne\ de\ toutes\ les\ valeurs\ du\ signal}{Nombre\ d'échantillon total}$

Nous calculons ensuite la moyenne des échantillons pour un temps bit en suivant les principes suivants :

- Signal NRZ : moyenne totale des échantillons pour un temps bit (30 échantillons par défaut),
- Signal NRZT : moyenne des échantillons correspondant au tier du temps bit central (les 10 échantillons centraux par défaut),
- Signal RZ : moyenne des échantillons correspondant au tier du temps bit central (les 10 échantillons centraux par défaut). Une évolution sera peut-être à apporter pour ne garder que les 4 ou 5 échantillons vraiment intéressant au centre.

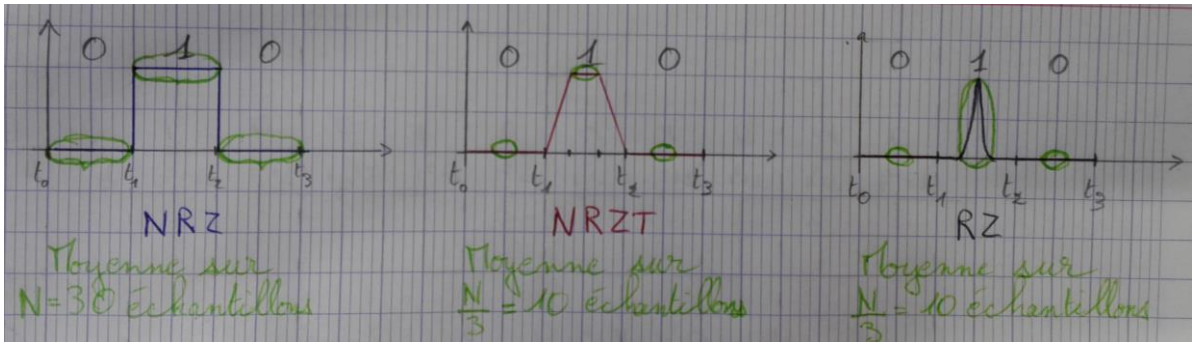

Figure 6 : Illustration de la méthode de moyennage pour la réception des signaux

Enfin, nous comparons la moyenne calculée sur le temps bit avec le seuil pour déterminer la nature du symbole reçu.

Au niveau de la validation, nous avons voulu tester si les signaux produits par les encodeurs à partir d'une suite binaire connue correspondaient bien aux exigences du cahier des charges. Nous avons donc analysé si le signal en sortie de l'encodeur puis du décodeur correspondait bien aux attentes.

Voici les différents résultats que nous avons obtenus :

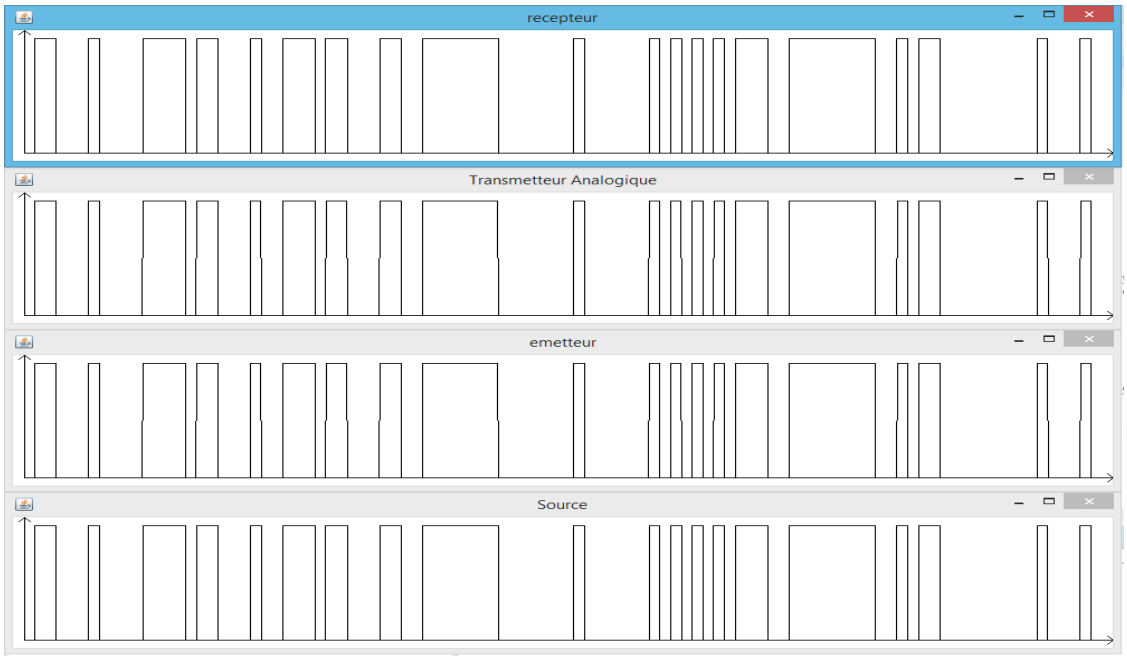

Figure 7 : Résultat de la simulation : java Simulateur -s -form NRZ => TEB : 0.0

La forme d'onde qui résulte du signal NRZ correspond à une forme rectangulaire générée par une impulsion.

