REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Hadj Lakhdar – Batna

Faculté des Sciences

Département d'Informatique

THESE

Présentée par

Hacene ZIDANI

Pour obtenir le grade de

Magistère

Spécialité : **I**ngénierie des **S**ystèmes **I**nformatiques (**ISI**)

Etude et optimisation des protocoles de transport pour les nano satellites

---

Soutenue publiquement le …. /…. /…… devant le jury formé de :

Dr. Brahim BELATTAR	M.C.	**Président**	Université de Batna
Dr. Abdelmadjid ZIDANI	M.C.	**Rapporteur**	Université de Batna
Pr. Mohammed BENMOHAMMED	PROF.	**Examinateur**	Université de Constantine
Dr. Okba KAZAR	M.C.	**Examinateur**	Université de Biskra

Remerciements

C'est avec un grand plaisir que je réserve ces lignes en signe de gratitude et de reconnaissance à tous ceux qui ont contribué de près ou de loin à l'élaboration de ce travail.

Je veux exprimer ma gratitude et ma reconnaissance à mon encadrant : ***Mr. Abdelmadjid ZIDANI*** *qui n'a cessé de me prodiguer ses conseils judicieux et ses suggestions pertinentes.*

Mes remerciements les plus sincères s'adressent à tous les membres du jury qui ont bien voulu me faire l'honneur d'examiner et de juger mon travail. J'aimerais également exprimer ma gratitude à tous les professeurs de l'école doctorale SCTIC.

Enfin je tiens à remercier tous les membre de ma famille qui m'ont aidé, soutenu de leur amour durant mon projet de magistère. Tous mes meilleurs amis et mes collègues de l'école doctorale avec lesquels j'ai passé des bons moments. Que tous ceux qui, de près ou de loin, ont contribué à l'aboutissement de ce travail soient assurés de ma profonde gratitude.

Résumé

Les réseaux d'accès satellitaires *Low Earth Orbiting* (LEO), qui sont à la fois sans fil et mobile, ont un ensemble unique d'erreurs de lien, y compris la corruption de bits, le handover et la connectivité limitée. Malheureusement, la plupart des protocoles de transport sont uniquement destinées à gérer la congestion liée à des erreurs communes dans les réseaux filaires. Cette incapacité de gérer plusieurs types d'erreurs résulte une dégradation grave dans le débit effectif et l'énergie stockée, qui sont les deux paramètres essentiels d'un environnement sans fil et mobiles.

Des recherches proposent un nouveau protocole de transport pour les satellites appelé *Satellite Transport Protocol* (STP), qui s'adresse aux problèmes des réseaux satellitaires seulement. STP ne fait pas une différentiation entre les types d'erreur qui se produisent. Pour cela, d'autres études ont intégré au protocole STP un mécanisme de contrôle d'erreur qui s'appelle *probing* pour le rendre plus réactif aux conditions d'erreur qui prévaut dans les réseaux satellitaires. Le mécanisme fonctionne en investissant le temps et la transmission afin de déterminer la cause de l'erreur. Ces overheads sont, cependant, récupérés par des gains dans le débit effectif de la connexion et son efficacité énergétique. Le protocole STP avec le nouveau mécanisme *probing* est appelé : *eXtended Satellite Transport Protocol* (XSTP). Dans notre étude nous intéressons aux performances du protocole XSTP pour les réseaux d'accès nanosatellitaires. L'étude est faite en implémentant ce protocole dans le simulateur réseau NS2 et en faisant des simulations, puis nous comparons XSTP avec les variantes du protocole TCP (clones TCP).

Mots clés : Protocole de transports, Réseau satellitaire, LEO, clones TCP, STP, XSTP, mécanisme probing.

Abstract

The design of efficient communication mechanisms for small satellite networks is a challenging task, requiring the definition and implementations of specific protocols and architectures appropriate to space's critical conditions. In this work, we have selected XSTP (*eXtended Setellite Transport Protocol*) as candidate protocol for nanosatellite networks. Foremost, we implemented XSTP in NS2 simulator. The initial simulations were done in a LEO satellite network scenario. According to our simulations, XSTP was shown to reach a higher effective throughput, much lower overhead, and better channel efficiency as compared to TCP clones, in case of high BER conditions. As to low BER environment, all the protocols have a comparable performance in terms of channel efficiency.

Keywords : Transport protocol, Satellite network, LEO, TCP clones, STP, XSTP, Probing mechanism.

Tables des Matières

Liste des Figures

Liste des Tableaux

Liste des Acronymes

GEO	Geostationary Earth Orbiting
LEO	Low Earth Orbiting
MEO	Middle Earth Orbiting
HEO	Hlliptic Earth Orbiting
GNC	Guidance, Navigation and Control
GPS	Global Positioning System
BER	Bit Error Rate
ACK	Acknowledgment
ARQ	Automatic Repeat Request
SACK	Selective Acknowledgment
NAK	Negative Acknowledgment
SNACK	Selective Negative Acknowledgment
RTT	Round Trip Time
ISO	International Organization for Standardization
CCSDS	Consultative Committee for Space Data Systems
OSI	Open Systems Interconnection
ATM	Asynchronous Transfer Mode
UNI	User Network Interface
NNI	Network Network Interface
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
IP	Internet Protocol
SCPS	Space Communications Protocol Standards
SCPS-TP	SCPS Transport Protocol
XTP	Xpress Transport Protocol
SSCOP	Service Specific Connection Oriented Protocol
NSP	Nanosatellite Protocol
STP	Satellite Transport Protocol
XSTP	eXtended Satellite Transport Protocol
SD	Sequenced Data
STAT	Status
USTAT	Unsolicited STAT
FTP	File Transfer Protocol
MSS	Maximal Segment Size
HTTP	HyperText Transfer Protocol
WWW	World Wide Web
NS2	Network Simulator 2
OTcl	Object Tool Command Langage

CHAPITRE I

INTRODUCTION

La communication a toujours été vitale pour la croissance et le développement de la société humaine. Les sociétés cherchent toujours à trouver des nouveaux moyens de communication. Aujourd'hui avec les réseaux informatiques et de l'évolution de l'Internet le monde est transformé en un seul village global. Cependant, l'explosion de l'e-communauté a suscité la nécessité d'un paradigme de communication plus global; celui qui assure la connectivité à la maison, au travail ou en déplacement. Cette exigence a permis d'orienter la recherche dans la direction du sans fil. Les premières tentatives à l'évolution des réseaux de données sans fil, où les utilisateurs de l'Internet en mouvement peuvent communiquer librement sans la nécessité d'une installation filaire. Bien que ces tentatives aient été un grand pas dans la bonne direction, ils n'ont pas l'échelle pour certaines applications nécessitant un déploiement rapide et une configuration automatique. Comme une logique de succession, une attention est orientée vers les réseaux mobiles *ad hoc*, où des groupes des appareils mobiles communiquent entre eux sans avoir besoin d'une configuration fixe du réseau.

Les réseaux de données satellitaires sont un exemple typique de réseaux mobiles *ad hoc*. Les réseaux satellitaires ont permis une nouvelle classe d'applications en connectant des réseaux de télécommunication, l'autonomisation des réseaux cellulaires, la transmission des signaux radio et vidéo et la collecte des informations. Dans ce mémoire l'accent est mis sur les réseaux contenant des liaisons satellites LEO, qui ont des caractéristiques différentes de leurs homologues traditionnels GEO. Les satellites LEO sont soumis à la réduction des pertes de transmission et à un délai de signaux plus court. En outre, les satellites LEO sont relativement bon marché, ce qui en fait une solution économique pour la connectivité Internet dans les régions du monde autrement handicapé par l'installation coûteuse des infrastructures terrestre. Les satellites LEO sont également idéals pour l'expérimentation, en particulier à une époque où les fonds de recherche sont faibles et la possibilité d'un échec est assez élevée.

Alors que beaucoup de recherches ont porté sur le développement des protocoles de routage pour les réseaux satellitaires, ce mémoire concentre principalement sur l'étude d'une optimisation d'un protocole de transport sur les réseaux contenant des liaisons satellites LEO. Bien que les liaisons sans fil et satellitaires partagent beaucoup de caractéristiques communes, comme le haut BER et la connectivité intermittente, ils ont aussi assez de propriétés distinctes qui doit être pris en compte comme des environnements différents pour le transport des données. Ces propriétés incluent un BER très variable, capacité des liens montants et descendants disproportionnée, des ressources de calcul limitées (puissance, mémoire et vitesse), et relativement un débit plus élevé.

I.1. Réseaux satellitaires LEO

I.1.1. Architecture du réseau

Comme un réseau satellitaire LEO est à la fois *ad hoc* et *mobile*, sa topologie et sa configuration changent fréquemment. Le réglage manuel des protocoles de transport, qui est habituellement fait par les administrateurs réseau, ne suffit pas dans ce cas. L'infrastructure générale d'un réseau satellitaire LEO se compose de trois éléments principaux [10] :

- ***Terminaux :*** qui sont des dispositifs de calcul équipés de matériel d'émission des paquets radio, ce qui leur permet de communiquer directement avec les satellites quand ils entrent dans leur zone de couverture.
- ***Stations de base :*** qui agissent en tant que passerelles entre le satellite et les réseaux terrestres. Ils envoient des signaux pour contrôler la configuration du satellite.
- ***Satellites LEO :*** qui tournent autour de la Terre dans de multiples chemins et constellations.

Les réseaux satellitaires LEO sont souvent utilisés comme points d'accès à d'autres réseaux comme Internet. Dans ce cas, ils sont appelés les *réseaux d'accès satellitaires LEO*. Pour un terminal fixe dans Internet, un satellite LEO est visible comme un autre nœud Internet dans la couche IP. Une configuration typique d'un réseau d'accès satellitaire LEO est représentée dans la Figure I.1.

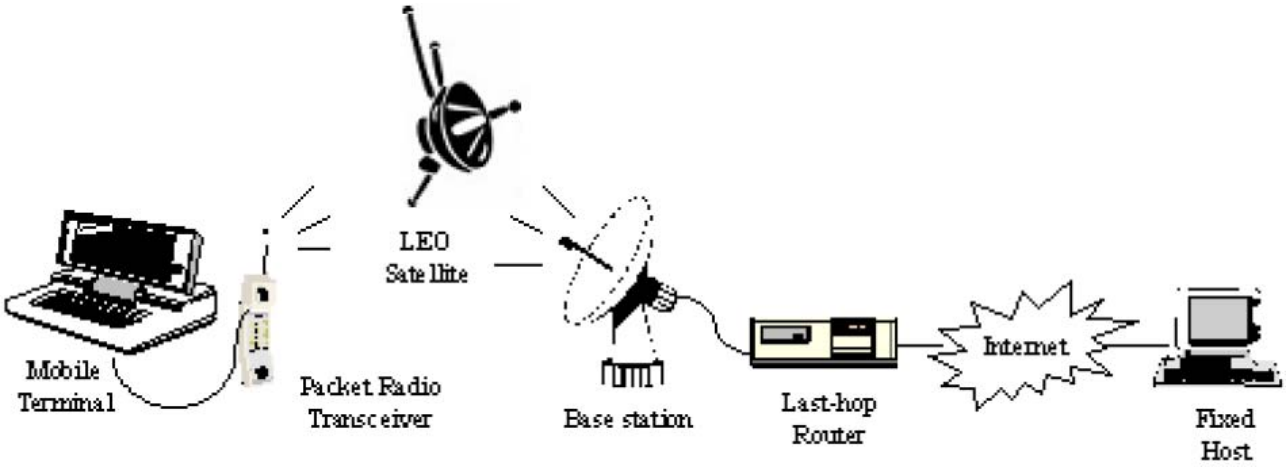

Figure I.1. Réseau d'accès satellitaire typique

I.1.2. Propriétés des liens

Un lien satellitaire est en fait un type particulier des liens sans fil, il hérite de toutes les caractéristiques d'un lien sans fil. Les liens satellitaires ayant une capacité naturelle de diffusion (*broadcast*) et d'une capacité inhérente à atteindre les utilisateurs mobiles, qui rendent les liaisons satellitaires idéales pour le remplacement des liaisons terrestres. Les liaisons satellitaires couvrent de vastes zones ce qui réduit les commutations et les transmissions d'*overhead*. Les liaisons satellitaires peuvent avoir une bande passante limitée en raison de la limitation naturelle du spectre radio et en raison de certains accords internationaux qui contrôlent l'affectation de cette ressource.

Les réseaux satellitaires sont aussi caractérisés par des liens asymétriques, généralement en raison du coût élevé de la technologie nécessaire pour soutenir les deux directions. Cette asymétrie est manifestée généralement dans la vitesse des liens, la bande passante ou les deux. Elle influe sur les performances des protocoles de transport qui utilisent des acquittements (ACKs) dans le sens inverse comme un mécanisme d'auto-synchronisation. Avec l'arrivée lente des ACKs, la vitesse de transmission de données diminue. Une autre restriction des satellites LEO est leurs ressources de calculs limités due aux limites de la puissance et la taille. Les restrictions de puissance, en particulier, peuvent affecter la stabilité de la connexion et la fréquence des erreurs, en mettant un frein à la performance globale d'un protocole de transport.

Semblables à d'autres liens sans fil, les liens satellitaires sont caractérisés par un taux d'erreur binaire (BER) élevé. Cet effet est principalement dû à plusieurs facteurs de l'environnement (pluie, pollution, etc) qui cause le bruit. Les contrôles d'erreur de codage avancés sont parfois utilisés dans la couche liaison pour atténuer ces problèmes. Il est encore difficile de cacher les effets secondaires de ces solutions sur les protocoles de transport. En fait, la plupart des protocoles de transport fiables ont une hypothèse que toute perte est due à la *congestion du réseau*.

Un autre problème pour les satellites LEO est leur connectivité intermittente en raison de leur mouvement dans leurs orbites. Un *handover* se produit quand un satellite LEO sort de la portée d'un dispositif d'accès par satellite et un autre entre dans la portée de ce dispositif. Il peut également se produire quand un satellite LEO sort de la portée d'une station de base et entre dans la portée d'une autre. La connectivité peut également être perdue pour des périodes longues en raison d'une obstruction physique du signal satellite ou d'une mauvaise répartition des satellites LEO par le fournisseur de services. Le *handover* et la connectivité limitée peut conduire à des périodes de *blackout*, au cours de laquelle tous les paquets envoyés sont perdus [10].

I.2. Transport sur les satellites LEO

Toutes les données des réseaux, y compris les réseaux d'accès satellitaires, sont gérées par des piles de protocoles de communication. Les performances de toute connexion est un reflet direct de l'ensemble des performances de tous les protocoles de la pile. Le protocole de transport est l'un des principaux protocoles dans toute pile de communication. Beaucoup de protocoles de transport standards (comme TCP), en général n'ont pas des caractéristiques spécifiques à leurs réseaux [10]. Ces protocoles sont à l'origine conçu pour résoudre les problèmes et de satisfaire les objectifs de transport des réseaux filaires. Par conséquent, ces protocoles supposent la continuité de la connectivité, perte de données résultant de la congestion du réseau et que les liens bi-directionnels sont équilibrés. Ces protocoles sont aussi calibrés afin de surmonter les problèmes de stabilité et de l'hétérogénéité en termes de tampon du récepteur, la bande passante du réseau et le délai. En outre, ces protocoles souci d'équité dans la consommation de la bande passante et l'efficacité de l'utilisation d'un lien par l'adoption des propres mécanismes de contrôle de congestion.

Les protocoles de transports visés par les réseaux satellitaires devraient s'efforcer de fournir un lien d'accès équitable, haut débit et une grande fiabilité grâce à la différenciation entre les conditions d'erreur. Les protocoles de transport devrait aussi utilisent la bande passante avec une politique de priorité, maximiser l'utilisation de lien et de fournir un service semi-fiable optionnel en cas de besoin [10]. Ces objectifs appellent à revoir les protocoles de transport standards et la possibilité de proposer de nouveaux protocoles.

I.2.1. Stratégie de contrôle d'erreur

Beaucoup d'études ont montré que la capacité d'un protocole de classer correctement la nature de l'erreur détectée peut faire toute la différence pour les performances d'un tel protocole [8]. Les protocoles qui n'ont pas la capacité de distinguer les erreurs risquent de prendre une position agressive, en réponse aux conditions de détérioration du lien, ou le gaspillage de bande passante en réponse aux rares erreurs transitoires. Ces deux situations conduisent à l'inefficacité de l'utilisation de l'énergie par le protocole.

En l'absence de commentaires (*feedback*) explicites du réseau, les protocoles de transport doivent pouvoir compter sur d'autres méthodes pour distinguer les différentes conditions d'erreur. Les erreurs de congestion communes dans les réseaux filaires, se produire quand un ou plusieurs routeurs intermédiaires débordés à la suite d'être submergés par le trafic. Ces erreurs sont généralement accompagnées par une augmentation notable dans les délais. Tous les protocoles de transport fiables répondent à cet événement par un ralentissement de leur taux de transmission, ou en d'autres termes la diminution de leur fenêtre d'envoi. Ne pas le faire peut influer sur le partage équitable de la bande passante entre les connexions concurrentes.

D'autre part, les erreurs de lien peuvent varier dans la nature. Les erreurs de lien sont caractérisées par leur durée et la fréquence. Généralement, des erreurs de lien plus fréquentes dégradent le débit. Même les protocoles de transport avec des meilleures stratégies de contrôle d'erreur peuvent dépenser de la transmission d'*overhead* pour enquêter sur la condition de l'erreur. Déterminer la cause exacte de l'erreur de lien est tout à fait fascinant. Cependant, il est généralement suffisant de conclure le burstiness et la fréquence d'une telle erreur.

I.2.2. Etat de l'art

Le protocole de contrôle de transmission (TCP) est devenu le standard pour l'Internet d'aujourd'hui. Plus de vingt années de recherche ont élaboré un protocole qui est parfaitement optimisé pour les réseaux d'aujourd'hui. Naturellement, le protocole TCP est l'un des premiers protocoles de transport choisi pour les nouveaux réseaux sans fil, satellitaires et hétérogènes. Rapidement, il est devenu évident que le protocole a besoin de quelques améliorations à effectuer dans ces nouveaux environnements. Beaucoup de chercheurs ont eu la démarche de proposer des extensions pour le protocole TCP pour le rendre plus efficace dans ces réseaux. D'autres efforts de recherche vérifier les

résultats obtenus. Cependant, la charge de rester compatible avec les systèmes est un obstacle majeur limitant l'amélioration de TCP.

D'autres chercheurs ont eu l'approche de développement de nouveaux protocoles de transport qui sont plus adaptées aux caractéristiques de leur environnement. Contrairement à la première approche, cette approche ne concerne pas le douloureux processus de la modernisation des extensions proposées de TCP, mais plutôt d'intégrer un grand nombre d'entre eux dans la conception de nouveau protocole. Cette approche ajoute généralement plus d'intégrité et de moins de complications au protocole. Cependant, elle risque de ne pas respecter les normes.

L'un de ces protocoles, proposés par Katz et Henderson [20], est *Satellite Transport Protocol* (STP). Ce protocole est conçu exclusivement pour les réseaux satellitaires. Les auteurs tentent de conserver l'interface avec le protocole TCP. Il s'adresse aux problèmes d'asymétrie, la variabilité du RTT et la dégradation des performances en présence de plusieurs erreurs par RTT. Ces éléments parmi d'autres font de ce protocole mieux adapté que le protocole TCP pour l'utilisation dans des environnements satellitaires. Cependant, le protocole STP manque d'un élément fondamental qui est indispensable pour tout protocole de transport visé pour les réseaux hétérogènes. Cette fonctionnalité manquante est un mécanisme de contrôle d'erreur. Malheureusement, STP hérite de l'hypothèse que toute erreur résultant de la congestion du réseau. Cette impossibilité de classer et de traiter correctement les différents types d'erreurs influe sur le débit et l'énergie dépensée.

I.3. Contributions

Dans ce mémoire nous allons étudier les performances d'un protocole de transport pour les nanosatellites, ce protocole découle d'une optimisation du protocole STP. Cette optimisation est le résultat de l'intégration d'une stratégie de contrôle d'erreur au protocole STP. La nouvelle stratégie est basée sur un mécanisme *probing* de bout-en-bout qui utilise la persistance de l'erreur comme une indication du type de l'erreur dans le réseau. Le mécanisme fonctionne par la suspension de la transmission de données quand une erreur est détectée et d'investir le temps et l'effort de la transmission dans la perception de délai actuel dans le réseau. Le mécanisme associe les conditions d'erreur accompagnées par un délai notable à la congestion du réseau. Sinon, il associe ces erreurs aux différents événements d'erreur de lien. La nouvelle stratégie s'appuie sur les caractéristiques uniques du STP pour améliorer la qualité de contrôle d'erreur. Elle réutilise le cycle POLL/STAT de STP comme un mécanisme de probing. Elle utilise aussi le *selective negative acknowledgement* de STP comme une indication explicite d'erreur. Le nouveau mécanisme probing préserve la sémantique de bout-en-bout de STP, et est implémenté dans l'émetteur uniquement. Le protocole STP avec le nouveau mécanisme probing est appelé : *eXtended Satellite Transport Protocol* (XSTP).