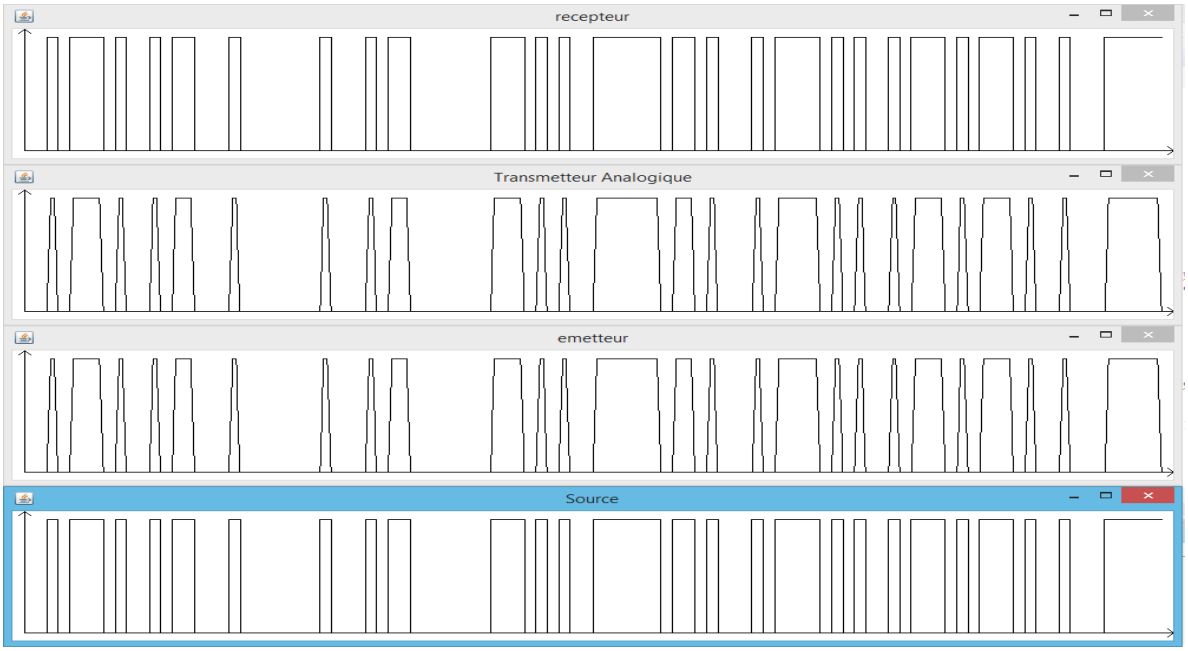

Figure 8 : Résultat de la simulation : java Simulateur -s -form NRZT => TEB : 0.0

La forme d'onde du signal NRZT est similaire à celle du signal NRZ mais cette fois ce ne sont plus des rectangles mais des trapèzes qui sont observés. Les temps de montée et de descente correspondent à 1/3 du temps bit ce qui donne cette forme trapézoïdale.

Figure 9 : Résultat de la simulation : java Simulateur -s -form RZ => TEB : 0.0

La forme d'onde du signal RZ correspond à une impulsion sur le tier central du temps bit (le premier et le dernier tier étant à la valeur min) avec un max atteint à la moitié du temps bit.

Les différentes formes d'ondes correspondent donc bien aux attentes exigées dans le cahier des charges.

De plus, nous avons modifié le script bash demo.sh afin de présenter les nouveaux modes de transmissions analogiques du signal conçus lors de cette étape.

Enfin, tout comme pour l'étape 1, nous obtenons un TEB de 0 ce qui correspond à une transmission parfaite. En effet, la chaine de transmission utilise un canal parfait pour le moment. Lors de l'étape 3, nous modifierons le canal de transmission pour ajouter un bruit blanc additif gaussien ce qui entrainera des variations du TEB.

ETAPE 3 : TRANSMISSION NON-IDEALE AVEC CANAL BRUITE DE TYPE « GAUSSIEN »

La troisième étape du projet reprend le travail effectué lors des deux précédentes étapes. Cependant nous modifions le transmetteur analogique parfait pour y introduire un bruit blanc additif gaussien. La figure 4 représente la nouvelle modélisation de la chaine de transmission :

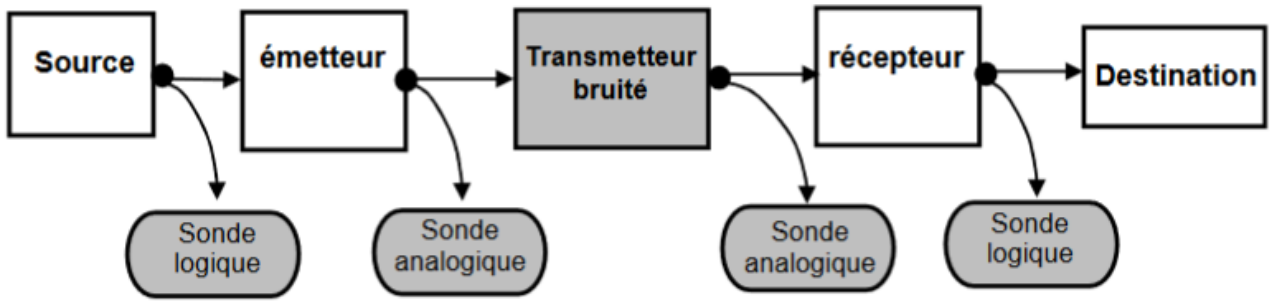

Figure 10 : Modélisation de la chaine de transmission à l'étape 3

Un bruit blanc correspond à la réalisation d'un processus aléatoire dans lequel la densité de puissance est la même pour toutes les fréquences de la bande passante. Le bruit blanc gaussien correspond à un bruit blanc qui suit une loi normale de moyenne et de variance données.

On utilise ce bruit blanc pour modéliser un canal de transmission en télécom. En effet tous les canaux de transmissions ajoutent du bruit au signal émis. Ainsi lorsque l'on reçoit le signal, du bruit s'est ajouté au signal émis. Le bruit blanc n'est qu'un type de perturbation qui se passe dans la nature, nous en verrons d'autres dans les prochaines étapes.

La figure 11 illustre la modélisation du canal de transmission auquel nous ajoutons un bruit blanc gaussien. A la réception, nous obtenons donc le signal émis s(n) plus le bruit b(n) avec b(n) qui suit une loi normale de moyenne nulle et de variance σ_b^2.

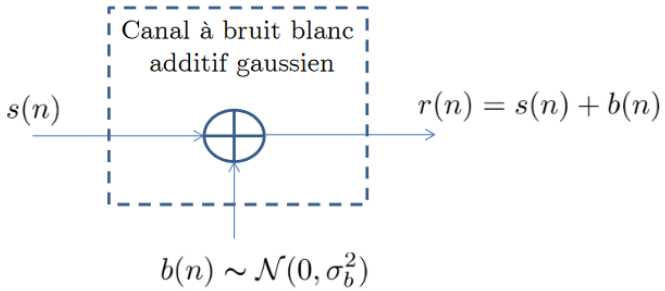

Figure 11 : Modélisation du canal de transmission avec un bruit blanc additif gaussien

La première étape de l'implémentation a donc consisté à réaliser ce bruit blanc gaussien. Pour cela nous avons utilisé la formule suivante :

$$b(n) = \sigma_b \sqrt{-2\ln(1 - a_1(n))}\cos(2\pi a_2(n)) \qquad \begin{matrix} a_1(n) \sim \mathcal{U}[0,1[\text{ (loi uniforme)} \\ a_2(n) \sim \mathcal{U}[0,1[\end{matrix}$$

Figure 12 : Formule pour générer un bruit blanc gaussien

Afin de vérifier que nous générions bien un bruit blanc gaussien, nous avons utilisé Matlab pour réaliser un histogramme et le comparer à une Normale de même paramètre : $N(0, \sigma_b^2)$. Voici la courbe que nous avons obtenue :