Le protocole XSTP est implémenté sous NS2 pour cette étude. En utilisant la simulation, cette étude montre que l'optimisation du protocole STP améliore le débit effectif et augmente l'efficacité énergétique du protocole. Les réseaux satellitaires étudiés dans ce mémoire sont les réseaux d'accès satellitaires LEO. Les performances de XSTP sont comparées avec des variantes du protocole TCP, ce qu'on appelle les *clones TCP*. Quatre paramètres de performances sont mesurés : le débit effectif (le taux moyen des données reçues au niveau récepteur), l'utilisation du lien (le ratio entre le débit effectif et l'overhead), la transmission d'overhead (le pourcentage d'octets supplémentaires transmis durant le transport de données) et la bande passante nécessaire pour le canal inverse.

I.4. Organisation du mémoire

Le reste du mémoire est organisé en six chapitres. Le Chapitre 2 présente une vue sur le domaine des nanosatellites, où nous avons présenté les différents concepts utilisés dans ce domaine, la structure générale d'un nanosatellite et la classification des satellites. Le Chapitre 3 étudie la littérature des protocoles de transport disponibles et quelques extensions proposées pour les améliorer dans les réseaux d'accès satellitaires. Le Chapitre 4 étudie les protocoles de transport pour les nanosatellites et considère l'optimisation du protocole STP par l'intégration du mécanisme probing. Le Chapitre 5 explique le cadre de la simulation, y compris la configuration de la simulation, les paramètres des performances et les

scénarios de tests. Le Chapitre 6 présente les résultats de la simulation et leurs interprétations. En fin, nous concluons en donnant un résumé et mentionnant quelques perspectives.

CHAPITRE II
DOMAINE DES NANOSATELLITES

Introduction

Un satellite artificiel est un objet fabriqué par l'homme, envoyé dans l'espace à l'aide d'un lanceur et gravitant autour d'une planète ou d'un satellite naturel comme la Lune. La vitesse imprimée par la fusée au satellite lui permet de se maintenir pratiquement indéfiniment dans l'espace en décrivant une orbite autour du corps céleste. Celle-ci, définie en fonction de la mission du satellite, peut prendre différentes formes (héliosynchrone, géostationnaire, elliptique, circulaire) et se situer à des altitudes plus ou moins élevées classifiées en orbite basse, moyenne ou haute.

La réduction des coûts dans le domaine spatial est devenue ces vingt dernières années un enjeu majeur. Cet objectif est réalisé principalement grâce à la diminution des temps de développement et à la miniaturisation des vaisseaux spatiaux. Ainsi, l'énergie à fournir est bien moindre pour le lancement qui représente une part importante du budget de fonctionnement [18].

Dans le cas des satellites, on peut constater une accélération de cette tendance sur les dix dernières années. Il est même apparu des réalisations de satellites dont la masse est comprise entre 1 et 10 kilogrammes : les nanosatellites [18].

II.1. Concepts utilisés dans le domaine des nanosatellites

II.1.1. Définition de nanosatellite

L'apparition du terme Nanosatellite est assez récente sur la scène de la recherche spatiale et il existe encore quelques ambiguïtés dans son usage. Lorsque l'on parle d'un nanosatellite, on ne fait référence ni à l'échelle nanométrique, ni à la présence de nanosystèmes, mais à un mémoire de taille existant avec les satellites traditionnels [18].

En fait, au fur et à mesure de la miniaturisation des satellites est apparu le besoin de définir une catégorie pour des satellites de masse inférieure à une demi-tonne (les minisatellites), une autre pour les satellites de moins de 100 kg (les microsatellites) et finalement une autre concernant les satellites dont la masse est comprise entre 1 et 10kg : les nanosatellites [18].

Il faut encore noter que tel que nous l'entendons ici, la dénomination nanosatellite ne donne aucune information sur la fonction. En effet, il faut prendre le terme satellite dans son sens technique le plus large, un corps (artificiel) en orbite autour de la terre. Ainsi, il peut s'agir d'un satellite traditionnel aussi bien que d'un mobile se déplaçant localement sur une orbite (autour d'un autre satellite, par exemple) [18].

II.1.2. Concept de Vol en formation

Un objet lancé à la surface de la Terre décrit une trajectoire parabolique qui le ramène au sol sous l'influence de la gravité terrestre (*cas A sur la Figure II.1*). Plus la vitesse initiale de l'objet est importante plus le point de chute est éloigné (*cas B sur la Figure*). Lorsqu'une certaine vitesse est atteinte, l'objet chute mais sans jamais atteindre le sol du fait de la courbure de la Terre (*cas C sur la Figure*). Pour que l'objet conserve indéfiniment sa vitesse, il faut toutefois que celui-ci se déplace dans le vide au-dessus de l'atmosphère, là où aucune force de traînée (frottement) ne s'exerce.

Pour qu'un objet soit satellisé autour de la Terre il faut que sa vitesse horizontale par mémoire au centre de la Terre (la vitesse d'injection) soit de 7700 mètres par seconde pour une orbite circulaire à 200km au-dessus de la Terre (au-dessous de cette altitude la traînée est trop importante). Si on communique une vitesse supérieure à un satellite circulant à la même altitude, l'orbite devient elliptique (*cas D sur la Figure*) : le point de cette orbite le plus éloigné de la terre se nomme l'*Apogée* et le point le plus proche le *Périgée* (voir Figure II.3). Cette orbite se trouve inclinée par mémoire à un plan passant par l'équateur terrestre [18]. Si la vitesse dépasse 11km par seconde (*cas E sur la Figure*), le satellite échappe à l'attraction terrestre : c'est la vitesse de libération de la Terre.

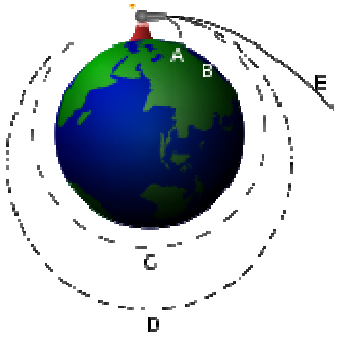

Figure II.1. Loi de Newton : au-delà d'une certaine vitesse le boulet ne retombe plus au sol

Pour décrire précisément la position d'un satellite sur une orbite terrestre, il faut un certain nombre de paramètres, appelés éléments Képlériens. Les premiers d'entre eux décrivent l'orbite elle-même par mémoire à la terre et au soleil, les seconds les coordonnées du satellite sur cette orbite. La trajectoire d'un satellite artificiel ou naturel est régie par les 3 lois formulées par *Kepler*. Ces trois lois sont utilisées pour décrire rapidement l'orbite d'un satellite car ils donnent une information immédiate sur sa fonctionnalité [18].

- *loi I :* l'orbite du satellite a la forme d'une ellipse dont un des deux foyers se trouve au centre du corps céleste (par exemple la Terre) autour duquel il gravite; une orbite circulaire est un cas particulier de l'ellipse dont les deux foyers sont confondus au centre de la Terre;

La forme de l'ellipse peut être définie par (voir Figure II.2):

- ➔ la distance r_p du point de l'orbite le plus proche de la Terre (le périgée) au centre de la Terre,
- ➔ la distance r_a du point de l'orbite le plus éloigné de la Terre (l'apogée) au centre de la Terre,

On utilise généralement à la place :

- ➔ le demi-axe a défini par la formule $2a = r_p + r_a$,
- ➔ l'excentricité e qui définit l'allongement de l'ellipse et peut être calculée par la formule $e = 1 - r_p/a$. Elle prend une valeur comprise entre 0 et 1 : 0 correspond à une orbite circulaire et plus la valeur est proche de 1 plus l'orbite est allongée,

- *loi II :* le satellite se déplace d'autant plus vite qu'il est proche du corps céleste; plus précisément la droite qui joint le centre du corps céleste au satellite balaie toujours une aire égale dans un intervalle de temps donné. La couverture instantanée est donc inférieure ou égale à l'aire d'accès instantanée, définie comme la surface du globe depuis laquelle le satellite est visible à un instant donné [18];
- *loi III :* le carré de la période de rotation du satellite autour du corps céleste varie comme le cube de la longueur du grand axe de l'ellipse. Si l'orbite est circulaire, le grand axe est alors le rayon du cercle.

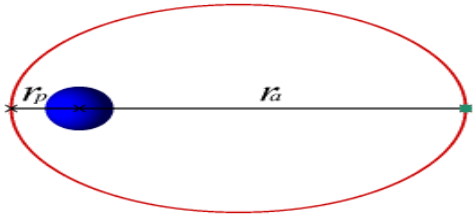

Figure II.2. La Terre se trouve à l'emplacement d'un foyer de l'orbite elliptique du satellite dont la vitesse croit d'autant plus que la Terre est proche

Six paramètres sont utilisés pour fournir la position et la trajectoire d'un satellite dans l'espace [18] :

- l'orbite d'un satellite est un plan. Si on ne tient pas compte des perturbations naturelles auxquelles elle est soumise et en l'absence de manœuvres du satellite, le plan d'orbite *est fixe dans l'espace*. Ce plan peut être défini par deux paramètres : l'inclinaison *i* et la longitude (ou ascension droite) du nœud ascendant Ω;
- trois paramètres : l'excentricité *e* et le demi-grand axe *a* de l'ellipse ainsi que l'argument du périgée ω permettent de décrire la trajectoire en forme d'ellipse dans le plan d'orbite;
- un dernier paramètre permet de situer le satellite sur son orbite : on peut par exemple prendre le temps *t* écoulé depuis le passage au périgée.

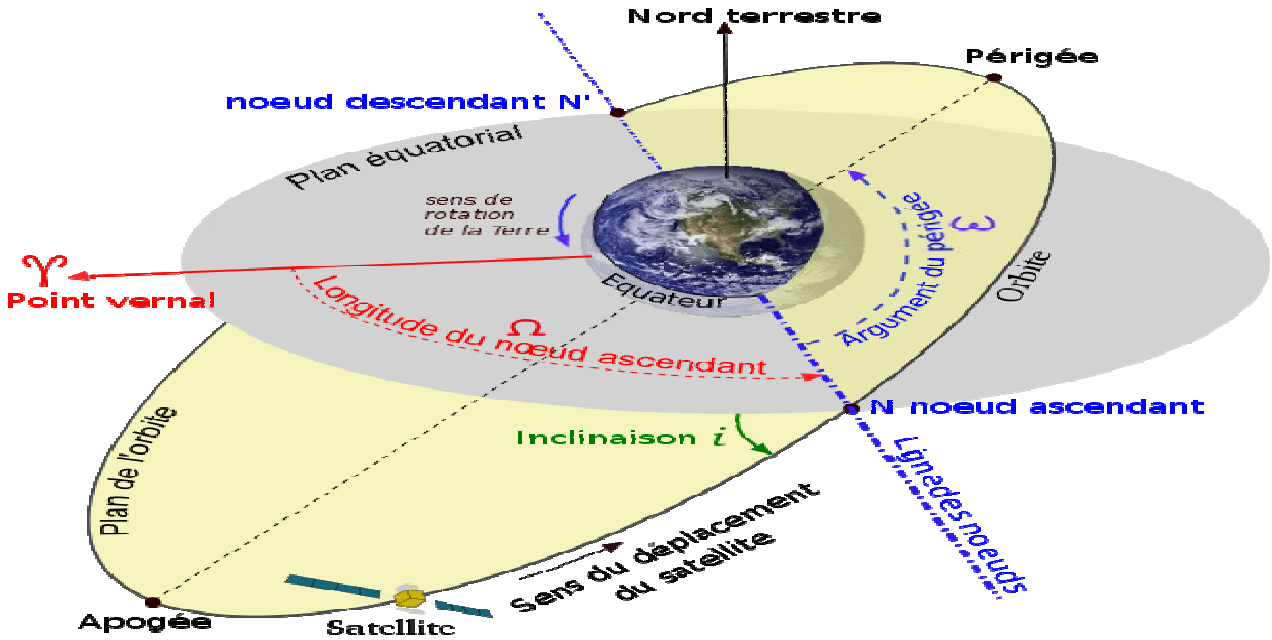

Figure II.3. Paramètres orbitaux d'un satellite artificiel : ascension droite du nœud ascendant *Ω*, inclinaison *i*, argument du périgée *ω*

Les paramètres de l'orbite sont définis dans un référentiel constitué de plusieurs plans et de droites [18] (voir la Figure ci-dessus) :

- la trajectoire de la Terre autour du Soleil s'inscrit dans un plan, dit *plan de l'écliptique*, passant par le centre du Soleil;
- le *plan de l'équateur terrestre* est le plan passant à la latitude de l'équateur;
- à l'équinoxe de printemps le 21 mars le plan de l'équateur terrestre coupe le plan de l'écliptique selon une ligne dite *ligne des équinoxes* passant par le Soleil. Cette droite qui désigne à l'infini le point vernal *v* est fixe dans le système solaire;
- l'orbite d'un satellite coupe le plan de l'équateur en deux points appelés *nœud ascendant* lorsque le satellite passe de l'hémisphère sud à l'hémisphère nord et *nœud descendant*. La ligne reliant les deux points est appelée *ligne des nœuds*.

La position du satellite est donnée en connaissant le grand axe et l'excentricité de l'ellipse, par quatre angles géocentriques :

- *l'inclinaison i du plan de l'orbite* du satellite (entre 0 et 180 degrés) est l'angle que fait le plan de l'orbite avec le plan de l'équateur. Lorsque $i = 90°$ l'orbite du satellite survole les pôles (orbite polaire); si $i = 0$ le plan de l'orbite se situe dans le plan de l'équateur. L'orbite est dite directe lorsque *i* est inférieur à 90° et rétrograde sinon;
- l'*ascension droite Ω*, l'angle entre l'équinoxe vernal et le nœud ascendant (intersection de la trajectoire montante de l'orbite avec le plan équatorial);
- l'*argument du périgée ω*, l'angle entre le nœud ascendant et le périgée;
- *l'anomalie vraie v*, l'angle entre le périgée et le satellite.

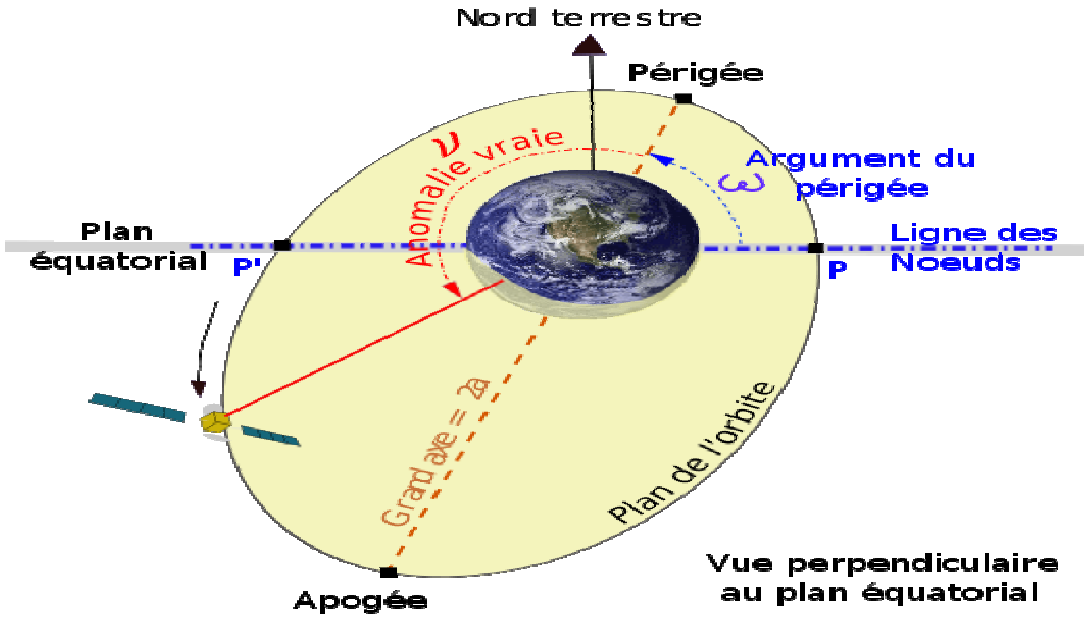

Figure II.4. Paramètres orbitaux : demi grand axe a, argument du périgée ω, anomalie vraie v

II.1.3. Types d'orbites pour les satellites

Les orbites des satellites terrestres peuvent avoir de nombreuses formes et orientations : certaines sont circulaires ou au contraire en forme d'ellipse très allongée. Elles peuvent se situer à basse altitude juste au-dessus de l'atmosphère terrestre (250 km) ou dépasser 30000 km. L'orbite d'un satellite artificiel est choisie pour répondre au mieux aux besoins de la mission. La plupart des satellites utilisent l'un des orbites suivantes [18] :

- Une ***orbite polaire*** est une orbite circulaire basse (par convention entre 300 et 1000 km d'altitude) dont l'inclinaison, proche de 90°, la fait passer au-dessus ou près des pôles. Un satellite situé sur une orbite polaire passe régulièrement au-dessus de tous les points de la surface grâce à la rotation de la Terre. Le désavantage de cette orbite est sa longue période entre deux passages au dessus d'un point particulier comme par exemple la station de communication au sol. Beaucoup de satellites d'observation qui doivent couvrir la terre entière sont dans des orbites polaires ou quasi-polaires;
- Une ***orbite géosynchrone*** décrite dans le même sens de rotation que la Terre, à environ 35800 km d'altitude. Autrement dit, la trajectoire de sa couverture instantanée est identique à celle de la période précédente. Un rôle typique pour un satellite en orbite géosynchrone est la surveillance, et la transmission de communications pour des latitudes géographiques élevées;
- Une ***orbite héliosynchrone***, ce type d'orbite conserve un angle constant avec la direction Terre-Soleil c'est-à-dire que le plan d'orbite tourne de 360° par an. Les orbites héliosynchrones permettent de passer toujours à la même heure solaire locale au-dessus d'un lieu donné : l'éclairage identique des prises de photo du lieu permet de faire ressortir les changements. Un satellite héliosynchrone croise l'équateur et chaque latitude à la même heure chaque jour. Ce type d'orbite est donc avantageux pour un satellite d'observation terrestre, car il fournit des conditions d'illumination constantes pour un même lieu;
- Les ***basses orbites terrestres*** (***LEO***) sont des orbites elliptiques et qui situe juste au-dessus de l'atmosphère terrestre à une altitude où la traînée ne freine pas trop la vitesse du satellite (par convention l'orbite basse se situe à une altitude inférieure à 2000 km). Une fusée a besoin de moins de puissance pour placer un satellite sur ce type d'orbite. Elle est utilisée par les satellites scientifiques qui explorent l'espace lointain. Le télescope Hubble, par exemple se situe sur une orbite de 610 km. On trouve également sur ce type d'orbite les satellites de radioamateur et les constellations de téléphonie mobile ou de télédétection terrestre;

- Les ***orbites terrestres moyennes*** (***MEO***) ou orbites circulaires intermédiaires (ICO) sont des orbites circulaires situées à environ 10000 km d'altitude dont la période de révolution est d'environ 6h. L'orbite située en dehors de l'atmosphère terrestre est très stable. Les signaux envoyés par le satellite peuvent être reçus sur une grande partie de la surface du globe terrestre. C'est l'altitude retenue pour les satellites de navigation comme le système GPS;
- Les ***orbites fortement elliptiques*** (***HEO***) possèdent typiquement un périgée à environ 500 km d'altitude et un apogée aux environs de 50000 km. Elles sont géosynchrones d'une période de 8 à 24 heures et sont inclinées à 63,4 degrés. En raison de l'excentricité élevée de l'orbite, un satellite passera environ deux tiers de la période orbitale proche de l'apogée, et pendant ce temps il semble être presque stationnaire pour un observateur sur la terre (ceci est désigné sous le nom de saturation d'apogée);
- L'***orbite géostationnaire*** (***GEO***) (ou de Clarke) est une orbite circulaire située dans le plan de l'équateur à une altitude de 35786 km du sol (le rayon de l'orbite est donc de 42164 km). À cette altitude la période de révolution du satellite correspond exactement à la période de rotation de la Terre, soit 23 heures, 56 minutes et 4 secondes. Vu de la Terre, un satellite géostationnaire semble immobile dans le ciel : c'est l'orbite parfaite pour les satellites de télécommunications et pour certains satellites d'observation (météo) qui doivent couvrir une zone fixe. Trois satellites géostationnaires suffisent pour l'ensemble de la surface du globe terrestre. La mise à poste d'un satellite géostationnaire nécessite, du fait de l'altitude, un lanceur puissant. Pour les télécommunications la distance franchie par un signal transitant par ce type de satellite crée un délai perceptible par un usager.