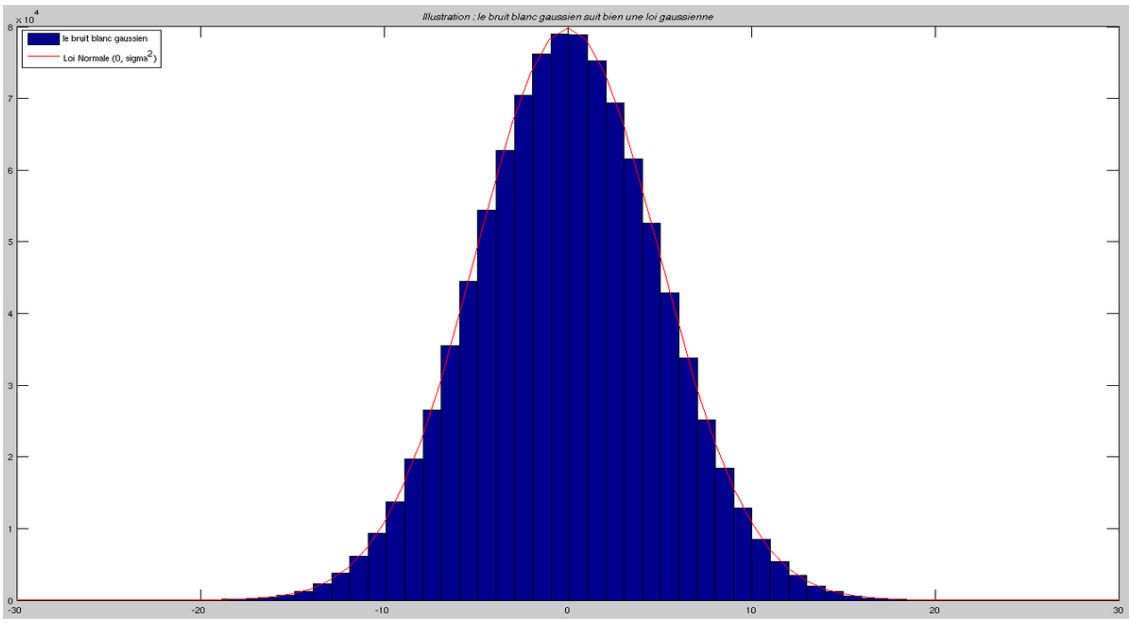

Figure 13 : comparaison du bruit blanc généré par rapport à la loi Normale $N(0, \sigma_b^2)$

Suite à l'implémentation du bruit, nous avons ajouté ce bruit à l'information émise. Lors de la première réalisation de l'étape nous avions eu un problème par rapport au bruit. Ce dernier était trop important par rapport au SNR rentré. Le calcule n'était pas bon, il manquait un rapport 10 entre les 2. Suite à la modification, les résultats sont désormais plus cohérents :

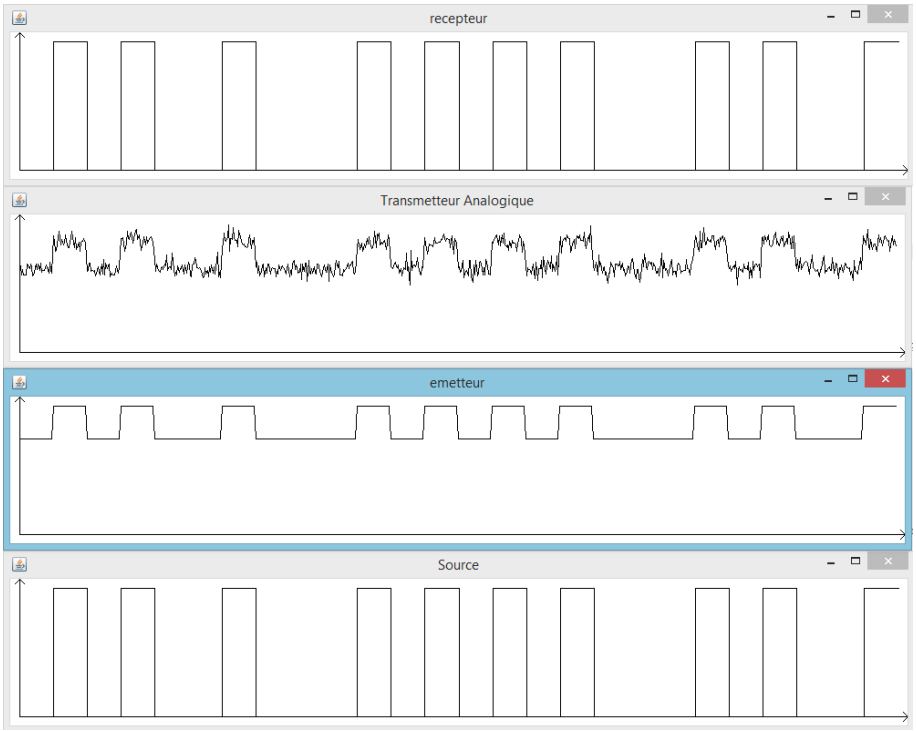

Figure 14 : Résultat de la simulation : java Simulateur -ampl 3 4 -s -form NRZ -mess 0101001000101010100010100 1 -snr 15 => TEB : 0.0

Pour ce même message, on obtient un TEB de 0 pour un signal NRZT et de 0,15 pour un signal RZ. Cela nous indique que le signal RZ est moins performant que les deux autres signaux pour récupérer les informations bruitées.

Afin d'approfondir les tests, nous avons testé la transmission de messages plus importants (100 000 bits). Le tableau suivant indique le TEB obtenu en fonction du signal utilisé et du nombre de bits dans le message. Pour ces simulations, nous avons utilisés les paramètres –ampl 3 4 et –snr 10.

	Signal NRZ	Signal NRZT	Signal RZ
1 000 bits	0	0	0.06306306
10 000 bits	0	0	0.06920692
100 000 bits	1.00001E-5	1.300013E-4	0.06319063

De ce tableau nous pouvons en déduire que le signal NRZ semble être le plus efficace lors de transmissions bruitées. Cela correspond à nos attentes puisque la durée d'émission (pour un bit à 1) se fait sur tout le temps bits et n'est pas découpée en 2 ou 3 parties.

Afin de mieux visualiser l'évolution du TEB, nous avons tracé la courbe du TEB en fonction du SNR par bit (Eb/N0). Pour ce faire, nous avons utilisé la formule suivante :

$$\left(\frac{Eb}{N0}\right)(dB) = 10\log\left(\frac{Ps * N}{2\sigma b^2}\right) = 10\log(\frac{Ps}{\sigma b^2}) + 10\log(\frac{N}{2}) = SNR\ (dB) + 10\log(\frac{N}{2})$$

Avec :

- Eb : énergie par bit (Joule: Watt. sec)
- N0 : densité spectrale du bruit (Watt/Hz)
- P_s : puissance du signal
- σ_b^2 : puissance bruit
- N : Nombre d'échantillon par bit

Pour obtenir cette courbe, nous effectuons plusieurs simulations en faisant varier le SNR entre -20 et +15 dB par pas de 0,1. De plus, nous effectuons 100 fois chaque simulation avec le même SNR pour obtenir une valeur moyenne de TEB plus précise. Nous avons réalisé cette simulation avec le signal NRZT pour une longueur de message de 1 000 bits. Pour des problèmes de performances sur les machines linux et l'école, nous n'avons pas essayé de monter au-delà de 1 000 bits.

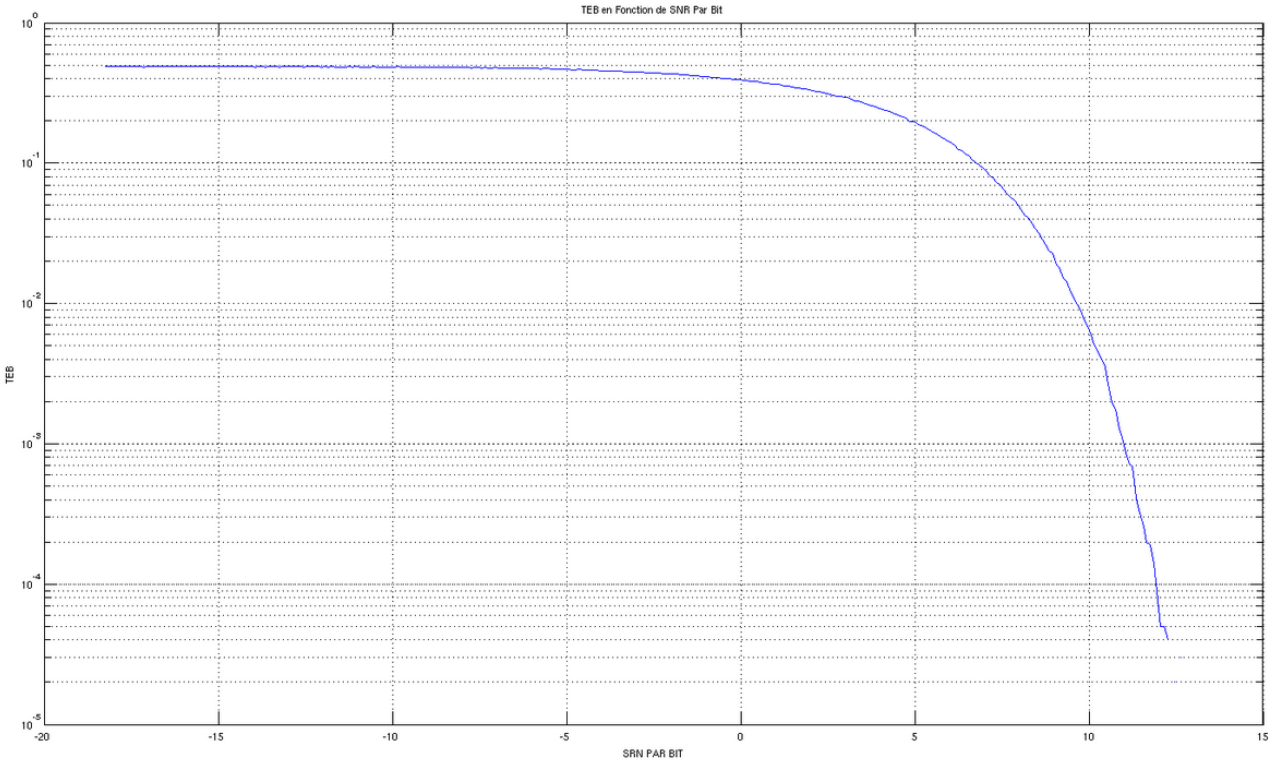

Figure 15 : Evolution du TEB en fonction du SNR pour le signal NRZT

On constate que pour un SNR inférieur à 0, le TEB est d'environ 0,5 soit la plus mauvaise valeur possible. Lorsque le SNR devient positif et proche de 10 dB, les valeurs du TEB avoisinent les 10^{-2} et 10^{-3}. Ces valeurs correspondent à nos attentes. Cependant nous avons une anomalie puisque nous envoyons un message de 1 000 bits mais nous constatons que la courbe descend jusqu'à 10^{-4} ce qui n'est pas possible.

Lors de l'étape 2 nous avions rencontré un problème de performance. Afin de le résoudre, nous avons remplacé les linkedlist dans la classe information par des arraylist qui sont plus performantes. Nous avons ainsi réduits le temps d'exécution d'une trentaine de secondes pour transmettre un message de 10 000 bits (avec l'affichage) à seulement 2 ou 3 secondes après la modification.