II.1.4. Lancement

Dans le cas des nanosatellites, un des modes de mise sur orbite courant est le parasitage du lancement de satellites plus gros : le lancement en "piggy-back" [18]. Il s'agit de profiter de l'espace libre restant dans le lanceur pour abaisser fortement le coût de l'opération. Le plus dur étant de trouver un hôte compatible qui veuille bien accepter le parasitage. Naturellement, plus la masse et le volume du nanosatellite sont faibles plus les opportunités sont nombreuses. Il semble que ce type de pratique soit de mieux en mieux accepté, ce qui représente souvent le seul mode d'accès à l'espace pour des réalisations académiques. On évalue généralement le coût d'un tel lancement à 10 k$/kg sans compter les frais éventuels d'adaptation.

II.2. Structure générale d'un système nanosatellite

La Figure ci-dessous montre l'ensemble des sous-systèmes qui constituent un satellite en général. Il s'agit d'un diagramme simplifié, mais il permet tout de même d'identifier les principales fonctions et les flux d'information ou d'énergie.

Dans un nanosatellite la masse de celui ci, structure comprise, ne doit pas dépasser 10 kg et l'orbite considérée est de la classe des très basses orbites (VLEO), soit entre 160 et 500 km d'altitude.

II.2.1. Sous-système GNC

Le bloc GNC (Guidage et Contrôle de la Navigation) regroupe deux fonctions : la détermination et le contrôle de la position et de l'attitude. Il s'agit souvent du sous-système le plus cher d'un satellite [18].

Le satellite doit faire face à la terre à tout moment. Le système de contrôle d'attitude permet au satellite de rester pointer correctement. Ceux-ci sont souvent très petits moteurs par mémoire au système de propulsion. La plupart des nanosatellites ont le magnetorquers COTS (*commercial-off-the-shelf*) pour le traitement de contrôle de l'orbite. Pour la détermination de la position, la quasi-totalité des nanosatellites utilisent des récepteurs GPS [11].

On définit l'attitude par trois angles et par la variation de ces angles par mémoire au temps:

- le roulis (roll) : rotation du satellite autour de l'axe passant par son centre de gravité et défini par son vecteur vitesse;
- le lacet (yaw) : rotation du satellite autour de l'axe passant par son centre de gravité et le centre de la terre;
- et le tangage (pitch) : rotation autour de l'axe du satellite passant par son centre de gravité et perpendiculaire aux deux précédents.

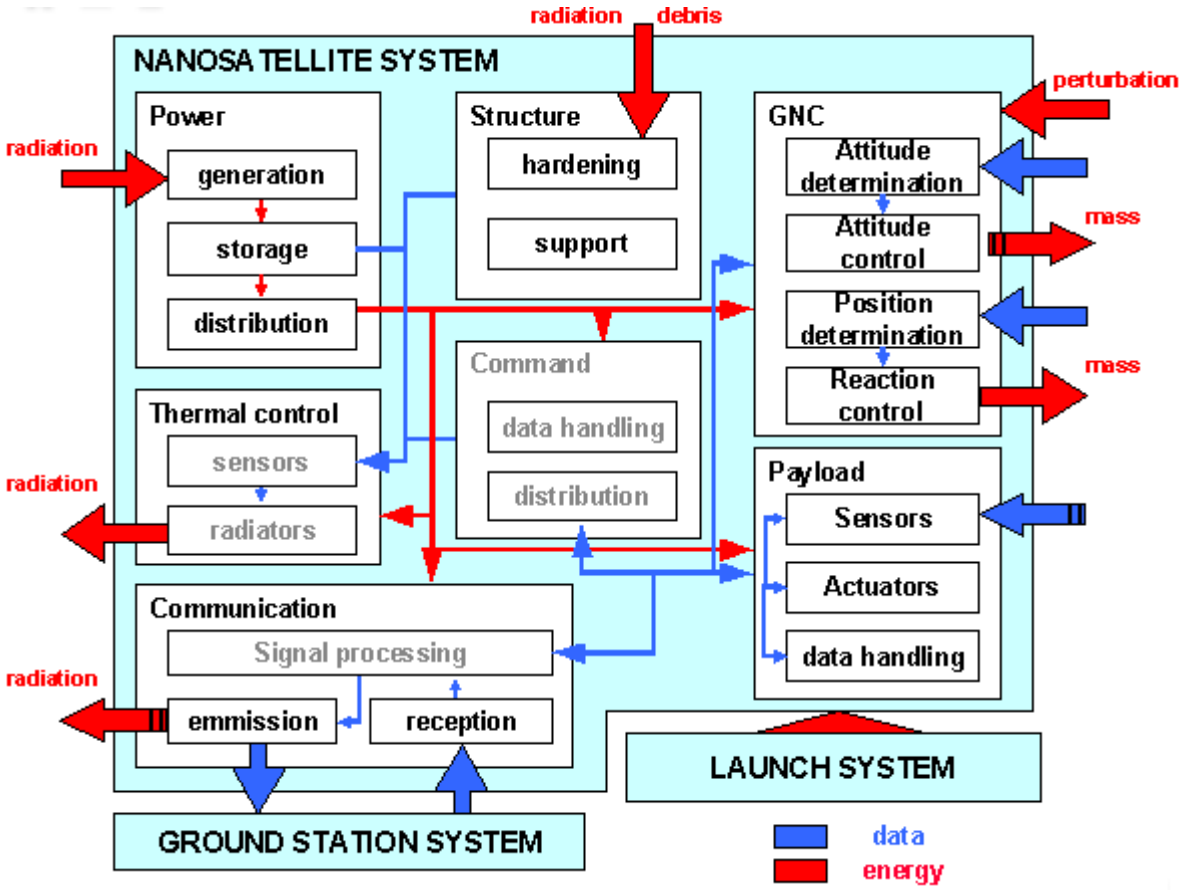

Figure II.5. Schéma bloc fonctionnel d'un satellite

II.2.2. Sous-système d'alimentation

Un autre sous-système important est celui de la génération et du stockage de l'énergie. Les panneaux solaires sont utilisés en combinaison avec des batteries pour fournir une source constante d'énergie électrique sur le satellite. Les batteries sont utilisées lorsque le satellite n'est pas dans la lumière directe du soleil et elles permettent au satellite de continuer à fonctionner. Les satellites LEO ont plus besoin de batteries que les satellites GEO, car les satellites GEO sont exposés au soleil plus longtemps. Presque tous les nanosatellites utilisent les cellules solaires GaAs et les batteries Li-Ion ou NiCad pour le stockage de l'énergie [11].

Il faut en effet minimiser la masse embarquée sans mettre en péril l'autonomie du satellite; rappelons qu'un satellite en orbite se retrouve forcément éclipsé par la Terre lors d'une partie de sa révolution [18]. Cette éclipse dure au minimum 35 minutes dans le cas d'une orbite terrestre circulaire. Naturellement, pour une orbite plus elliptique, la durée de l'éclipse peut être plus longue. En ce qui concerne les orbites en LEO, elle représente environ 40 % de la période orbitale. Cela constitue encore une limitation importante à l'expansion des nanosatellites. Les solutions envisageables sont par exemple la mise en veille lors de l'éclipse ou le fonctionnement alterné des sous-systèmes.

II.2.3. Sous-système de Communication

Un sous-système de communication satellitaire assure la transmission, la réception et l'acheminement de données. Il utilise des émetteurs, récepteurs ou transporteurs (transmetteur et récepteur dans un seul composant). Le sous-système de communication s'occupe de toutes les fonctions de transmission et de réception des communications. S'il est un satellite de communication, ce sera la partie lourde de la construction du satellite.

Le dimensionnement du sous-système de communication dépend très étroitement du fret embarqué et des besoins de la mission. Il est utile d'attirer l'attention sur de nouvelles options qui pourraient modifier fondamentalement la manière de communiquer des satellites avec leur station au sol. Il s'agit de l'utilisation des constellations de satellites destinées à la transmission de données à haut débit, comme vecteur des communications du nanosatellite [18].

On trouve plusieurs candidats potentiels dont :

- le réseau *Iridium* de télécommunication qui se situe en LEO (à 780 km d'altitude) qui propose le service de relais de communication mobile. Le principal avantage réside dans la faible puissance nécessaire pour communiquer avec ces satellites, mais le revers de la médaille est que la différence d'altitude avec un nanosatellite en orbite à 400 km d'altitude ne sera peut être pas suffisante pour assurer une communication ininterrompue;
- le réseau *GlobalStar* possédera 48 satellites à une altitude de 1414 km. Si le problème de l'altitude semble résolu, le désavantage de cette solution réside dans la faible bande passante que le système procure;
- et bientôt le réseau *Teledesic* se composera de 288 satellites opérationnels, divisés en 12 plans (vers 1400 km) ayant chacun 24 satellites. Il s'agit d'un véritable réseau de satellites, les données transitant par plusieurs satellites avant de redescendre sur terre. La bande passante pourrait aller jusqu'à 64 Mbit/s sur la liaison descendante et jusqu'à 2 Mbit/s sur la liaison montante.

Les avantages de cette dernière solution sont indéniables sur les communications entre le nanosatellite et la station au sol :

- Possibilité de communication à haut débit;
- Faible puissance nécessaire (pas d'atmosphère à traverser);
- Possibilité d'accès quasi permanente.

Si l'on considère la façon dont l'adaptation de la navigation GPS aux satellites a été réalisée, il y a fort à parier que des solutions adaptant ces nouveaux protocoles verront rapidement le jour et que la communication en temps réel avec des satellites est pour très bientôt.

II.2.4. Sous-systèmes de Propulsion

Le sous-système de propulsion n'est qu'une partie des éléments qui mettre le satellite en orbite. Autres produits chimiques ou des moteurs électriques sont utilisés pour déplacer le satellite dans la bonne orbite quand l'atmosphérique, champs magnétiques, les vents solaires détournent le satellite sur sa bonne trajectoire.

Afin d'effectuer les manœuvres nécessaires pour mettre le satellite en orbite, nanosatellites utilisent souvent le gaz froid ou des moteurs plasmiques. Il existe également des nanosatellites sans aucun système de propulsion, donc en se fondant uniquement sur la torquers. C'est le cas de nanosatellites qui sont utilisés comme des bancs d'essais pour les différents sous-systèmes [11].

II.2.5. Sous-systèmes structurel

Le satellite doit survivre à la violence des forces de la fusée monté dans l'espace. La superstructure du satellite, il soutient non seulement dans l'espace, mais réduit les chocs et les vibrations des composants internes risquent de subir au cours du lancement.

La plupart des nanosatellites ont la structure mécanique de *Al 6061* ou *Al 7075* plutôt que les matériaux composites. Parce que la plupart des nanosatellites ont une courte durée de vie, ils n'ont pas besoin aux matériaux plus résistant. Aussi, l'aluminium est moins cher que les matériaux composites [11].

II.2.6. Sous-systèmes de contrôle thermique

Le sens du système thermique est de réguler la température des composants de satellite. Trop chaud ou trop froid, ou une trop grande balançoire température sera prématurément finir la durée de vie utile d'un satellite. Ce système dissipe la chaleur de la terre, dans l'espace, afin de ne pas interférer avec le fonctionnement du satellite.

Deux types de matériels utilisés pour le contrôle thermique : des capteurs et des radiateurs. Les détecteurs de température de la résistance Thin film sont sélectionnés en tant que capteurs car ils offrent plus de souplesse et de stabilité par mémoire à d'autres capteurs de température [11].

II.2.7. Sous-systèmes de télémétrie, suivi et commande

Le satellite doit informer le centre d'opérations de satellite quel est son état actuel, et où il se trouve en orbite. Souvent, un simple phare est utilisé pour permettre à la station au sol de suivre les satellites en orbite. Des informations complémentaires sont transmises au sol, tels que le fonctionnement de l'artisanat de la température, l'état de ses programmes et du système d'exploitation, ainsi qu'une foule d'autres fonctions internes.

Plusieurs solutions sont disponibles pour les communications, la plupart des systèmes utilisent UHF, d'autres S-bande ou X-band. *Satellite Toll Kit* (STK) est utilisé pour la visualisation et le suivi des stations sol. D'autres options sont FreeFlyer de AI Solutions Inc ou DSST de Draper Labs [11].

II.3. Classification des satellites

Le Tableau II.1 présente la classification des satellites, avec spécification de masse, coût et le temps de développement [11].

Catégorie	Masse (kg)	Coût	Temps (ans)
Large	≥ 1000	≥ 500 Million \$	≥ 15
Small	500 - 1000	100 Million \$	5
Mini	100 - 500	20 Million \$	2 - 3
Micro	10 - 100	10 Million \$	1.5
Nano	1 - 10	350 k\$ - 1 Million \$	≈ 1
Pico	0.1 - 1	≥ 100 k\$	≥ 1
Femto	≤ 0.1	_	_

Tableau II.1. Classification des satellites

II.4. Avantages et Inconvénients des nanosatellites

Les satellites minimisés présentent plusieurs avantages par mémoire aux satellites traditionnels, tels que [11] :

- coût de fabrication réduit;
- facilité de la production de masse;
- coût de lancement réduit;
- capacité à être lancé en groupes ou «piggyback» avec des satellites plus gros;
- perte financière minimale en cas d'échec.

Les inconvénients des satellites minimisés par mémoire aux satellites plus grands, en particulier lorsque mis en orbite terrestre basse sont les suivants [11] :

- en générale réduction de la vie active;
- réduction de capacité de transport de matériels;
- basse capacité de puissance de sortie de l'émetteur;
- corruption orbitale plus rapide.

II.5. Applications des nanosatellites

Les missions adaptées d'un point de vue technologique aux nanosatellites sont les suivantes [11] :

II.5.1. Communication

Les microsatellites utilisent les première techniques de Internet pour fournir le monde entier, les données numériques non temps réel, en particulier pour des régions éloignées où existent des télécommunications avec des infrastructures inadéquate ou inexistante.

Plusieurs constellations des satellites petits en orbite basse (LEO) ont été proposées pour prévoir des communications de monde entier en utilisant uniquement des terminaux portables pour des services temps réel (voix et données) (par exemple : Iridium, Globalstar) et le transfert de données non temps réel (par exemple : Orbcom, HealthNet, Temisat, VITASat, GEMStar et E-SAT).

II.5.2. Espace scientifique

Les nanosatellites peut offrir un demi-tour rapide et peu coûteuse d'explorer des moyens bien ciblés, les petits objectifs scientifiques (par exemple : le contrôle de l'espace de radiation, la mise à jour de champ de référence géo-magnétique, etc) ou de fournir un début de preuve de concept avant le développement de grands instruments. Celle-ci offre également des possibilités pour les jeunes scientifiques et ingénieurs d'acquérir «la vie réelle» du satellite et de l'ingénierie de la charge utile et pour être en mesure de lancer un programme de recherche, de proposer et de construire un instrument scientifique.

II.5.3. Technologie de vérification

Les nanosatellites donnent aussi un coût moyen faible de démonstration, de vérification et d'évaluer de nouvelles technologies ou de services rapidement dans un environnement orbital réaliste et avec des risques acceptables avant l'engagement dans une mission chère.

II.5.4. Observation de la Terre

Les nanosatellites ont provoqué une révolution dans l'observation de la Terre depuis l'espace. Les observations classiques de la Terre et les missions de télédétection par satellite sont extrêmement coûteux (500 millions$).

La disponibilité commerciale de détecteurs optiques (utilisée dans la consommation vidéo et appareils photo numériques), associée à une faible consommation d'énergie pourtant le calcul puissant des microprocesseurs, présente une nouvelle occasion pour la télédétection et l'utilisation de petits satellites.

II.5.5. Applications militaires

Une version militaire de la plate-forme de microsatellite SSTL avec déploiement des panneaux solaires ont été développé pour supporter les différents déploiements militaires.

Les principaux différences entre les versions commerciales et le militaires sont dans la spécification et l'achat de composants et en particulier à la quantité de documentation des traces de matériel et de procédures. L'équilibre optimum entre les contraintes d'un programme militaire et économique à été demandé, se qui résulte en une augmentation des facteurs de coût et de temps, environ 1.5 par mémoire au processus de microsatellite commercial.

II.6. Campagnes de lancement des nanosatellites

Entre 2004 et 2007 on a recensé 55 nanosatellites qui sont lancés [12].

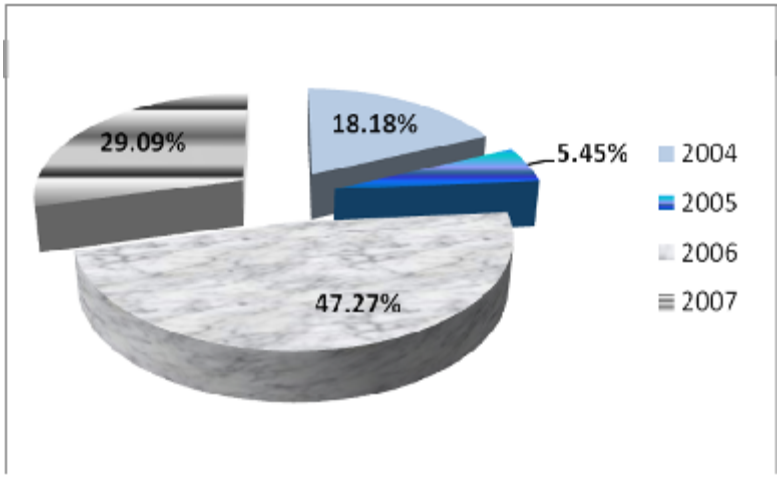

Figure II.6. Nanosats lancé dans la période 2004 – 2007

Comme le montre la Figure II.6, nous pouvons conclure que 18,18% des nanosatellites ont été lancés en 2004, 5,45% en 2005, 47,27% en 2006 et 29,09% en 2007. 2006 représente près de 50% du nombre total de lancements.

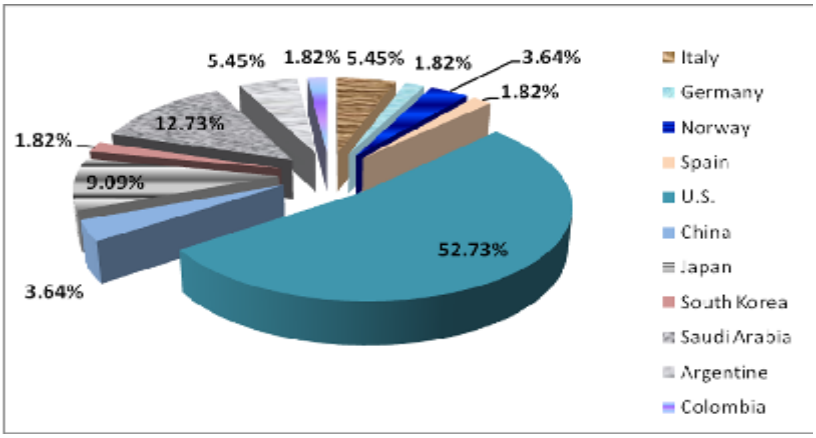

Figure II.7. Lancement des nanosatellites par pays

En ce qui concerne le pays propriétaire de déploiement (Figure II.7), le U.S.A avec 29 déploiements, avec un pourcentage de 52,73% du total des lancements. Arabie saoudite prend la deuxième place avec 12,73%, elle a lancé 7 nanosatellites. Le Japon a lancé 5 nanosatellites (9,09%), gagne la troisième place, la quatrième place c'est pour l'Italie et l'Argentine avec 5.45% du total des lancements, la Norvège et la Chine a lancé tous les 2 nanosats, ce qui représente 3,64% du total, l'Allemagne, l'Espagne, la Corée du Sud et la Colombie ont chacun mené, une seule campagne de lancement (1,82%).