Les résultats obtenus à l'étape 3 sont concluants, le calcul du TEB semble correct et son évolution en fonction du SNR correspond à nos attentes. Lors de l'étape suivante, nous ajouterons, en plus du bruit blanc gaussien, perturbations « réels » telles que : des trajets-multiples, de la dispersion chromatique, du bruit électronique, etc. Autant de phénomènes qui sont rencontrés dans la nature et qui impactent les transmissions télécom.

ETAPE 4 : TRANSMISSION NON-IDEALE AVEC DIVERS BRUITS « REELS » PLUS LE BRUIT GAUSSIEN DE L'ETAPE 3

L'objectif de l'étape 4 est d'appliquer des modèles physiques sur notre chaine de transmission. Nous allons donc ajouter un phénomène de trajets-multiples tout en ajoutant le bruit blanc gaussien que nous avons réalisé à l'étape précédente. La figure 16 illustre les modifications que nous allons effectuer.

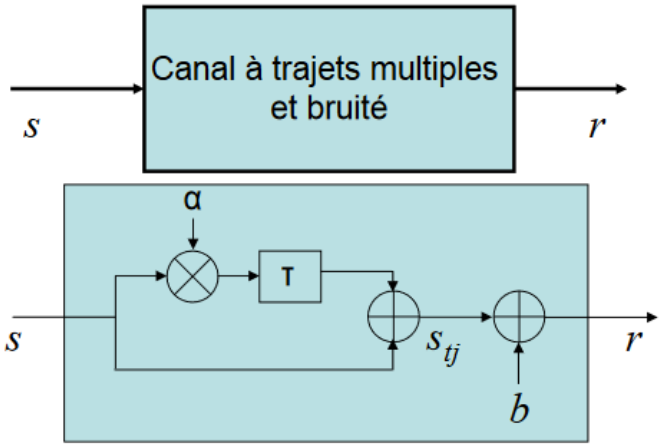

Figure 16 : Ajout d'un phénomène de trajets-multiples et du bruit blanc gaussien

La réalisation du trajet multiple consiste à reprendre le signal émis auquel nous ajoutons un retard T (en nombre d'échantillon) et une atténuation α (comprise entre 0 et 1). Voici un exemple d'un trajet multiple :

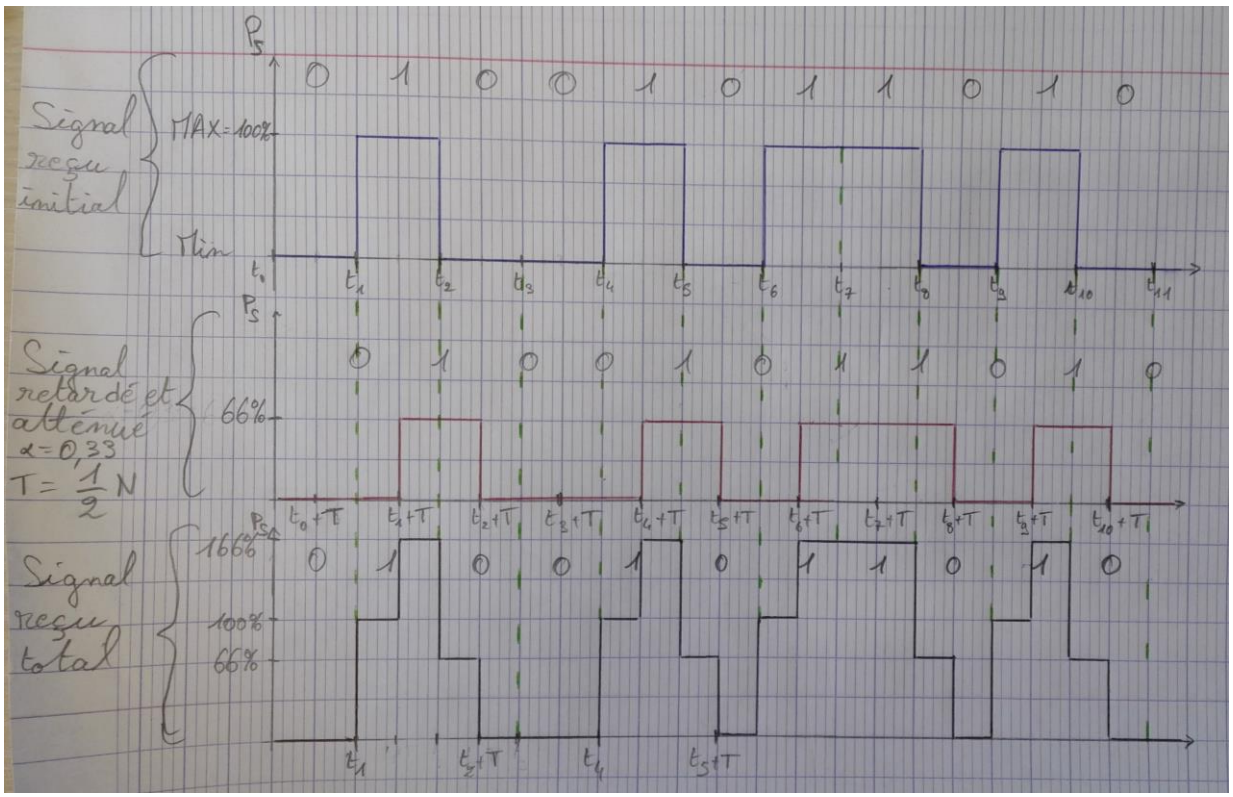

Figure 17 : Illustration d'un trajet multiple pour le signal NRZ avec α = 0,33 et T = N/2

Comme nous pouvons le constater, le retard provoqué pour le trajet multiple fausse le signal reçu par le récepteur. Il faut donc réussir à isoler l'information utile du signal pour ne pas prendre en compte le retard.

Nous avons tout d'abord tenté d'implémenter un code matlab pour simuler un filtre adapté. Cependant cette méthode n'est pas complètement fonctionnelle (le développement est toujours en cours).

Pour remplacer cette solution, nous avons gardé notre récepteur dont le seuil de décision s'adapte parfaitement à l'atténuation du trajet retardé. Toutefois lorsque le retard est équivalent à un temps bit (30 échantillons par défaut), notre récepteur n'est plus capable de faire la différence entre le signal utile et le signal retardé.

On constate également qu'en utilisant un retard de 20 échantillons, le signal NRZ est beaucoup moins performant que les signaux NRZT et RZ qui sont moins perturbés par ce retard en raison de la prise de décision qui se fait sur la moyenne des 10 échantillons centraux du temps bit. Les figures 18 et 19 illustrent cette différence :

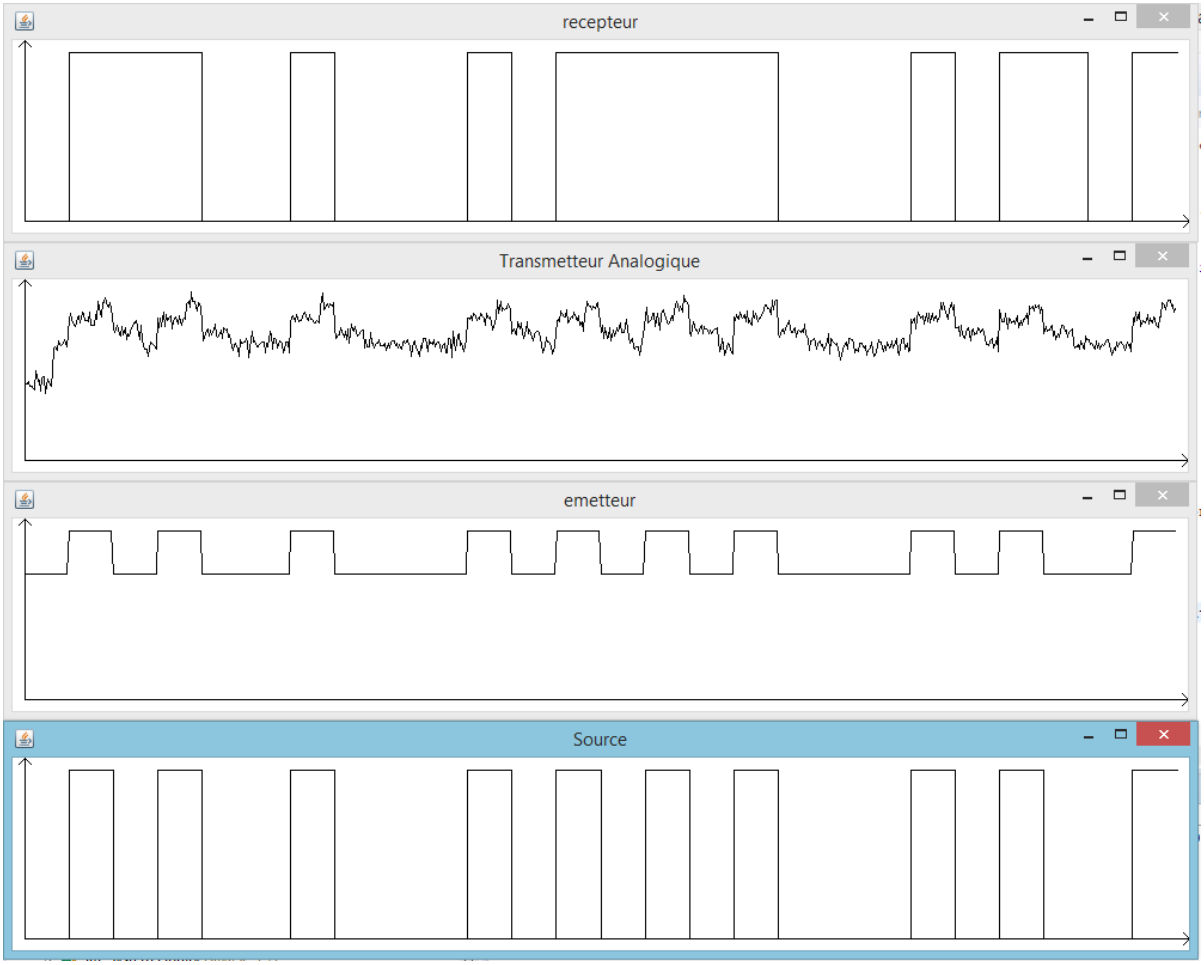

Figure 18 : Résultat de la simulation : java Simulateur : -ampl 3 4 -s -form NRZ -mess 0101001000101010100010100 1 -snr 15 -ti 20 0.5 => TEB : 0.0