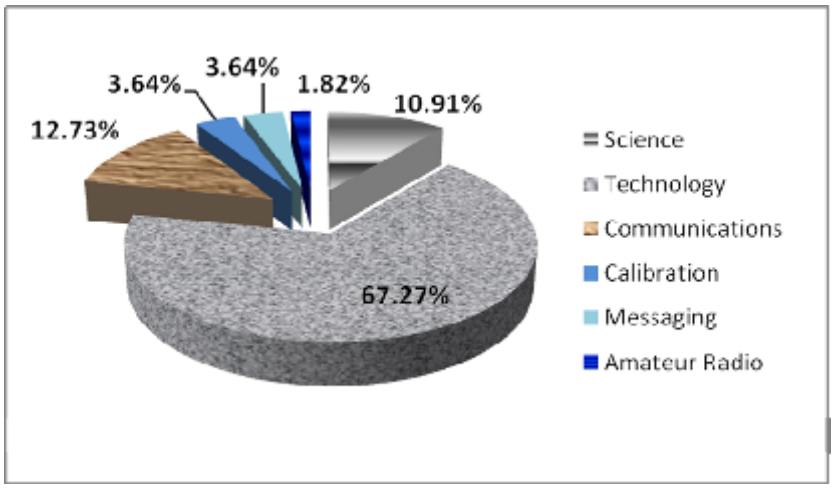

Figure II.8. Lancement des nanosatellites par mission

Comme le montre la Figure II.8, en termes de catégorie de mission, 67,27% des campagnes d'appartenir à la catégorie technologie. Les missions de communications ont participé 12,73% du nombre total de nanosats lancé et 10,91% des lancements sont appartient à la catégorie scientifique. Certains nanosatellites ont été conçus pour une mission de messagerie (3,64%), mission de calibration (3,64%) et la mission Radio Amateur (1,82%).

La majorité des nanosatellites (37 déploiements) ont été conçus pour des missions technologiques.

En ce qui concerne l'état de lancements des nanosats dans la période 2004-2007 (Figure II.9), 70,91% des lancements ont été réalisé avec succès et 29,09% des missions ont échoué. Le seul échoue qui a été enregistrée le 26 Juillet 2006, lorsque tous les 16 nanosatellites ont été perdues en raison de l'explosion du lanceur.

Figure II.9. L'état de tous les lancements de nanosats

Conclusion

La réalisation de nanosatellites est aujourd'hui technologiquement possible, tout particulièrement, dans le cas des orbites très basses, où l'emploi de technologies standards est envisageable (sous réserve de quelques précautions à prendre à l'encontre des radiations résiduelles). De plus, l'utilisation de satellites situés sur des orbites bien supérieures, comme par exemple les systèmes GPS ou Teledesic, offre de nouvelles options de navigation et de communication à ces altitudes.

D'un point de vue économique maintenant, c'est très certainement la faible masse, et par voie de conséquence le faible volume d'un nanosatellite, qui constitue le principal argument en sa faveur. En effet, l'énergie qu'il faut dépenser pour mettre un satellite en orbite étant proportionnelle à cette masse, cela provoque une baisse phénoménale des coûts de lancement : alors qu'il représente entre 20 et 50% du coût d'une mission traditionnelle, le budget de lancement d'un nanosatellite se monte entre 5% et 10% du total.

Après cette description des différents concepts utilisés dans le domaine des nanosatellites, dans le chapitre suivant nous allons décrire certains protocoles de transport utilisés pour les communications satellitaires.

CHAPITRE III
PROTOCOLES DE TRANSPORT POUR LES COMMUNICATIONS SATELLITAIRES

Introduction

Les communications satellitaires face à des défis particuliers qui sont dérivés des caractéristiques des liaisons radio de satellite. Premièrement, le temps aller-retour (RTT) est très grand et cela à cause de long temps de propagation sur les canaux radio de satellite. Deuxièmement, la présence des pertes aléatoires qui ne sont pas causées par la congestion et qui ne peuvent pas être considérées toujours comme négligeable. Les pertes dues à la conception du lien (marge de lien, codage et modulation) et aux conditions opérationnelles (obstructions des liens, mobilité de terminal, conditions météorologiques, etc.) [2].

Les liaisons satellitaires sont asymétriques, généralement en raison de coût élevé de la technologie nécessaire pour soutenir la même technologie pour les deux directions. Cette asymétrie influe sur les performances des protocoles de transport qui utilisent des ACK dans le sens inverse comme mécanisme d'auto-synchronisation comme TCP [19]. Elles sont caractérisées aussi par une haute latence et bande passante et aussi un Bit Error Rate (BER) plus élevé.

Il y a deux approches dans la conception des protocoles de transport pour les réseaux satellites : une qui essaye d'améliorer les protocoles standard comme TCP et une autre qui conçoit des nouveaux protocoles dédiés pour les satellites seulement [19]. Au cours des dernières années, plusieurs solutions ont été proposées pour améliorer les performances de TCP sur les deux liaisons filaires et sans fil, y compris les liaisons par satellite [2].

Les limitations du protocole TCP sur satellite ne sont pas causées par les conditions de satellite elles-mêmes, mais l'interaction de TCP avec ces conditions [4]. Si le protocole TCP ne fonctionne pas bien dans un environnement où la latence élevé, BER élevé, ou une bande passante sur des liens asymétrique, une solution consiste à utiliser un autre protocole qui est mieux adapté à ces conditions. Il existe des protocoles qui offrent des débits et de la réactivité au-delà de ce qui est possible avec le protocole TCP dans le cadre de ces conditions particulières. Ce chapitre passe en revue un certain nombre de ces protocoles.

III.1. Problèmes de communication spatiale

L'environnement spatial en général offre un certain nombre d'obstacles à une communication de données fiable.

III.1.1. Error-Prone Links

Dans les communications sans fil en général, mais surtout dans les communications spatiales, les erreurs de bit causées par le bruit ne sont pas rares. TCP est conçu pour gérer la perte de paquets par l'identification et la retransmission des segments perdus, mais TCP suppose que la source de toutes les pertes de paquets est la congestion du réseau. Cette réponse n'est pas face à la perte due à la corruption plutôt que de la congestion, comme c'est souvent le cas sur l'espace et d'autres liens de communications mobiles [15, 19].

III.1.2. Canaux asymétrique

Les canaux de communication spatiale entre le satellite et la Terre sont souvent asymétriques en termes de capacité de canal et des caractéristiques d'erreur. Cette asymétrie est le résultat de différents compromis (comme l'énergie, la masse et de volume), ainsi que la plupart des données provenant de satellites vers la Terre. Le lien de retour est généralement utilisé pour faire des commandes au satellite, et non pas pour le transfert de données. La réduction de nombre d'ACK est toujours un objectif souhaitable, car il réduit l'énergie de l'émission d'un mobile quand il reçoit des données seulement. La bande passante asymétrique peut limiter le débit de TCP [15, 17, 19].

III.1.3. Capacité limitée des liens

Les canaux de communication sans fil offre de petite bande passante par mémoire aux réseaux filaires. Dans les environnements mobiles et spatiaux, ce problème est couplé avec la contrainte de l'énergie limitée et l'efficacité de bit qui est important en termes de coût de la transmission ainsi que en termes de capacité de lien. Le nombre de bit-overhead est important dans TCP, en particulier lors de l'utilisation de petits segments pour augmenter la probabilité de réussir la transmission d'un paquet [15].

III.1.4. Connectivité intermittente

Pour les satellites en orbite non géostationnaire, la connectivité sur un lien de communication est généralement intermittente. Des communications peuvent être interrompues pour un certain nombre de raisons, y compris handover, changement de la topologie du réseau, météo, et la dynamique orbitale [15].

III.2. XTP : Xpress Transport Protocol

A la fin des années 1980, les limitations du protocole TCP/IP sont bien connues et il y avait un grand intérêt dans le développement de quelque chose de mieux. Le Forum XTP est un organisme non lucratif composé de chercheurs, de développeurs, et d'utilisateurs fondé en 1992 afin de superviser, d'élaborer et de discuter de la spécification de XTP. XTP est un standard ouvert, qui peut être librement téléchargé sur le site du Forum XTP et utilisé sans aucune contrainte [3, 4].

III.2.1. Définition du protocole XTP

XTP combine les fonctionnalités des couches réseau et transport du modèle ISO OSI en une seule couche. Les objectifs de XTP sont de répondre aux besoins des systèmes distribués, temps réel et multimédias. XTP est un protocole orienté connexion [6].

Dans les protocoles de transport classiques comme TCP, le problème est le *bottleneck* situé au récepteur qui gère un ensemble de *timers* pour acquitter et superviser le contrôle de flux. Pour résoudre ce problème, XTP pencher sur une partie des fonctions de contrôle (des erreurs et contrôle de flux) du récepteur à l'émetteur, ce qui réduit le nombre de *timers*. L'émetteur XTP doit demander périodiquement des informations sur l'état du récepteur pour superviser et gérer l'association [1].

En XTP, une connexion est appelée une association, et considérés comme une paire de contextes actif, avec un contexte, à chaque extrémité. Le contexte indique les informations d'état qui représentent un instant d'une communication active entre deux extrémités XTP. Il est important de noter que le contexte doit être créé avant l'envoi et la réception des paquets XTP. Une fois l'association est arrêtée, les contextes reviennent à l'état *null* [6].

XTP supporte les deux mécanismes *unicast* et *multicast*. Dans le mode *unicast*, une association est un flux *full-duplex*. *Multicast* est conçu pour transférer des messages d'un émetteur à plusieurs récepteurs simultanément. Dans le mode *multicast*, le contexte d'émetteur est associé à de multiples contextes du récepteur. Chaque association entre un émetteur et un récepteur (dans un seul sens) n'est qu'un simple flux, c'est à l'émetteur au récepteur [6].

III.2.2. Structure du protocole XTP

Dans XTP, il y a sept types de paquets, qui utilisent une syntaxe fixe d'en-tête pour des mécanismes d'échange d'information et de contrôle dans une association entre un émetteur et un récepteur [1].

III.2.2.1. Segment de contrôle

Le segment de contrôle donne des mémoires sur l'état du contexte qui l'a envoyé. Les paquets XTP qui contient un segment de contrôle sont désignés comme des paquets de contrôle. Les paquets qui contiennent un segment de contrôle sont : CNTL, ECNTL, et TCNTL. CNTL c'est un paquet de contrôle qui transmet des informations de contrôle telles que : les valeurs de la fenêtre de contrôle de flux à travers un segment de contrôle commun (*Common Control segment*). Le paquet ECNTL transmet des informations de contrôle d'erreur par le biais de son segment de contrôle d'erreur (*Error Control segment*). Le paquet TCNTL est utilisé pour la négociation de la spécification du trafic par son segment de contrôle du trafic (*Traffic Control segment*). Ces trois types de paquets sont responsables des échanges des informations d'état entre les contextes [1, 3, 16].

III.2.2.2. Segment d'information

Un segment d'information contient les données de l'utilisateur et les informations de diagnostic. Les paquets XTP qui contient des segments d'information sont des paquets de données. Ce segment contient les données des couches supérieures ou des messages de la couche transport. Quatre paquets utilisent un segment d'information. Le paquet FIRST est un paquet d'initialisation d'une association et il contient un segment d'adresse, une spécification du trafic, et un segment de données optionnel. Les paquets DATA sont utilisés pour transférer des données et ils ne contiennent que des segments de données. Les paquets FIRST et DATA sont les deux types de paquets responsable de transfert de données utilisateur. Un paquet JOIN qui est utilisée pour joindre une association de *multicast* encours. Son format est le même que celui du paquet FIRST sans le segment de données optionnel. Finalement, le paquet DIAG utilise un segment diagnostique pour transmet des informations de diagnostique [1, 3, 16].

III.2.3. Timers XTP

Il y a quatre *timers* qui facilitent les procédures de contrôle du protocole. Les pertes de paquets sont découverts à l'aide du WTIMER et les pertes d'associations est découvert à l'aide du CTIMER. Le timer CTIMEOUT surveille une synchronisation poignée de main (handshake). Quand un paquet de contrôle est envoyé, sa valeur de champ de synchronisation est enregistrée dans la variable *saved_sync*, et le WTIMER est armé. Si le WTIMER expire avant l'arriver d'un paquet de contrôle dont la valeur de champ *echo* est égale à la valeur *saved_sync*, le contexte entre dans la procédure de synchronisation de handshake, et la CTIMOUT est démarré. L'objectif est de sonder (probe) le récepteur avec les paquets de contrôle à l'augmentation exponentielle des intervalles de temps jusqu'à ce qu'il y a un succès de handshake ou le CTIMOUT est expiré et l'association est interrompu. Pas de paquets portant des données peuvent être envoyés au cours de la synchronisation de handshake, y compris la retransmission de données; la retransmission à lieu une fois le handshake est terminé [1, 3, 16].

RTIMER est un timer de contrôle de taux servant à gouverner la fréquence de l'envoi des bursts de données. Le contrôle de taux régissant la relation producteur-consommateur entre les terminaux XTP. Le contrôle de taux est préoccupé par la rapidité avec laquelle les paquets et leur contenu peuvent être traitées ou consommés au niveau du récepteur. Le taux de sortie de paquets est réglé par deux champs de contexte qui sont *rate* et *burst* et de segment de spécification de trafic (Traffic Specification Segment). La valeur de *rate* indique le taux maximum de données en octets par seconde. La valeur de *burst* spécifie le nombre maximal d'octets à être envoyés dans un burst de paquets. La valeur de burst divisé par la valeur de taux donne le RTIMER [1, 16].

WTIMER est le timer qui assure une protection contre la perte d'un paquet avec un ensemble de bit SREQ. Chaque fois quand un paquet est envoyé avec un ensemble de bit SREQ, l'émetteur augmente sa valeur *saved_sync* par un et place là dans le champ de synchronisation du paquet. Le WTIMER est le temps que l'émetteur attend l'arrivée des paquets de contrôle en réponse à sa requête [1, 16].

Le CTIMER est le timer qui garantit que l'autre extrémité de l'association est encore en vie. Si le CTIMER expire et il y a des données à transférer, le CTIMER est rechargé. Et s'il n'y a pas de données à transférer, le CTIMER est rechargé et le contexte entre dans une synchronisation de handshake pour faire en sorte qu'il n'existe pas de données à transmettre ou retransmettre [1, 16].

III.3. SCPS-TP : Space Communications Protocol Standards Transport Protocol

SCPS (*Space Communications Protocol Standards*) est une suite de protocoles, couvrant de couche 3 de l'ISO et au-dessus, développer par le département américain de la Défense et la NASA pour répondre aux besoins des communications spatiale [4, 17]. Les protocoles ont été créés comme des spécifications militaires et normalisé par l'ISO dans le cadre du *Consultative Committee for Space Data Systems* (CCSDS). La suite des protocoles SCPS et leurs couches est représenté par la Figure III.1.

<table>
<tr><td>7: Application</td><td>SCPS-FP</td></tr>
<tr><td>6: Presentation</td><td></td></tr>
<tr><td>5: Session</td><td></td></tr>
<tr><td rowspan="2">4: Transport</td><td>SCPS-TP</td></tr>
<tr><td>SCPS-SP</td></tr>
<tr><td>3: Network</td><td>SCPS-NP</td></tr>
<tr><td>2: Link</td><td>CCSDS Telemetry and Telecomm and Frames
CCSDS Channel Code</td></tr>
<tr><td>1: Physical</td><td>CCSDS RF and Modulation</td></tr>
</table>

Figure III.1. Protocoles SCPS et leurs couches

Le protocole SCPS-TP est constitué de la norme TCP complétée par un ensemble d'extensions et d'améliorations qui consistent en l'implémentation et la spécification des changements. Ces modifications répondent aux exigences des caractéristiques de l'environnement spatial.

III.3.1. Différentes sources de perte

La clé pour l'optimisation des performances de débit TCP est d'identifier la source de perte de paquets et de réagir de façon appropriée. Il existe au moins trois sources de perte dans les environnements spatial et mobile : congestion de réseau, corruption et coupure de lien [15]. La réponse appropriée pour chacune des ces pertes est différente et SCPS-TP implémente pour chacune une réponse distincte. SCPS-TP a deux mécanismes pour déterminer la source de perte d'un paquet. TCP considère par défaut toutes les pertes sont causées par la congestion, mais SCPS-TP a un paramètre qui peut être fixé par un gestionnaire de réseau ou une application sur une route. Dans certaines circonstances, lorsque la bande passante du réseau est géré avec soin sur des liens privés, la congestion est peu probable et il est raisonnable pour SCPS-TP à assumer par défaut que toute perte due à des erreurs.

Le second mécanisme pour identifier la source de perte est la signalisation explicite. L'hôte de destination ou d'autres éléments de réseau (routeurs ou stations terrestre) envoie des signaux explicites au transmetteur TCP concernant la source de perte des paquets [15].

III.3.1.1. Perte causée par la congestion

Lorsque le contrôle de congestion est activée, les SCPS-TP utilise l'algorithme de contrôle de congestion TCP Vegas pour minimiser la perte et de faciliter l'utilisation de grandes fenêtres. L'algorithme de contrôle de congestion TCP Vegas élimine la nécessité de considérer la fenêtre de réception comme une limite supérieure de la taille de la fenêtre de congestion. Le contrôle de congestion TCP Vegas évite la congestion du réseau, sans trop élever le lien pour trouver le point de saturation, même en opérant avec de grandes fenêtres. SCPS-TP utilise une variante de l'algorithme Slow Start de TCP Vegas [15, 17], qui double la fenêtre de congestion tous les RTT, par opposition à TCP. SCPS-TP modifie cet algorithme en fournissant un déclenchement additionnel pour la transition de la phase de croissance exponentielle de la fenêtre de congestion à la phase de croissance linéaire pour éviter la congestion. En outre, SCPS-TP entre dans la phase d'éviter la congestion, une fois la taille de la fenêtre de congestion atteint la bande passante.

III.3.1.2. Perte causée par la corruption

Sur les liens spatiaux, il est souvent le cas où les BER sont la principale cause de perte de paquets. Dans ce cas, SCPS-TP désactive son hypothèse par défaut que la congestion est la source de la perte et ne pas invoquer le contrôle de congestion en réponse à la perte de paquets. Pour éviter la sur-gestion de la capacité du lien, SCPS-TP utilise une boucle ouverte, mécanisme de contrôle de taux par jeton [15], où la transmission à un taux déterminé. A chaque point terminal, le taux autorisé pour chaque lien est un paramètre de gestion qui est stocké dans une structure de routage accessible pour tout le monde. Sur un hôte, la capacité disponible pour un lien particulier est partagée entre toutes les connexions SCPS-TP utilisent ce lien.

III.3.1.3. Coupure de lien

Une coupure de lien est une perte de connectivité, résultant de passage temporaire d'un satellite hors de vue d'une station terrestre, un handover dans un réseau mobile avec des changements topologique dynamique, ou d'autres interruptions de courte durée. En général, une coupure de lien peut être identifiée à une station terrestre par la perte d'un *carrier lock* ou le signal reçu baisse en dessous d'un seuil. Une fois la station terrestre détecte la coupure d'un lien, elle envoie un message de coupure de lien ICMP à tous les hôtes de son côté. L'émetteur SCPS-TP répond au ICMP par la rentre en mode persistent, qui envoi périodiquement des paquets *probe*. SCPS-TP sort de l'état persistent quand il reçoit un message ICMP de restauration de lien, ou lorsque l'un des probes passe et est reconnue. Lorsque la liaison est rétablie, l'émetteur SCPS-TP reprend la transmission à partir de numéro de séquence qui se trouve dans l'en-tête TCP du message ICMP de coupure de lien.

III.3.2. Canaux asymétriques

TCP repose sur la transmission d'un ACK pour chaque paquet reçu. Les liens de communication entre les satellites et la Terre ont généralement un taux haut dans un sens et un taux beaucoup plus faible dans le sens inverse. Si un satellite transmettre de données sur un canal, le volume des ACKs TCP généré peut facilement dépasser la capacité du canal inverse [15]. Pour surmonter ce problème, SCPS-TP supprime l'obligation d'envoyer un ACK pour chaque paquet reçu du protocole TCP. SCPS-TP envoie un ACK pour chaque segment quand il existe un hors-séquence (*out-of-sequence*) des segments dans la file d'attente. Le récepteur SCPS-TP envoie un ACK chaque période de temps configurable.

III.3.3. Capacité limitée des liens

SCPS-TP utilise deux mécanismes pour améliorer les performances dans un environnement avec une bande passante limitée [15] : compression d'en-tête SCPS-TP et l'option Selective Negative Acknowledgment (SNACK).