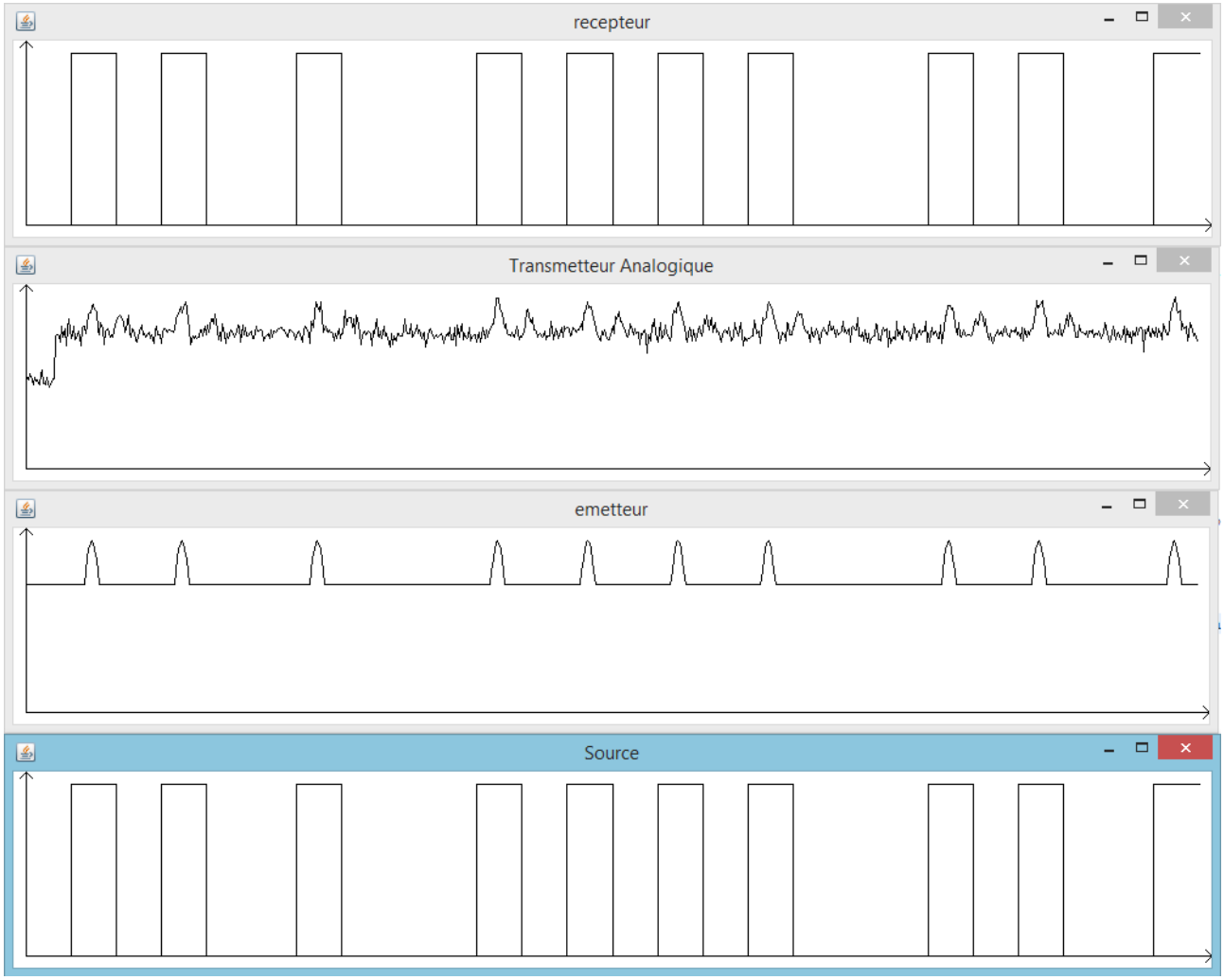

Figure 19 : Résultat de la simulation : java Simulateur -ampl 3 4 -s -form RZ -mess 0101001000101010100010100 1 -snr 15 -ti 20 0.5 => TEB : 0.0

La solution de secours n'est que temporaire en attendant de finaliser le code matlab qui nous permettra de simuler un filtre adapté. Pour le moment le code matlab fonctionne pour des amplitudes symétriques (max à +2 et min à -2 par exemple).