III.3.3.1. Compression d'en-tête SCPS-TP

La compression d'en-tête TCP/IP est effectuée à la couche liaison, où les tables d'état de la connexion sont maintenues pour les connexions TCP/IP entrantes et sortantes. L'état de chaque connexion est composé des dernière en-têtes non compresser TCP et IP envoyés (sortant) ou reçus (entrant) sur cette connexion. Le compresseur initialisé par l'attribution d'un identifiant de connexion pour la connexion et l'enregistrement des première en-têtes TCP et IP envoyé. Ultérieurement les en-têtes sont construits en envoyant uniquement les modifications de l'en-tête précédent. A la réception, des en-têtes non compresser sont créés par l'application des modifications contenues dans le nouveau comprimé reçu à l'en-tête sauvegarder. Des copies des nouvelles en-têtes sont sauvegardée dans la table d'état de la connexion et des en-têtes non compresser sont transmis à la destination avec les données.

La compression d'en-tête SCPS-TP a été conçu pour une utilisation sur des liaisons série à faible vitesse. Elle fonctionne dans la couche transport, et elle réduit la taille des en-têtes TCP d'environ 50%. Elle fonctionne en résumant l'information statique, pour toute la durée de la connexion et en omettant les informations qui ne sont pas pertinentes pour le segment en cours d'envoi. L'en-tête SCPS-TP actuelle est de longueur variable, mais toujours contient un identifiant de connexion, un bit qui indique quand est-ce des champs optionnels sont présentés et quand est-ce des drapeaux sont définis, et un checksum. Contrairement à la compression d'en-tête TCP/IP, la compression d'en-tête SCPS-TP ne fonctionne pas à la couche liaison, elle n'utilise pas de delta de codage et elle compresse seulement l'en-tête TCP et ne compresse l'entête IP.

III.3.3.2. SNACK SCPS-TP

Une deuxième extension qui améliore l'utilisation de bande passante limitée est le SNACK SCPS-TP. Cette option permet au récepteur d'informer l'émetteur SCPS-TP sur un ou plusieurs trous dans la file d'attente du récepteur. SNACK c'est un segment ACK qui identifie plusieurs trous dans le tampon récepteur. La capacité de continuer à transmettre dans la présence de perte de paquets est particulièrement importante lorsque la perte est causée par la corruption plutôt que de la congestion. SCPS-TP SNACK attire à la fois de TCP Selective Acknowledgment (SACK) et le TCP Negative Acknowledgment (NAK) [3, 15]. Ces deux options résoudre le problème dans lequel le débit diminue lorsqu'il existe plus d'un segment perdu dans TCP. SNACK est un ACK négatif, comme NAK, mais il est capable de spécifier un grand nombre de trous de manière efficace. Il s'agit d'une option de longueur variable qui se compose de 5 champs : le type et la longueur de tous les champs obligatoires dans TCP, suivi par les champs *offset* et *longueur* (chacun 16 bits) et un champ bit-vecteur (*bit-victor*) optionnel et de longueur variable. Le champ de longueur précise de la taille du trou. S'il existe un *out-of-sequence*, le récepteur scanne son tampon de réception, forme un SNACK et de l'envoyer dans des segments ACK. A la réception d'un SNACK, l'émetteur retransmet immédiatement tous les segments nécessaires.

III.4. STP : Satellite Transport Protocol

STP est une adaptation d'un protocole des réseaux ATM qui s'appelle *Service Specific Connection Oriented Protocol* (SSCOP). SSCOP est le résultat d'un effort international de normalisation de la période 1990-1994. A l'origine SSCOP était destiné à donner une couche de liaison fiable pour la signalisation ATM et une couche de transport fiable pour les données de l'utilisateur sur les réseaux ATM [20]. SSCOP est actuellement utilisé pour la signalisation ATM dans les deux interfaces UNI et NNI. Il n'est pas utilisé pour le transfert des données utilisateur. La spécification actuelle de SSCOP repose sur la garantie de livraison des données en séquence (*in-sequence*), ce qu'il ne convient pas à l'utiliser dans un réseau de datagramme.

III.4.1. Opérations du protocole

La principale fonction du protocole est le transfert fiable des données dans des paquets de longueur variable. L'émetteur envoie les paquets au récepteur, les stocker pour une retransmission potentielle jusqu'à ce que le récepteur les acquitter. STP à un mécanisme de répétition de requête automatique (*Automatic Repeat reQuest* : ARQ) utilise des acquittements sélectifs seulement. Les paquets sont numérotés séquentiellement, et l'émetteur ne retransmet que les paquets qui ont été explicitement demandées par le récepteur (c'est-à-dire, il n'ya pas de délai d'expiration : *timeouts*) [19, 20].

STP a quatre types de paquets pour le transfert de données et quatre autre pour l'établissement de la connexion et la déconnexion [19, 20]. *Sequenced Data* (SD) est tout simplement un paquet de longueur variable des segments de données de l'utilisateur, avec 24 bits pour un numéro de séquence et un *checksum*. Les paquets SD qui n'ont pas encore été acquitté sont stockés dans un tampon, avec un *timestamp* indiquant la dernière fois qu'ils ont été envoyés vers le récepteur. Pas de données de contrôle inclus dans les paquets SD; mais plutôt l'émetteur et le récepteur change des messages POLL et STAT (us). L'émetteur envoie périodiquement un paquet POLL au récepteur; ce POLL paquet contient un *timestamp*, et le numéro de séquence du prochain paquet SD en séquence à être envoyé. Le récepteur répond au POLL par l'émission d'un message STAT qui inclue le *timestamp*, la valeur actuelle de la fenêtre du récepteur et le numéro de séquence du paquet le plus élevé dans la séquence qui a été reçu avec succès, et contient une liste de toutes les lacunes (*gaps*) dans les numéro de séquence allant du numéro de séquence le plus élevé (*highest*) des paquets reçus au numéro de séquence indiqué par le paquet POLL. Le message STAT est similaire au concept d'un acquittement sélectif TCP (SACK), sauf que le message STAT mémoires tout l'état du tampon du récepteur (plutôt que les trois dernières lacunes dans un SACK), et le récepteur STP ne revient pas aux données acquittée précédemment (ce qui est permet pour TCP avec les acquittements sélectif). Finalement, le récepteur peut détecter des pertes des paquets de manière indépendante par l'intermédiaire d'un paquet USTAT (Unsolicited STAT). Ce paquet permet au récepteur de stimuler rapidement une retransmission, et cela permet d'effectuer moins d'échange de POLL et STAT. USTAT est un acquittement négatif explicite demandant la retransmission immédiate.

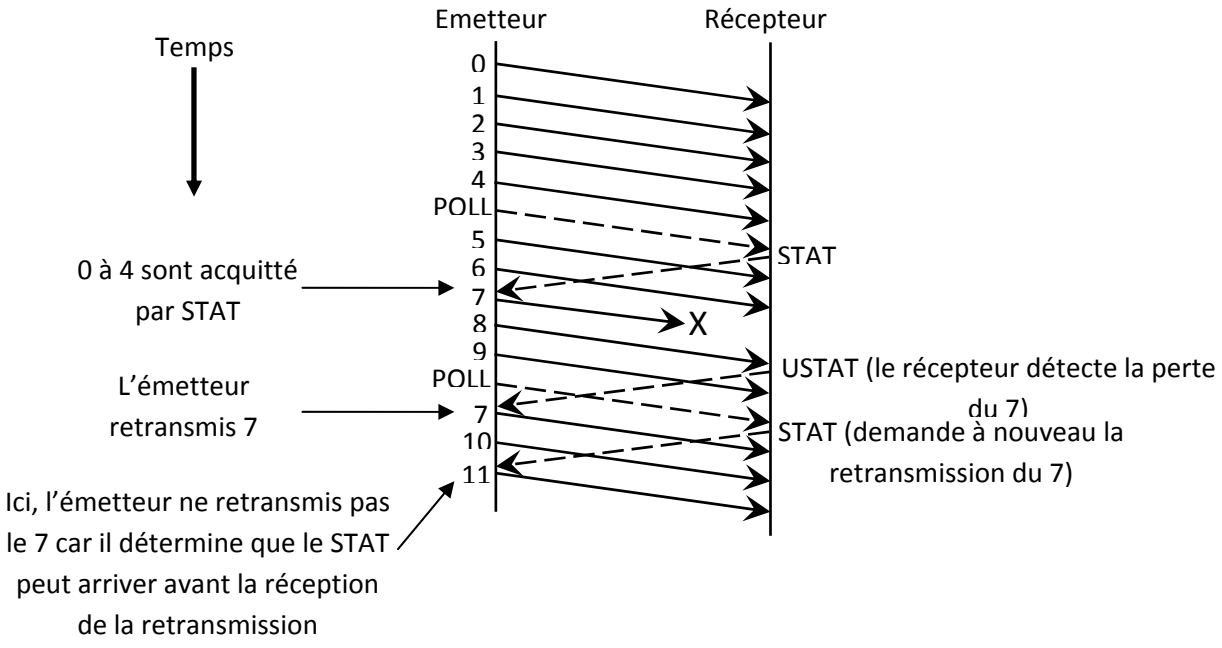

Figure III.2. Exemple d'opération STP

Le fonctionnement de base du STP (et de SSCOP) peut être mieux compris en considérant l'exemple illustré à la Figure III.2 [19, 20]. Cet exemple illustre une seule direction de transfert de données. Dans l'exemple, l'émetteur envoie une série de paquets numérotés consécutivement. Après que le paquet numéro 4 est envoyé, un paquet POLL est envoyé. Le POLL raconte au récepteur que le

message suivant à envoyer est numéro 5, de sorte que le récepteur sache qu'il doit avoir reçu les paquets de 0 à 4. Dans ce cas, puisque tous les paquets ont été reçus, le récepteur retourne un paquet STAT en acquittant tous les paquets et y compris le paquet numéro 4. Après l'envoi du POLL, l'émetteur se poursuit avec les paquets de 5 à 9; mais le paquet numéro 7 est perdu. Le récepteur détecte la perte après la réception du paquet numéro 8 et immédiatement demande la retransmission du paquet numéro 7 avec un paquet USTAT. Avant que cet USTAT soit reçu à l'émetteur, l'émetteur envoie un nouveau paquet POLL. Lors de la réception d'un USTAT, l'émetteur retransmet immédiatement le paquet numéro 7 et continue avec le paquet numéro 10 et numéro 11. Avant que la retransmission soit reçue, le récepteur répond au POLL en demandant à nouveau le numéro 7. Toutefois, le *timestamp* dans le paquet STAT permet à l'émetteur de déterminer que sa retransmission n'a pas encore eu l'occasion d'atteindre le récepteur, ce qui permet d'éviter une retransmission non nécessaire. Si le paquet numéro 7 est perdu de nouveau, le prochain message STAT aurait suscité une deuxième retransmission.

III.4.2. Formats des paquets

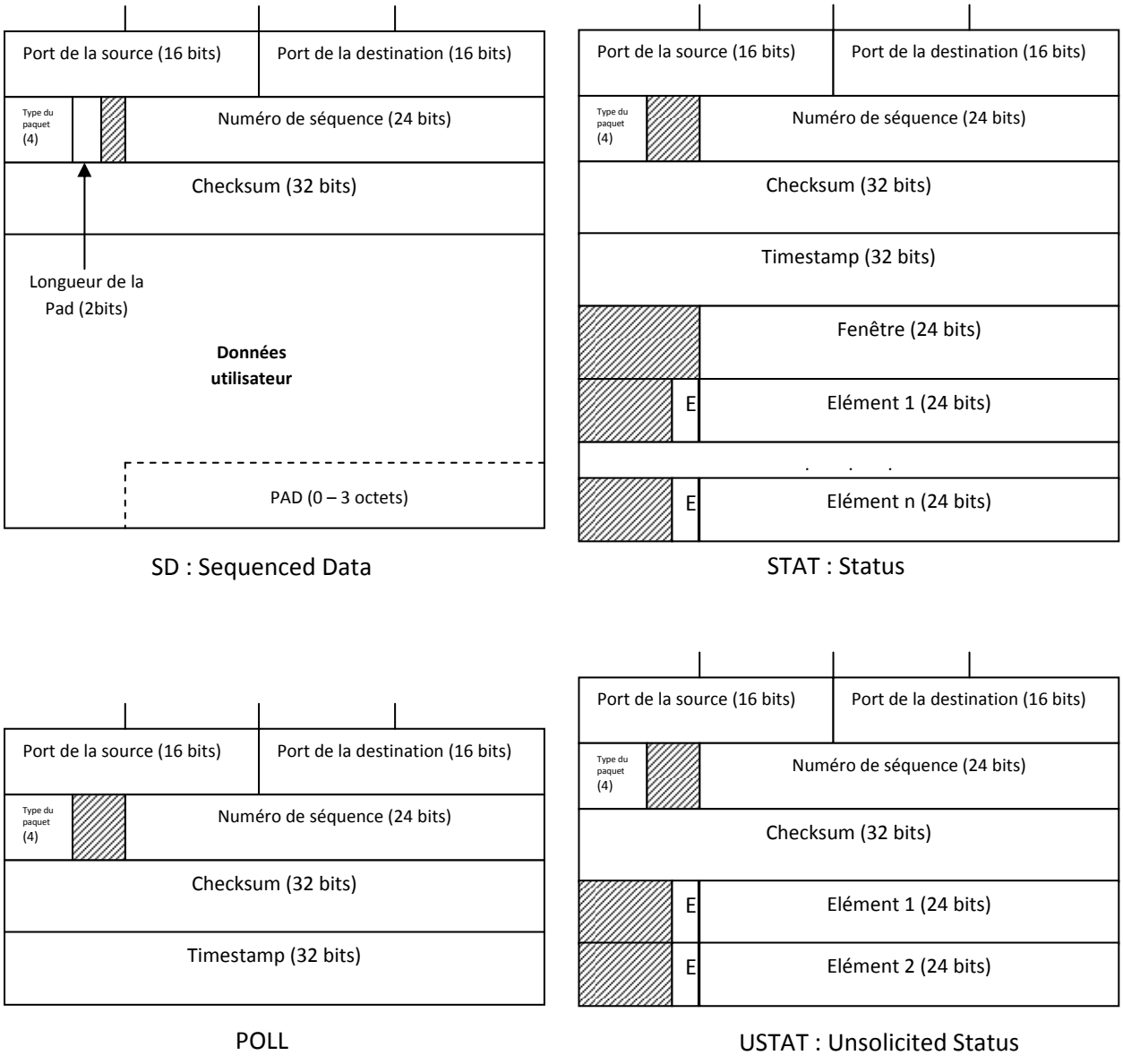

Figure III.3. Quatre types de paquets de base STP

Les formats des paquets STP sont illustrés dans la Figure III.3. Je décris d'abord les champs d'en-tête commun à tous les paquets. Comme dans le protocole TCP, chaque paquet a un ID de port source et destination, dans les 4 premiers octets. Le prochain octet contient 4 bit pour le type de paquet. Cet octet est suivi d'un numéro de séquence de 24 bits. Pour les paquets SD, il est supposé que les données de l'utilisateur sont alignées sur 32 bits. Si les données ne sont pas alignées sur 32 bits, jusqu'à 3 octets de *padding* sont ajoutés, et la longueur de ce champ PAD est codée dans les 2 bits qui suivent le type du paquet. Le champ *sequence number* dans le paquet SD contient le numéro de séquence du paquet. Pour le paquet POLL, le champ *sequence number* contient le numéro de séquence de prochain paquet dans la séquence à être transmis. Pour les paquets STAT et USTAT, le champ *sequence number* indique le numéro de séquence le plus élevé en séquence des paquets reçu. Le troisième champ de 32 bits de l'en-tête de paquet contient un *checksum* de 32 bits qui couvre l'ensemble du paquet STP. La sélection d'un *checksum* dépend de la protection de l'erreur des couches en bas. Les bits non utilisés dans les en-têtes sont réservés, ils sont codés par 0 par l'émetteur et ignoré par le récepteur [20].

En plus de ces champs communs d'en-tête, les paquets POLL, STAT, et USTAT contiennent des champs supplémentaires. Le paquet POLL encode une valeur *timestamp* correspondant au temps auquel le POLL est envoyé. Ce temps est repris dans le paquet STAT. Le paquet STAT contient aussi une valeur de la fenêtre du récepteur (mesurée en paquets). En plus, s'il y a des lacunes dans le tampon du récepteur, elles sont présentées dans une liste. Une paire de la liste des éléments caractérisant le début et la fin des numéros de séquence d'une lacune (gap). Si le récepteur découvre que la lacune était due à un défaut de contrôle (*checksum failure*) plutôt que l'absence du paquet, le bit E (Error) est utilisé. Les paquets STAT large peuvent être segmentés s'ils dépassent certaine taille configurée. Enfin, le paquet USTAT est une version simplifiée du STAT qui indique une seule lacune [20].

Conclusion

Les limitations de TCP sur satellites sont dues à la conception du protocole TCP, et en utilisant un protocole différent spécialement optimisé pour les satellites, il est possible d'améliorer les performances. Les protocoles satellitaires sont rarement utiles car ils ne sont pas compatibles avec les applications basées sur TCP. Les protocoles satellitaires trouvent leur utilisation réelle comme protocoles de passerelle, qui convertit le trafic TCP qui arrive des derniers nœuds d'un réseau IP à un protocole satellitaire pour l'utilisation des liaisons satellites.

Il n'y a pas un seul et un meilleur choix pour un protocole satellitaire. Différentes alternatives offrent divers compromis en matière de performance, de flexibilité et de facilité de mise en œuvre.

Dans ce chapitre nous avons décrit certains protocoles de transport pour les communications satellitaires. Le protocole qui nous intéresse est le protocole STP, puisque il est à l'origine du protocole XSTP dont nous allons étudier les performances. Par contre, le chapitre suivant va se baser sur les protocoles de transport utilisés pour les nanosatellites, et va décrire le protocole XSTP en détail.

CHAPITRE IV
PROTOCOLES NANOSATELLITES

Introduction

Des recherches se concentrent principalement sur l'amélioration du comportement des protocoles de transport sur les réseaux contenant des liaisons satellites LEO. Les réseaux sans fil et mobiles ont un ensemble unique d'erreurs de lien, y compris la corruption de bits, le handoff et la connectivité limitée. Malheureusement, la plupart des protocoles de transport, comme TCP, sont conçus pour traiter la congestion liée à des erreurs communes dans les réseaux filaires. Cette incapacité de gérer plusieurs types d'erreurs résulte la dégradation de débit effectif et des économies d'énergie.

Dans ce chapitre on va voir la description de deux protocoles de transport pour les liaisons satellites LEO et on va se baser principalement sur l'ajout d'une stratégie de contrôle d'erreur au protocole STP. On va voir aussi la conception du protocole XSTP et leur mécanisme probing.

IV.1. NSP : Nanosatellite Protocol

NSP est un protocole pour les nanosatellites, à l'origine développé à UTIAS/SFL pour l'utilisation sur les nanosatellites *CanX*. Ce qui est descendu du *Simple Serial Protocol* (SSP) utilisé par UTIAS/SFL et *Dynacon* sur les vaisseaux spatiaux MOST et CHIPSAT [13, 21].

Le protocole est conçu pour être simple afin de réduire les paquets d'*overhead* et maximiser l'utilisation de bande passante effective [21].

L'unité de base de la communication dans le NSP est le message. Un message du satellite ordinateur de bord (*On-Board Computer* : OBC) à un dispositif interplanétaire *Sinclair* porte une seule télécommande. Le dispositif répond alors à l'OBC avec un seul message de réponse. Cette réponse doit être un simple acquittement ou elle contient des télémétries additionnelles. Les messages NSP peuvent être mis sur un certain nombre de différents types de lien : asynchrone, synchrone et I2C. Large réseaux NSP contiennent différents types de liens avec des routeurs et des ponts pour les relier [13].

IV.1.1. Format du message

Les messages NSP sont composés d'une séquence ordonnée d'octets. Là où plusieurs octets adjacents sont regroupés ensemble pour former des grandes unités (16-bit en entier, 32-bit en virgule flottante, etc.) les octets les moins significatif sont transmis et reçu en premier.

Longueur	Champ
1 octet	Adresse destination
1 octet	Adresse source
1 octet	Message de contrôle
0 ou plus d'octets	Données
2 octets	Message CRC

Tableau IV.1. Champs du message NSP

Le Tableau ci-dessus montre les champs qui forment un message NSP. Le champ de données est de longueur variable. Si sa longueur est égale à zéro (le message ne comporte pas de données) alors le message total contient cinq octets. Le nombre maximal d'octets de données qui peut être placé dans un message dépend de l'implémentation du NSP dans chaque périphérique. 260 octets est une limite commune [13, 21].

Dans chaque message NSP il y a un octet pour un message de contrôle. Le Tableau ci-dessous montre comment les bits de cet octet sont décodés. Il y a trois booléens marqués A, B et Poll. Il y a aussi cinq bits pour un code de commande, interprété comme un entier non signé entre 0 et 31 [13, 21]. Chaque message NSP contient 2 octets (16-bit) CRC pour se prémunir contre les erreurs de transmission.

Numéro du bit	Etiquette
Bit 7 (MSB)	Poll
Bit 6	B
Bit 5	A (ACK)
Bit 4-0	Code de commande

Tableau IV.2. Champs du message de contrôle

IV.1.2. Format de la Télécommande

L'OBC envoie une télécommande aux dispositifs interplanétaires *Sinclair*. Ces messages ont les attributs suivants :

Attribut	Signification
Adresse destination	L'adresse du périphérique SI
Adresse source	L'adresse unique de l'OBC
Bit Poll	1 Si la réponse est demandée 0 Si aucune réponse n'est souhaitée
Bit B	Ignoré par le périphérique SI
Bit A	Ignoré par le périphérique SI
Code de commande	Commande désirée
Données	Octets de données
CRC	CRC, calculé à partir du message

Tableau IV.3. Attributs d'une télécommande

La liste des codes de commande, et le formatage des données pour les accompagner, est spécifique à chaque périphérique interplanétaire *Sinclair* [13].

IV.1.3. Validation de la télécommande

Lors de réception d'une séquence d'une télécommande, le périphérique Interplanétaire *Sinclair* suivra les étapes suivantes pour la validation de la télécommande [13] :

- L'adresse de destination sera comparé à l'ensemble des adresses que le périphérique peut accepter;
- Le message doit satisfaire le minimum de longueur du message (5 octets);
- Le message ne doit pas dépasser la longueur maximale du message, déterminée par l'implémentation du périphérique;
- Il ne doit pas avoir une erreur dans la formulation (framing) du SLIP;
- Le message du CRC doit être bon.

Les messages qui ne respectent pas l'un de ces critères sont rejetés. Les télécommandes valides sont transmises à la prochaine couche.

IV.1.4. Format de la réponse

Après l'exécution d'une télécommande, le périphérique Interplanétaire *Sinclair* va générer un message de réponse, si et seulement si le bit Poll de la télécommande à la valeur 1. La réponse aura les attributs suivants [13] :

Attribut	Signification
Adresse destination	L'adresse source de la télécommande
Adresse source	L'adresse destination de la télécommande
Bit Poll	1 dans toutes les cases
Bit B	Le bit B de la télécommande
Bit A	1 si le périphérique signale une condition ACK 0 Si le périphérique signale une condition NACK
Code de commande	Le code de commande de la télécommande
Données	Les octets de données de la télécommande, suivie de zéro ou plusieurs octets de télémétrie
CRC	CRC, calculé à partir du message

Tableau IV.4. Attributs d'une réponse de la télécommande

IV.1.5. Formulation du SLIP

Les messages NSP sont encapsulés dans des paquets en utilisant la formulation SLIP avant d'être transmis à un autre périphérique. Lors de la réception la formulation est enlevée [13]. Un caractère spécial FEND (0xc0) marque à la fois le début et la fin de chaque message NSP. Là où se produirait FEND dans le message, il est remplacé par deux octets : FESC TFEND (0xdb 0xdc). Où se produirait FESC dans le message, il est remplacé par FESC TFESC (0xdb 0xdd). Lors du traitement d'un message formulé par SLIP, c'est une erreur de voir FESC suivi par autre chose que TFESC ou TFEND.

IV.1.6. Adresses NSP

Les adresses NSP ont une longueur de huit bits. Les plages d'adresses recommandées sont comprises entre 1 et 127. Tout utilisateur d'un périphérique Interplanétaire *Sinclair* devrait prendre une adresse NSP pour le OBC (ou l'ordinateur de test GSE). Par convention 0x11 est utilisé comme adresse NSP pour l'ordinateur principal.

Certains périphériques Interplanétaire *Sinclair* ont une adresse NSP unique programmé dans l'usine. D'autres ont des broches sur leurs connecteurs qui peuvent être choisir entre un certain nombre d'adresses différentes. Enfin, les périphériques fonctionnant sur des bus synchrone (SPI) utilisent un adressage *out-of-band* et ne pas avoir une adresse NSP fixe [13].

IV.2. XSTP : eXtended Satellite Transport Protocol

Les deux réseaux sans fil et mobiles ont un ensemble unique d'erreurs de lien, y compris la *corruption de bits*, le *handoff* et la *connectivité limitée*. Malheureusement, la plupart des protocoles de transport, comme TCP sont conçus pour traiter la congestion liée à des erreurs communes dans les réseaux filaires. Cette incapacité à gérer plusieurs types d'erreurs résulte la dégradation de débit effectif et de l'énergie stockée. Une recherche qui a proposée un nouveau protocole de transport pour les satellites appelé *Satellite Transport Protocol* (STP) qui adresse aux problèmes des réseaux satellitaires seulement [20]. STP ne fait pas une différentiation entre les types d'erreur qui se produisent dans le réseau. Pour cela une autre recherche a optimisé le protocole STP par l'ajout d'une nouvelle stratégie de contrôle d'erreur qui s'appelle *probing* [8, 9, 10]. Ce dernier essaye de déduire la cause de l'erreur. L'optimisation du protocole STP a donné naissance à un nouveau protocole de transport qui s'appelle : *eXtended Satellite Transport Protocol* (XSTP).

IV.2.1. Architecture générale de XSTP

Le protocole XSTP peut être déployé au dessus d'un protocole réseau tels que *Internet Protocol* (IP). XSTP fournira également un service de streaming fiable au niveau des protocoles d'application tels que *File Transfer Protocol* (FTP). Une session XSTP a des sessions plus bas et d'autres plus haut. Une configuration typique d'une suite de communication qui comprend XSTP est illustrée à la Figure IV.1. Quand une telle suite est initialisée, une instance du protocole XSTP est crée, configurée et installée dans l'emplacement approprié dans la hiérarchie des protocoles [10].

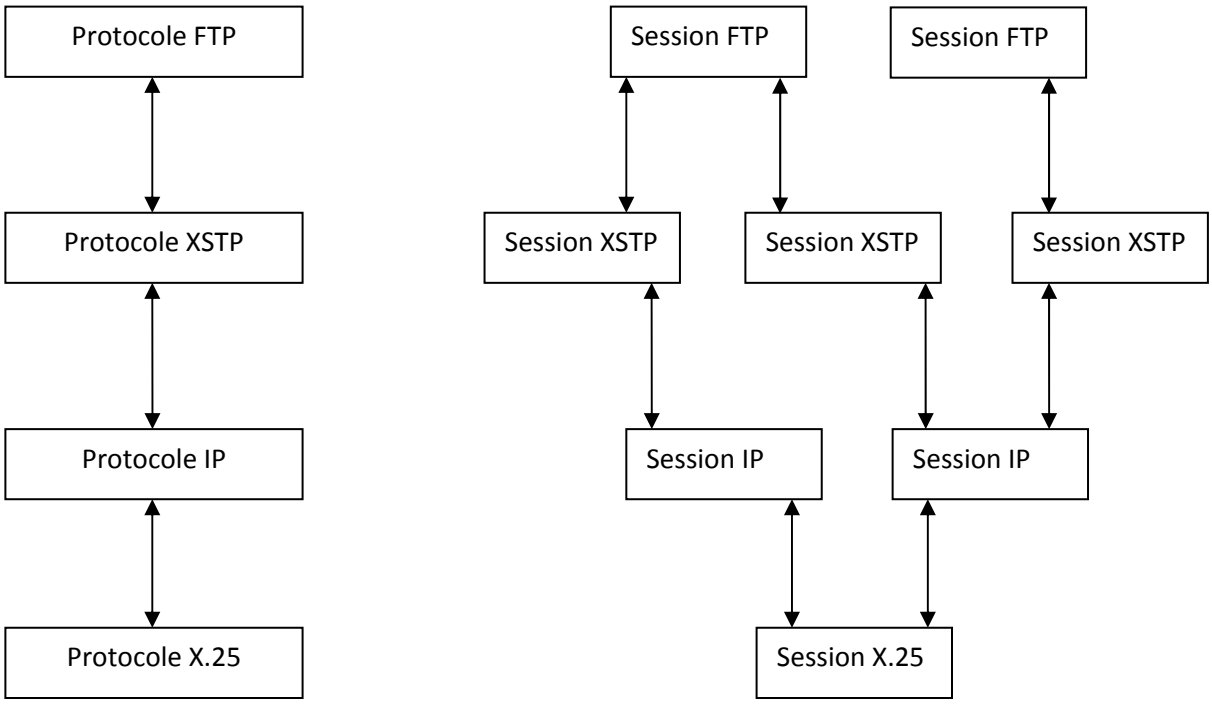

Figure IV.1. Exemple de configuration des protocoles et sessions inclue XSTP

Il y a une séparation claire entre le concept d'un protocole de celui d'une session. L'abstraction d'un protocole définit la sémantique de base de la représentation du protocole comme son schéma d'adressage. Il définit également les règles de la gestion des sessions du protocole. L'abstraction d'une session d'autre part représente une instance de connexion. La relation entre le protocole et la session est qu'un protocole peut posséder une ou plusieurs sessions (voir Figure IV.1). Une session XSTP joue essentiellement le rôle d'un émetteur et d'un récepteur. Pour cette raison les fonctionnalités de la session sont prises en compte par deux classes : *un émetteur XSTP* et *un récepteur XSTP*. Une instance de chacune de ces deux classes est crée pour gérer une session. Ces deux instances jouent le rôle de l'envoi et

de la réception de la session. La conception complète du protocole XSTP est donnée par la section suivante.

IV.2.2. Conception détaillée

IV.2.2.1. Classe émetteur

La classe émetteur est appelée *XSTP*. Cette classe est responsable de la construction et la transmission des messages et l'implémentation de la stratégie de contrôle de flux. La session passe les données qui ont arrivé de la session application à un objet émetteur. L'émetteur ne transmet pas un segment de données immédiatement. Au lieu de cela, il copie les données dans un buffer de transmission. S'il n'y a pas assez d'espace pour les données dans le buffer, alors le message est suspendu. On le traite plus tard lors de la disponibilité d'espace dans le buffer. Dès qu'il y a de nouvelles données en buffer, l'état de l'émetteur est mis à jour [8, 9, 10].

A tout moment, l'émetteur est soit dans l'état de transmission, persistance ou inactif. L'émetteur entre dans l'état de transmission quand il y a suffisamment de données à transmettre et il y a de la place dans la fenêtre de transmission. La fenêtre de transmission est régie par une stratégie de contrôle de fenêtre qui contrôle le nombre de segments qui peuvent être transmis à tout moment. Cette fenêtre est calculée comme étant le minimum entre la fenêtre du récepteur et la fenêtre de congestion de l'émetteur en utilisant la méthode *int window()*. Le taux de transmission de l'émetteur n'est pas basé sur le taux d'arrivée des acquittements (auto-synchronisation), mais plutôt sur un temps d'envoi *del_send_timer_*. Ce temps permet de réduire la possibilité d'introduire d'énormes bursts dans le réseau. Chaque fois quand l'émetteur est entré dans l'état de transmission, il met à jour le nombre total de segments permis a envoyé et l'intervalle du temps de transmission. Si le nombre de segments autorisé est au dessous d'un certain seuil configurable (*double rate_*), l'émetteur transmet les tous dans un seul burst.

En plus de la transmission de données, les émetteurs XSTP envoient des segments POLL périodiquement. Ces segments sont envoyés pour déclencher la transmission des segments STAT contenant des acquittements pour les segments reçus. Le taux de *Polling* est un paramètre configurable. Un taux de *Polling* plus élevé implique une obtention plus rapide des acquittements des segments, mais aussi beaucoup d'*overhead* sont utilisés. A chaque instant l'émetteur vérifier s'il y a une transmission d'un segment de données, un segment POLL ou les deux.

Un émetteur XSTP transmet un burst de données en créant des segments de données et de les transmis un à la suite de l'autre. Tous les segments dans un burst sauf le dernier ont la taille maximale autorisée à un segment. Si un POLL est prévu en même temps d'un burst de données, le POLL est concaténé avec le dernier segment du burst. L'émetteur peut également forcer un POLL a concaténé avec un segment de données si l'émetteur a transmis un certain ratio configurable de données depuis le dernier POLL. Ce processus est fait pour accélérer la réception des acquittements, donc de libérer rapidement le buffer. L'émetteur maintient une file d'attente pour les segments non acquitter appelée *trans_buffer[B_SIZE]*. Avant de transmettre n'importe quel segment, l'émetteur fait une nouvelle entrée pour ce segment dans le buffer. L'entrée contient le temps de transmission ou de la dernière retransmission *timestamp*, le numéro de séquence du segment *seqno_*, un champ d'acquittement *ack_* et un drapeau *flag_* qui indique si le segment est ustaté ou non.

Dans certains cas, l'émetteur peut ne pas être capable de ne transmettre aucun nouveau segment de données. Cette situation peut se produire si un récepteur lent annonce une taille zéro de sa fenêtre de réception afin d'éviter une transmission rapide des segments de données. Dans ce cas l'émetteur sort de l'état de transmission et va à l'état de persistance. Dans cet état l'émetteur suspend la transmission de données et transmit les segments POLL périodiquement pour demander des mises à jour de la fenêtre. C'est seulement après réception d'une taille de fenêtre non nulle l'émetteur peut revenir à l'état de transmission. Lorsque l'émetteur n'est ni à la transmission, ni à la persistance, il est dans l'état inactif. Lorsque l'émetteur est dans cet état la session peut être fermée. Une session peut forcer l'émetteur d'entrer dans l'état inactif en appelant la méthode *void finish()*. Ce processus se fait quand une session est terminée.

Afin de faciliter la transmission de tout type de segment, l'émetteur utilise la méthode *void send_data()* qui détermine le nombre de données permis à envoyer et les deux méthodes *void send_poll()*

et *void send_sd(int seqno)* qui envoient les segments POLL et les segments de données respectivement. Ces méthodes remplissent les en-têtes et collectent toutes les informations nécessaires à chaque type de segment. Après avoir collecter toutes les informations nécessaires, les méthodes crée un nouveau message dont la taille est égale à celle du segment.

Une autre responsabilité de l'émetteur est de gérer la stratégie de contrôle de flux. Cette stratégie est affectée par des *feedback* reçus du récepteur. Ces *feedback* peuvent être soit sous la forme d'un segment STAT ou USTAT. Ces segments de statut contiennent à la fois *cumulative* et *selective negative acknowledgments*. Le *cumulative acknowledgment* informe l'émetteur sur le prochain segment qui manque dans la séquence. Cela permet à l'émetteur d'augmenter la fenêtre de transmission en utilisant la méthode *void opencwnd(int num)* et de supprimer des entrées du buffer. L'émetteur libère les entrées du buffer qui appartiennent à tous les segments acquittés. Après ce processus, l'émetteur ouvre sa fenêtre de congestion soit exponentiellement ou linéairement selon que l'émetteur est dans la phase *slow start* ou en phase *congestion avoidance* de contrôle du flux respectivement.

D'autre part le *selective negative acknowledgment* est essentiellement un mémoire des lacunes (*gap*). L'émetteur examine chaque segment mémoireé perdu et détermine si ces segments sont qualifiés pour la retransmission ou non. L'émetteur distingue les lacunes signalées par des segments STAT de celles signalées par des segments USTAT. Les segments signalés perdus par un USTAT sont retransmis directement, mais ceux signalés par un STAT ne peuvent être retransmis que si un RTT (sorte d'un *backoff*) s'est écoulé depuis leur dernière transmission. Pour retransmettre un segment l'émetteur doit trouver son entrée dans le buffer. L'émetteur utilise les informations trouvées dans l'entrée pour créer une autre copie du segment de données perdu. Après l'examen de toutes les lacunes et de faire les retransmissions nécessaires, l'émetteur ferme sa fenêtre de congestion en coupant en deux par la méthode *void closecwnd(int how)*. D'autres retransmissions dans la même fenêtre ne doit pas aboutir à de nouvelles fermeture de la fenêtre de congestion, étant donné que ces retransmissions sont probablement causés par le même événement de perte.

IV.2.2.2. Classe récepteur

La classe récepteur est appelée *XSTPSinker*. Cette classe peut être décrite comme une machine d'état qui contrôle la session. Elle conserve toutes les connaissances sur les transitions possibles de la machine d'état. Ces transitions sont généralement déclenchées par la réception des segments en utilisant la méthode *recv(Packet*, Handler*)* [8, 9, 10].

Pour qu'un segment est accepté, il doit avoir réussi les tests de contrôle (*checksum*) et d'adressage effectué par le protocole en utilisant la méthode *process_sd(Packet* pkt)*. Une fois le segment est validé, des informations de base et des options sont extraites de l'en-tête avant d'inspecter le type du segment. XSTP traite le segment selon son type simple ou composite. XSTP permet de combiner certains types de segments pour sauver la bande passante du réseau en réduisant le nombre de segments transmis.

Pour chaque type de segment, le récepteur définit une méthode de gestion. On passe à chaque méthode autant d'informations du segment dont elle a besoin. Le récepteur effectue également au moins une des actions suivantes : déclenche une transition d'état, ordonne certain segment et demande certain post-action. Une demande de transmettre un segment composite est d'abord transmis à l'émetteur suivi de l'exécution des post-actions demandées.

Une autre responsabilité importante du récepteur XSTP est le montage et la présentation des données aux protocoles hauts. Si un segment de données arrive, le récepteur applique certaines sémantiques de tests au segment. Selon les résultats de ces tests, le segment est soit ignoré, présenté ou sauvegardé dans le buffer du récepteur. Le buffer est essentiellement une liste triée des segments *out-of-sequence*. Si le segment en buffer est également le prochain segment attendu, le récepteur vérifie s'il remplit la première lacune dans le buffer. Si c'est le cas, un nouveau message de données est créé pour contenir toutes les données présentées du segment et du buffer et est transmis au protocole de la couche supérieure. Si le segment n'est pas le prochain qui manque, le récepteur le copie dans le buffer dans l'ordre approprié.

Comme dans STP, le récepteur XSTP a la responsabilité d'envoyer des notifications USTAT à l'émetteur quand une nouvelle lacune est découverte en utilisant la méthode *check_for_ustat()*. La lacune indiquée est décrite par ses numéros de séquences inférieures et supérieures. Dans un réseau où la

réorganisation des paquets est possible, le récepteur peut être configuré de façon à retarder cette notification pour un certain nombre de segments.

La responsabilité finale d'un récepteur XSTP est de traiter les acquittements des segments. Le récepteur utilise le *timestamp* du segment pour s'assurer qu'il traite uniquement les nouveaux segments. Quand un segment POLL est reçu, le récepteur traite ce segment en utilisant la méthode *process_poll(Packet* pkt)* et demande la transmission d'un segment STAT en utilisant la méthode *send_stat(double tstamp)*. Ce segment comporte un mémoire des lacunes construit par la traverse du buffer du récepteur et la création d'une liste des lacunes. Une lacune est facilement reconnue lorsque deux segments dans le buffer n'ont pas des numéros de séquence consécutifs.

IV.2.3. Mécanisme probing de XSTP

IV.2.3.1. Motivation

Dans un réseau hétérogène, il en existe de nombreux types d'erreurs variables dans leur nature. L'un des problèmes classiques des protocoles de transport standards dans ces réseaux est leur incapacité de détecter et de réagir efficacement à ces différentes conditions d'erreur. L'hypothèse de base faite par ces protocoles est que la congestion du réseau est la cause de toutes les erreurs. Dans un réseau d'accès par satellites LEO, il existe d'autres types de conditions d'erreur y compris la corruption de bit, le handover et la connectivité limitée. Ces erreurs qui sont liées au lien et non à la congestion sont systématiquement perçues comme liées à la congestion par les protocoles de transport qui ignorent ces types d'erreurs. L'engagement de contrôle de congestion dans ces cas diminue le débit. Aussi, basé sur l'utilisation du contrôle de congestion peut augmenter le nombre de segments retransmis qui se traduit par l'augmentation de l'énergie dépensée [8, 9, 10].

Le protocole STP [20] hérite ce contrôle de congestion. Bien que le protocole puisse différencier entre ces types d'erreur, le contrôle de congestion peut avoir un effet négatif sur sa performance globale. Dans cette section une optimisation du protocole STP qui consiste à lui proposer une nouvelle stratégie de contrôle d'erreur pour lui permet de s'adapter aux différents types d'erreur qui se trouvent dans les réseaux d'accès satellitaires LEO.

Cette optimisation est basée sur le mécanisme probing et qui s'appelle XSTP-probing. Le mécanisme est déclenché lors de la détection d'une perte d'un segment pour évaluer le niveau de congestion dans le réseau. Si une congestion est détectée, le mécanisme répond par l'invocation du contrôle de congestion, sinon il restaure la fenêtre de congestion au même niveau qu'avant le probing.

IV.2.3.2. Description

Le mécanisme probing accomplit leur tâche par la suspension de la transmission de nouvelles données quand il détecte une perte et il lance un cycle de probing pour collecter un ensemble des RTT. Le mécanisme compare ces RTT au RTT disponible lorsque la perte a été découverte. Après la terminaison du cycle probing, s'il y a une congestion, le contrôle de congestion est immédiatement invoqué. Sinon, la transmission est restaurée sans prendre aucune action. La Figure IV.2 illustre l'algorithme de base du mécanisme XSTP-probing [8, 9, 10].

XSTP-probing exploite la même sémantique de STP et ne pas introduire de nouveaux types de segment. En fait, l'optimisation du protocole STP consiste en la réutilisation du cycle *polling* de STP comme son cycle de probing. Plus précisément, le segment POLL devient un *probe* et le segment STAT devient un *acquittement probe*. Le récepteur XSTP ne sait pas si le POLL reçu est un probe ou un POLL normale.

Le mécanisme est déclenché lorsque une perte est détectée, soit implicitement ou explicitement. La méthode implicite est appelée si un *timeout* est dépassé et cela s'il y a une perte d'un segment POLL ou STAT. Pour expliquer, un émetteur XSTP transmit un nombre configurable de segments POLL chaque RTT. Après le premier RTT les segments STAT commencent à arriver à l'émetteur. Le taux d'arriver des segments STAT devient semblable au taux d'envoi des segments POLL, la Figure IV.3 illustre le déclenchement de probing s'il y a une perte d'un POLL ou STAT. Si un POLL ou un STAT est perdu

dans la connexion, la session détecte cette perte entre le RTT/POLLS_PER_RTT minimum et le RTT maximum.

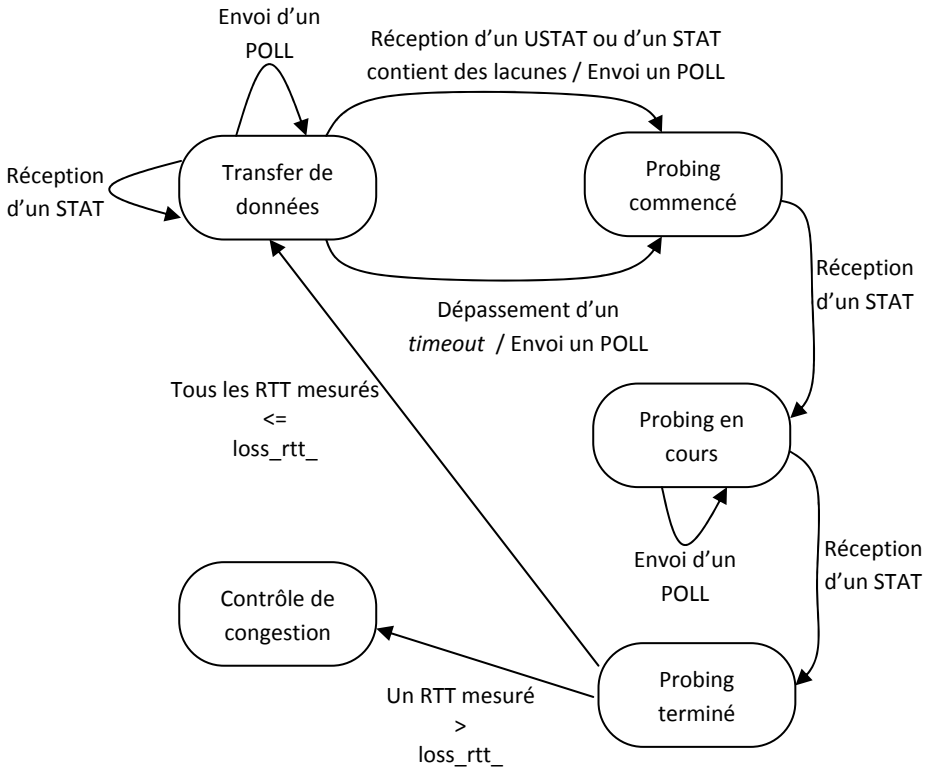

Figure IV.2. Diagramme d'état de XSTP

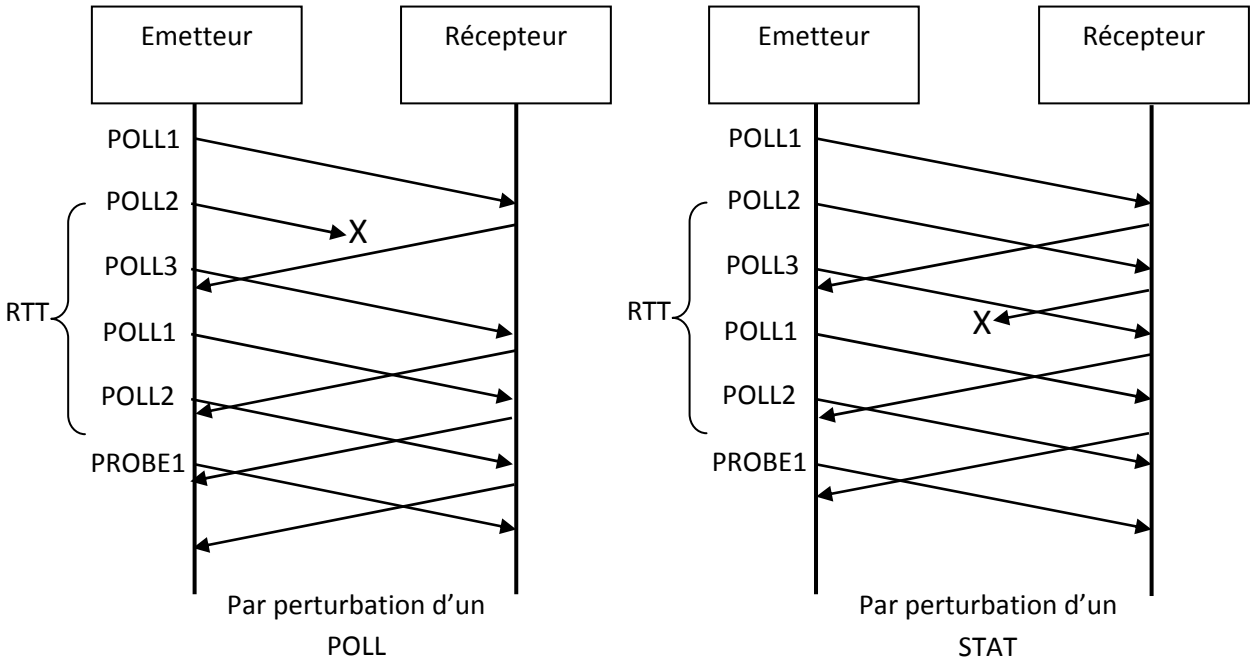

Figure IV.3. Déclenchement du mécanisme probing par la perte d'un POLL ou d'un STAT

D'autre part, la méthode de détection de perte explicite repose sur la réception des segments STAT ou USTAT. Le mécanisme probing est déclenché si et seulement si au moins un segment nécessite la retransmission. Cette condition protège le déclenchement du mécanisme suite à une fausse alarme.

Quand le mécanisme probing est démarré la session passe en mode probing. Ce mode annule et remplace l'état actuel de l'émetteur. Après cet événement, le mécanisme suspend la transmission de nouvelles données et enregistre le *timestamp* actuels et l'estimation du RTT (*loss_ts* et *loss_rtt*).

Le mécanisme envoie un segment probe chaque RTT, indépendamment de savoir si un acquittement probe est reçu ou non. L'avantage est évident lorsque le RTT est un peu étendu (un phénomène commun dans les liaisons par satellite LEO où le RTT varier beaucoup). Dans ce cas l'échange précédent n'est pas ignoré, mais plutôt on lui donne plus de temps (jusqu'à un RTT en plus) pour terminer. S'il n'est pas encore terminé après ce moment, un nouvel échange commence et le vieil échange devient obsolète (car il est suivie par deux nouveaux échanges de probes). Cette situation est illustrée dans la Figure IV.4, où PROBE1 est transmis en l'absence de l'arriver d'un STAT depuis le dernier POLL.

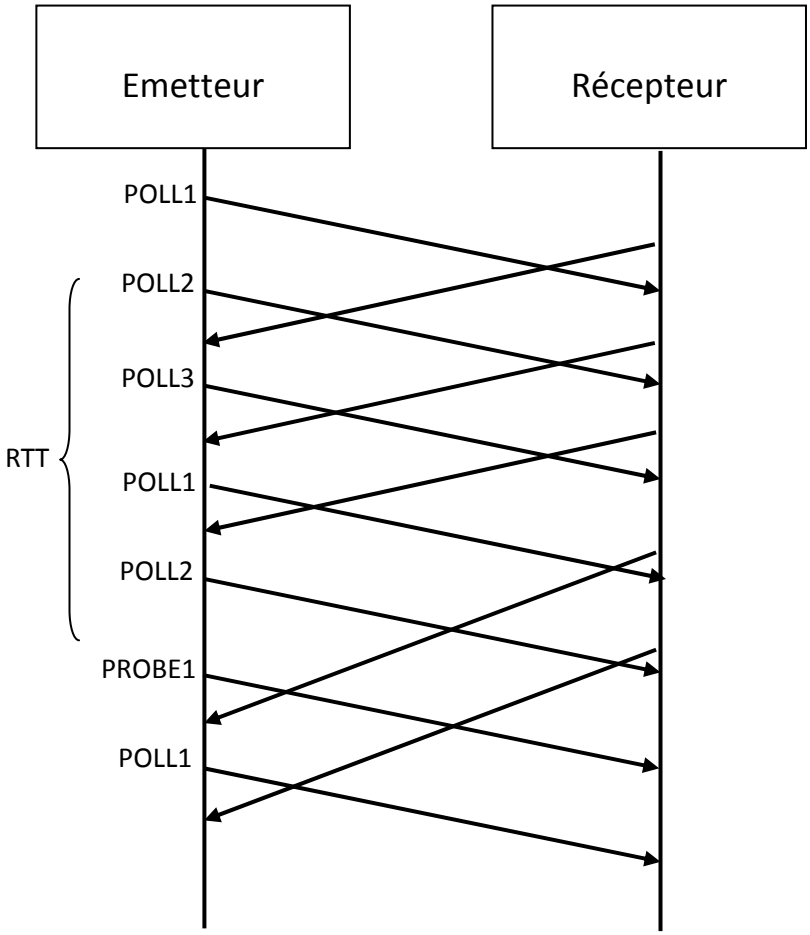

Figure IV.4. Déclenchement du mécanisme probing par un événement de fausse *timeout*

L'optimisation du protocole STP implémente la stratégie de contrôle d'erreur par la définition d'une table ordonnée (qui s'appelle *map*) entre les *timestamp* de l'envoi des probes (POLL) et son RTT mesuré de l'acquittement (STAT). Chaque fois qu'un probe est envoyé, son *timestamp* est enregistré dans la prochaine entrée vide dans la table *map*. Aussi, chaque fois qu'un acquittement du probe valide est reçu, son RTT mesuré est enregistré dans l'entrée correspondante dans la *map*. Le cycle probing ne se termine pas jusqu'à ce que deux entrées consécutives dans la *map* se remplissent avec des mesures de RTT. Il est important de mentionner que la *map* a une taille fixe qui est définie comme un paramètre configurable. La *map* dispose également d'une politique de suppression de l'entrée la plus ancienne pour

faire place aux nouvelles entrées. La Figure IV.5 illustre les différentes phases d'un cycle typique de probing comme elles se produisent dans le réseau. La Figure IV.6 montre les étapes correspondantes du cycle comme en témoigne dans la *map*.

Trois points importants dans l'optimisation du protocole STP. Le premier point est lié à l'arrivée des acquittements sans aucune lacune (mémoires vides). Cet événement est généralement pris par le mécanisme pour indiquer que le probing est lancé prématurément en raison de la réorganisation des paquets ou d'un événement de fausse *timeout*. Dans ce cas le mécanisme est immédiatement terminé et restaurer la transmission de données. Le deuxième point est relié à l'arrivée des acquittements (STAT) qui ne correspondent pas à des probes existes dans la *map*. Ce scénario apparaît généralement dans l'une des deux situations : soit les acquittements ont pris un long temps pour arriver que leurs entrées de probes sont supprimées de la *map*, ou ces acquittements correspondent aux segments POLL qui avaient été envoyés avant le début du probing (leurs *timestamps* sont inférieures à *loss_ts_*). Bien que le mécanisme ne tienne pas en compte des acquittements dans le premier cas, il les enregistre dans la *map* dans le second. Au cours du premier RTT du probing le mécanisme réserve une entrée dans la *map* (généralement la première) pour ces types d'acquittement. La logique de considérer ces acquittements est qui sont assez proches dans le temps à l'erreur (envoyé durant le même RTT contenant l'erreur), que leur avis fournit des informations précieuses sur l'état du réseau à ce moment-là. Le mécanisme enregistre le *timestamps* et RTT mesuré dans l'entrée réservée dans la *map*. XSTP envoie un nombre de segments POLL chaque RTT, le mécanisme probing peut recevoir plusieurs acquittements *pre-loss*. Dans ce cas le mécanisme écrase le contenu de l'entrée réservée par les nouvelles données de l'acquittement arrivant. Fait intéressant, cette capacité de traiter les probes *pre-loss* peuvent permettre au cycle probing de terminer plus rapidement (dans environ un RTT), si le premier acquittement est arrivé au temps (deux échanges sont nécessaires). Le troisième et dernier point est lié aux segments USTAT. Bien qu'ils soient utilisés comme déclencheurs explicites du mécanisme probing, les segments sont totalement ignorés tandis que le mode probing est activé. Leur mémoires de lacune, souvent répétés par des segments STAT consécutifs, sont traitées uniquement après la terminaison du probing.

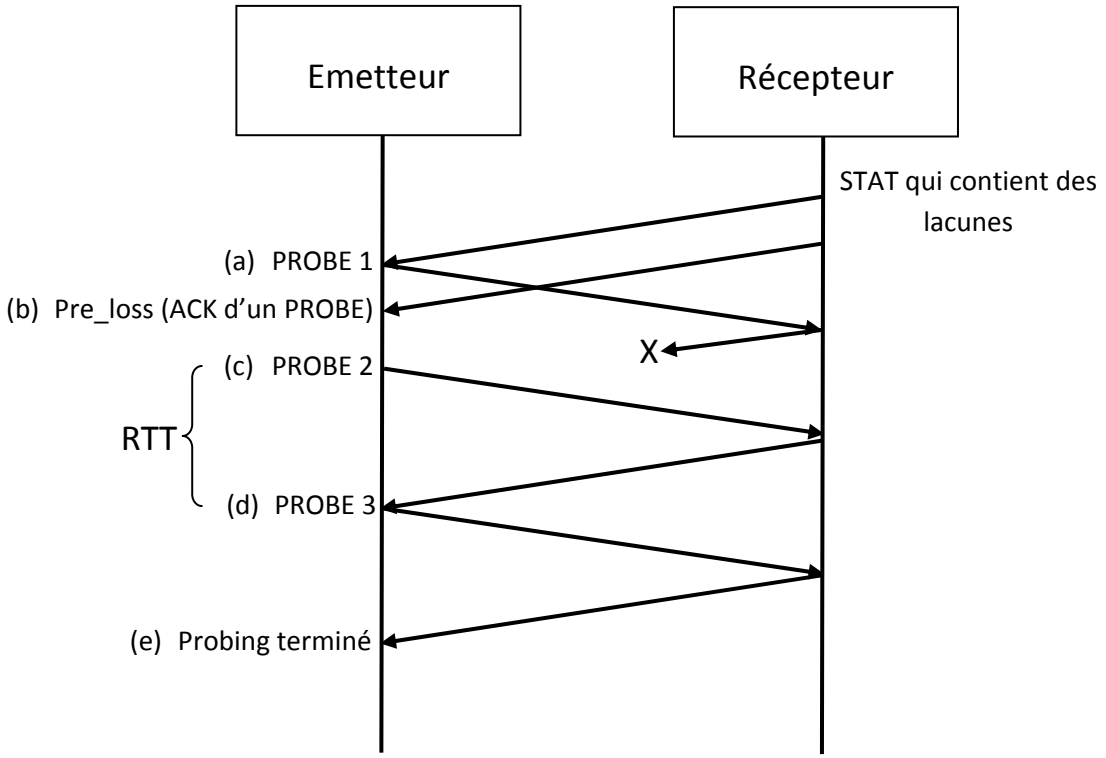

Figure IV.5. Phases d'un cycle probing comme cela se passe dans le réseau

Table de probing	
TS PROBE	RTT PROBE
1er TS	

(a) Début du probing

Table de probing	
TS PROBE	RTT PROBE
Pre-loss TS	**Pre-loss RTT**
1er TS	

(b) Après réception d'un Pre-loss

Table de probing	
TS PROBE	RTT PROBE
Pre-loss TS	Pre-loss RTT
1er TS	
2ème TS	

(c) Après un RTT

Table de probing	
TS PROBE	RTT PROBE
Pre-loss TS	Pre-loss RTT
1er TS	
2ème TS	**2ème RTT**
3ème TS	

(d) Après 2 RTTs

Table de probing	
TS PROBE	RTT PROBE
Pre-loss TS	Pre-loss RTT
1er TS	
2ème TS	2ème RTT
3ème TS	**3ème RTT**

(e) Après 3 RTTs

Figure IV.6. Phases d'un cycle probing comme indiqué dans la *map*

Une très importante partie dans l'optimisation du protocole STP (mécanisme probing) est la décision de prise de critères à la fin du cycle probing pour déterminer la cause probable de l'erreur. Bien que de nombreuses heuristiques peuvent être utilisées. L'heuristique utilisée dans cette optimisation du STP compare les deux RTT mesurés des probes avec *loss_rtt_*. Si les deux RTT mesurés sont inférieures ou égales au *loss_rtt_*, la congestion est non détectée et l'erreur est considéré liée au lien. Sinon, le contrôle de congestion est appliqué.

Comme mentionné précédemment, le mécanisme de contrôle de congestion invoque dès qu'il détecte la congestion. La mesure de contrôle de congestion dépend de combien de temps prend le cycle probing pour finir. Si le cycle est assez long et il a atteint un certain seuil pour une transmission inactive, l'émetteur va au *slow start*, sinon la fenêtre de congestion est fermée par l'algorithme *congestion avoidance* (la fenêtre est réduite de moitié). Après la terminaison du cycle probing, les segments mémoireés perdus par le dernier acquittement (STAT) sont retransmis et le taux normal de polling est restauré.

Conclusion

Les réseaux d'accès satellitaire LEO constituent un défi pour un transport efficace et fiable des données. La principale raison de l'échec des protocoles de transport standards faire face à la diversité des caractéristiques des liens hétérogènes, y compris les liaisons satellitaires LEO. Ces protocoles sont centrées sur la congestion et ils ont manqué d'une stratégie de contrôle d'erreur qui est adaptable aux différentes conditions d'erreur dans le réseau.

Après que nous ayons vu un aperçu général sur les protocoles de transport nano satellites dans ce chapitre, nous nous intéresserons au mécanisme probing du protocole XSTP. Ce dernier est le résultat de l'optimisation du protocole STP pour lui permettre de différencier entre les différents types d'erreurs qui

se produisent dans un réseau satellitaire LEO. La simulation et l'étude des performances du protocole XSTP sous NS2 feront l'objectif du prochain chapitre.

CHAPITRE V
SIMULATION ET ETUDE DES PERFORMANCES

Introduction

NS est un simulateur à événement discret orienté objet. Il est écrit en C++ avec une interface textuelle (ou shell) qui utilise le langage OTcl (*Object Tool Command Langage*). C'est le simulateur le plus célèbre dans le domaine de la simulation des réseaux.

Nous avons implémenté le protocole STP et son optimisation (mécanisme probing) sous NS2. Le mécanisme établit une stratégie de contrôle d'erreur pour le protocole. Dans un réseau d'accès satellitaire LEO, des erreurs peuvent varier en nature (liées à la congestion ou liées au lien). En utilisant la simulation, les performances du protocole XSTP sont étudiées.

Dans ce chapitre, nous allons présenter le simulateur des réseaux NS2, l'implémentation des variantes de TCP sous NS2, la configuration de la simulation, les paramètres des performances et nous finirons par la description des scénarios de nos tests.

V.1. Environnement de simulation

Beaucoup de simulateurs pour les réseaux informatiques ont été développés pour répondre aux attentes des utilisateurs. Parmi ces simulateurs, nous pouvons citer :

- OPNET : un simulateur à caractère commercial. Il fournit des outils optimisés pour créer et tester des modèles de réseaux. Il utilise un paradigme de modélisation hiérarchique dans chaque couche pour aider les utilisateurs à effectuer des simulations à évènement discrets de grande exactitude, et à récupérer les métriques voulues.
- GloMoSim : un simulateur basé sur un ensemble de modules et de bibliothèque chacune d'elles simule un aspect particulier des réseaux filaires ou sans fil sur une couche particulière du modèle OSI. GloMoSim est développé en langage PARSEC (*Parallel Simulation Environnement for Complex Systems)* qui permet la simulation parallèle des processus via les messages et les entités indépendantes. GloMoSim est extensible vers d'autres modèles via le langage PARSEC.
- QualNet : produit commercial dérivé de GloMoSim.
- INSANE : le simulateur INSANE est conçu dans le but de tester les algorithmes IP sur ATM avec un trafic réel dérivé d'un trafic empirique mesuré.
- NetSim : ce simulateur est conçu pour fournir des simulations détaillées d'ETHERNET incluant la détection des collisions.
- NS2 : simulateur à événements discrets, écrit en C++. Il est le simulateur le plus célèbre dans le domaine de la simulation des réseaux. Le projet de NS2 a débuté en 1989 avec le simulateur réseau REAL, et a connu plusieurs extensions via les contributions de la communauté scientifique. Il permet des simulations filaires et sans fil. C'est ce dernier simulateur que nous avons utilisé pour nos simulations.

V.1.1. Simulateur des réseaux NS2 (Network Simulator 2)

NS est un simulateur à événement discret orienté objet. Il est écrit en C++ avec une interface textuelle (ou shell) qui utilise le langage OTcl (Object Tool Command Langage). L'OTcl est une extension objet au langage de commande Tcl. Le langage C++ sert à décrire le fonctionnement interne des composants de la simulation. Pour reprendre la terminologie objet, il sert à définir les classes. Quant au langage OTcl, il fournit un moyen flexible et puissant de contrôle de la simulation comme le déclenchement d'événements, la configuration du réseau, la collecte de statistiques, etc. L'application NS se compose de deux éléments fonctionnels : un interpréteur et un moteur de simulation. Au moyen de l'interpréteur l'utilisateur est capable de créer le modèle de simulation ce qui revient à assembler les différents composants nécessaires à l'étude. Les composants du modèle de simulation sont appelés objets ou encore instances de classe. Le moteur de simulation effectue les calculs applicables au modèle préalablement construit par l'utilisateur via l'interpréteur [14].

NS bénéficie de toutes les possibilités qu'offrent les techniques objets comme l'héritage, le polymorphisme, la surcharge, etc. L'héritage permet d'élaborer des arborescences de classes. Le modèle de simulation est construit à partir d'une arborescence de classes qui en fait se dédouble :

- une définie en OTcl dite arborescence interprétée. Elle est utilisée par l'interpréteur et est visible par l'utilisateur;
- une définie en C++ que l'on nommera compilée. Elle forme l'arborescence utilisée par le moteur de simulation (que l'on appellera par la suite simulateur). C'est l'ombre de l'arborescence interprétée.

Les deux arborescences sont très proches l'une de l'autre. Du point de vue de l'utilisateur, il y a une correspondance univoque entre une classe d'une arborescence et une classe de l'autre arborescence. La Figure suivante montre la dualité des classes qui peut exister dans NS.

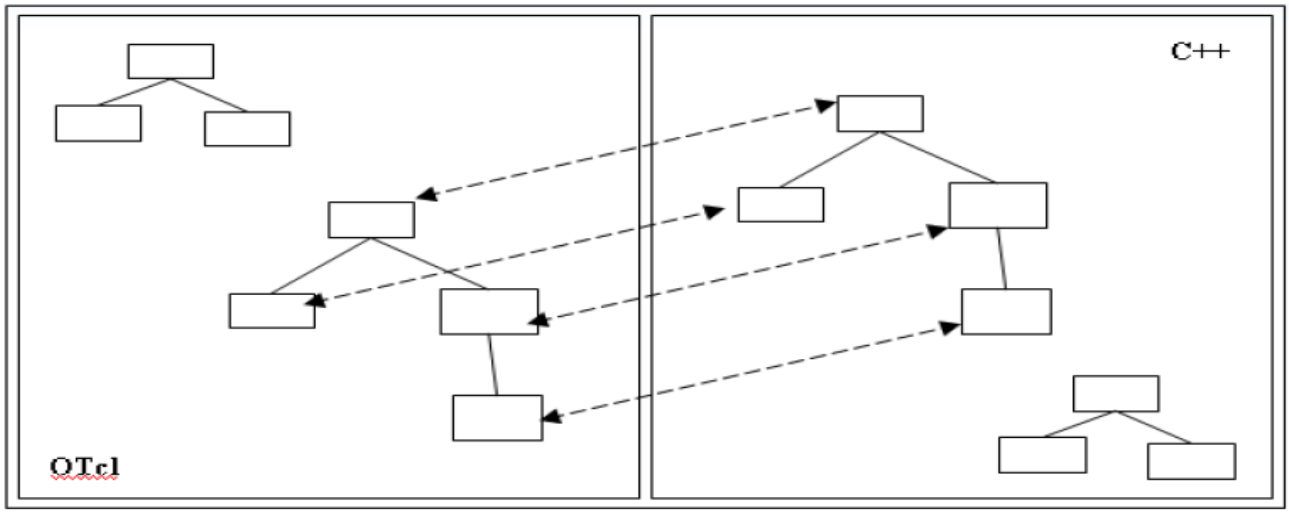

Figure V.1. Dualité des classes OTcl et C++

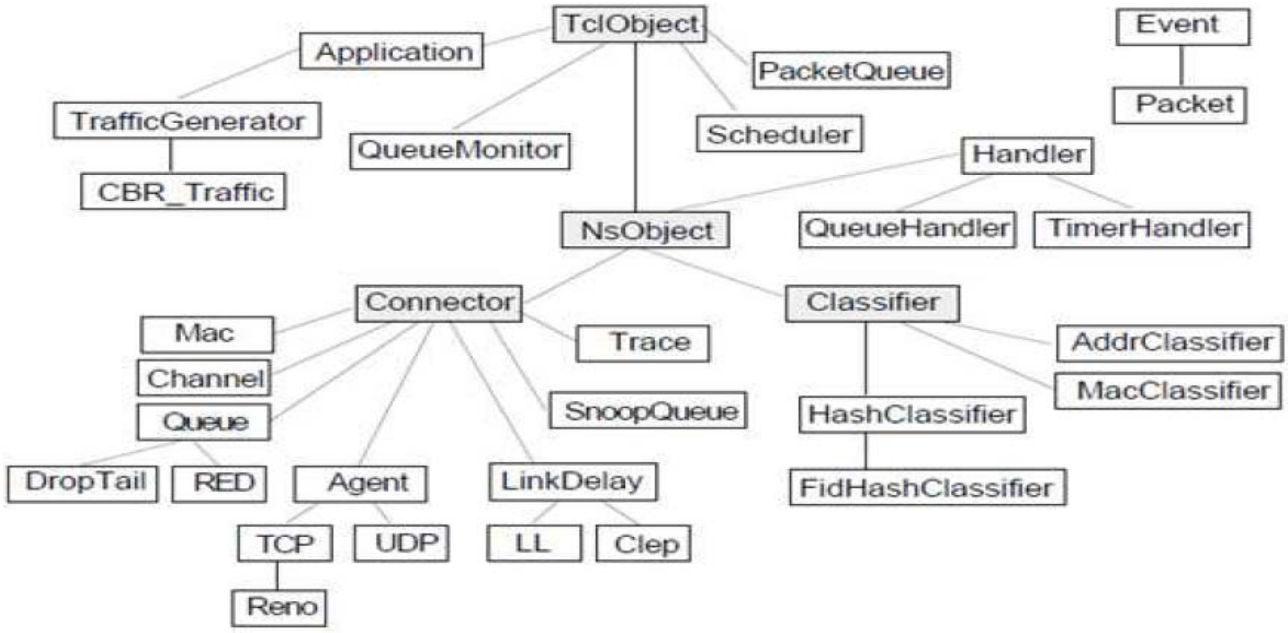

Figure V.2. Arborescence de dérivation des classes C++ du simulateur

Le principe général de la création des objets du modèle de simulation est le suivant : l'utilisateur crée un nouvel objet via l'interpréteur OTcl. Le nouvel objet interprété est cloné en un objet compilé correspondant dans le simulateur.

En réalité toutes les classes ne sont pas dans les deux arborescences de NS. Nous pouvons avoir des classes qui ne sont que dans l'interpréteur : elles servent à faire par exemple des assemblages (ou agrégations de classes) pour faciliter la manipulation. Nous pouvons avoir des classes qui sont purement dans le simulateur : elles ne sont pas visibles de l'utilisateur et servent au fonctionnement interne d'un composant comme par exemple certaines structures de données.

La Figure V.2 représente l'arborescence de classes utilisée par le simulateur. Les classes visibles au niveau de l'interpréteur comportent une déclaration dans la classe TclClass. Le nom des classes correspond à celui utilisé dans le code source [7].

V.1.2. Implémentation de protocoles de transport sous NS2

V.1.2.1. UDP

User Datagram Protocol (UDP, en français protocole de datagramme utilisateur) est l'un des principaux protocoles de télécommunication utilisés par Internet. Il fait partie de la couche de la pile de protocole TCP/IP : dans l'adaptation approximative de cette dernière au modèle OSI, il appartiendrait à la couche 4, comme TCP.

Le rôle de ce protocole est de permettre la transmission de données de manière très simple entre deux entités, chacune étant définie par une adresse IP et un numéro de port (pour différencier différents utilisateurs sur la même machine). Contrairement au protocole TCP, il travaille en mode non-connecté : il n'y a pas de moyen de vérifier si tous les datagrammes envoyés sont bien arrivés à destination et ni dans quel ordre (le séquençage peut cependant être assuré par un protocole réseau de couche inférieure). Il n'est prévu aucun contrôle de flux ni contrôle de congestion. C'est pour cela qu'il est souvent décrit comme étant un protocole non-fiable. En revanche, pour un paquet UDP donné, l'exactitude du contenu des données est assurée grâce à une somme de contrôle (*checksum*).

Les agents UDP sont implémentés en *udp. {cc, h}*. Un agent UDP accepte les données en taille variable à partir d'une application, et segmente les données si nécessaire. Les paquets UDP contiennent un numéro de séquence et un timestamp RTP. Bien que les paquets UDP réel ne contiennent pas de numéros de séquence ou de timestamps, ce numéro de séquence n'encourt aucun overhead simulé, et peut être utilisé pour l'analyse de fichier de trace ou pour simuler les applications basées sur UDP [7].

La taille maximale par défaut d'un segment (MSS) pour les agents UDP est 1000 octets :

Agent/UDP set packetSize_ 1000

Cette variable d'instance OTcl est liée à la variable *size_* de l'agent C++.

Les applications peuvent accéder aux agents UDP via la fonction *sendmsg()* en C++, ou par les méthodes *send* ou *sendmsg* en Otcl.

Ce qui suit est un simple exemple de la façon dont un agent UDP peut être utilisé dans un programme. Dans l'exemple, le générateur de trafic CBR est lancé au temps 1.0, temps auquel le générateur commence à appeler périodiquement la fonction *sendmsg()* de l'agent UDP.

```
set ns [new Simulator]
set n0 [$ns node]
set n1 [$ns node]
$ns duplex-link $n0 $n1 5Mb 2ms DropTail
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
$udp0 set packetSize_ 536
set null0 [new Agent/Null]
$ns attach-agent $n1 $null0
$ns connect $udp0 $null0
$ns at 1.0 "$cbr0 start"
```

Les commandes suivantes sont utilisées pour l'utilisation des agents UDP dans des scripts de simulation :

- *set udp0 [new Agent/UDP]* : Cela crée une instance de l'agent UDP.
- *$ns_ attach-agent <node> <agent>* : Il s'agit d'une commande utilisée pour joindre un agent *<agent>* à un nœud *<node>*.
- *$traffic-gen attach-agent <agent>* : Il s'agit d'une méthode de la classe Application/Traffic/<traffictype> qui relie le générateur de trafic à un agent donné *<agent>*. Par exemple, si nous voulons créer un trafic de flux CBR pour un agent udp, nous utilisons les commandes suivantes :
  - ➔ *set cbr1 [new Application/Traffic/CBR]*
  - ➔ *$cbr1 attach-agent $udp1*
- *$ns_ connect <src-agent> <dst-agent>* : Cette commande met en place une connexion de bout-en-bout entre deux agents (à la couche transport).

$udp set packetSize_ <pktsize>
$udp set dst_addr_ <address>
$udp set dst_port_ <portnum>
$udp set class_ <class-type>
$udp set ttl_ <time-to-live>
..... etc

Les différents paramètres ci-dessus sont des valeurs qui peuvent être fixés comme indiqué ci-dessus pour les agents udp. Les valeurs par défaut peuvent être trouvées dans *ns/tcl/lib/ns-default.tcl*.

V.1.2.2. Clones TCP

Transmission Control Protocol (protocole de contrôle de transmissions) abrégé TCP, est un protocole de transport fiable, en mode connecté.

Dans le modèle TCP/IP, TCP est situé au niveau de la couche transport (entre la couche de réseau et la couche application). Les applications transmettent des flux d'octets sur le réseau. TCP découpe le flux d'octets en *segments*, dont la taille dépend de la *Maximum Transmission Unit* MTU du réseau sous-jacent (couche liaison de données).

Une session TCP fonctionne en trois phases :

- l'établissement de la connexion;
- les transferts de données;
- la fin de la connexion.

Cette section décrit le fonctionnement des agents TCP dans *ns*. Il existe deux grands types des agents TCP : les agents *one-way* et les agents *two-way*. Les agents o*ne-way* sont subdivisés en un ensemble d'émetteurs TCP (qui obéissent à la congestion et les différentes techniques de contrôle d'erreur) et les récepteurs (*sinks*). Un agent *two-way* est symétrique dans le sens où il représente à la fois l'émetteur et le récepteur. Il est encore en développement [5, 7].

Les fichiers présentés dans cette section sont trop nombreuses à énumérer ici. Essentiellement, il couvre plusieurs fichiers correspondant à l'expression régulière *~ns/tcp*.{cc, h}*.

Les agents émetteurs TCP *one-way* supportés actuellement sont :

- Agent/TCP : un émetteur TCP tahoe;
- Agent/TCP/Reno : un émetteur TCP Reno;
- Agent/TCP/NewReno : Reno avec une modification;
- Agent/TCP/Sack1 : TCP avec sélective répéter;
- Agent/TCP/Vegas : TCP Vegas;
- Agent/TCP/Fack : TCP Reno avec *forward acknowledgment*.

Les agents récepteurs TCP *one-way* supportés actuellement sont :

- Agent/TCPSink : puits TCP avec un ACK par paquet;
- Agent/TCPSink/DelAck : puits TCP avec un délai configurable par ACK;
- Agent/TCPSink/Sack1 : puits selective ACK;
- Agent/TCPSink/Sack1/DelAck : Sack1 avec DelAck.

Les émetteurs expérimentales *two-way* ne sont actuellement disponible que dans une forme de TCP Reno:

- Agent/TCP/FullTcp.

Le simulateur prend en charge plusieurs versions d'un émetteur TCP abstrait. Ces objets, essayer de capturer la congestion de TCP et le contrôle d'erreur des comportements, mais ne sont pas destinés à être des répliques fidèles des implémentations du monde réel TCP. Ils ne contiennent pas de fenêtre dynamique, ils calculs le nombre de segment et le nombre d'ACK en unités de paquet, il n'y a pas de SYN/FIN pour l'établissement/démontage de la connexion, et aucune donnée qui est toujours transférée (par exemple, pas de somme de contrôle (*checksum*) ou de données urgentes) [7].

V.1.2.2.1. Tahoe

L'agent TCP Tahoe *Agent/TCP* effectue le contrôle de congestion et l'estimation du RTT d'une manière similaire à la version du protocole TCP publié avec le système 4.3BSD Tahoe UN'X de l'UC Berkeley. La fenêtre de congestion est augmentée par un paquet à chaque réception d'un nouvel ACK durant la phase *slow-start* (quand *cwnd_* < *ssthresh_*) et elle est augmentée par *1/cwnd_* pour chaque nouvelle réception d'un ACK durant la phase *congestion avoidance* (quand *cwnd_* ≥ *ssthresh_*).

- ***Réponses à la congestion :*** TCP Tahoe suppose qu'un paquet a été perdu (à cause de la congestion) quand il observe NUMDUPACKS (définie dans tcp.h, actuellement 3) de duplication d'ACKs, ou quand le temps de retransmission est expiré. Dans les deux cas, TCP Tahoe réagit par la mise de *ssthresh_* en moitié de la taille de la fenêtre (le minimum de *cwnd_* et *window_*) ou 2, selon le plus grand. Il initialise ensuite *cwnd_* par le retour à la valeur de *windowInit_*. Ce qui cause en générale l'entrer du protocole TCP à la phase *slow-start*.
- ***L'estimation de Round-Trip Time et la sélection de Timeout RTO :*** Quatre variables sont utilisées pour estimer le RTT et le temps de la retransmission : *rtt_*, *srtt_*, *rttvar_*, *tcpTick_* et *backoff_*. TCP initialise *rttvar_* par 3/tcpTick_ et *backoff_* par 1. Quand une horloge d'une future retransmission est réglée, son timeout est mis au *max(bt(a + 4v + 1), 64)* secondes, où *b* est la valeur currente du backoff_, *t* est la valeur du *tcpTick_*, *a* est la valeur de *srtt_* et *v* est la valeur de *rttvar_*.

  Les échantillons du RTT arrivent avec les nouveaux ACKs. L'échantillon du RTT est calculé comme la différence entre le temps actuel et le champ *time echo* dans le paquet ACK. Lorsque le premier échantillon est prélevé, sa valeur est utilisée comme valeur initiale de *srtt_*. La moitié du premier échantillon est utilisé comme valeur initiale de *rttvar_*. Pour les échantillons ultérieures, les valeurs sont mises à jour comme suit :
  - *srtt_ = 7/8*srtt_ + 1/8*échantillo;*
  - *rttvar_ = 3/4*rttvar_ + 1/4*|échantillo – srtt_|.*

L'exécution d'une simulation TCP nécessite la création et la configuration d'un agent, en le joignant une source de données du niveau application (un générateur de trafic), puis on lance l'agent et le générateur de trafic.

```
set ns [new Simulator]
set node1 [$ns node]
set node2 [$ns node]
set tcp1 [$ns create-connection TCP $node1 TCPSink $node2 42]
$tcp1 set window_ 50
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ns at 0.0 "$ftp1 start"
```

Cet exemple illustre l'utilisation du simulateur intégré en fonction *create-connection*. Les arguments de cette fonction sont : un agent source, un nœud source, un agent destination, un nœud destination, et l'ID de flux pour être utilisé dans la connexion. La fonction opère par la création de deux agents, mettre l'ID de flux dans les champs des agents, en attachant les agents source et destination à leurs nœuds et enfin connectant les agents. La valeur de retour de la fonction est le nom de l'agent source créé.

- ***Source de données TCP :*** un agent TCP ne génère pas de données d'application sur sa propre initiative; au lieu de cela, l'utilisateur de la simulation peut connecter n'importe quel module de génération de trafic à un agent TCP pour la production de données. Deux applications sont couramment utilisées pour le protocole TCP : FTP et Telnet. FTP représente en gros le transfert de données de grande taille, et telnet choisit la taille de son transfert aléatoirement à partir de *tcplib* (voir le fichier *tcplib-telnet.cc*).
- ***Autres paramètres de configuration :*** En plus du paramètre *window_*, l'agent TCP supporte d'autres variables de configuration. Chacune des variables décrites dans ce paragraphe est à la fois une variable de classe et une variable d'instance. Modification d'une variable de classe modifie la valeur par défaut pour tous les agents qui sont créés ultérieurement. Modification d'une variable d'instance d'un agent ne touche que les valeurs utilisées par cet agent. Par exemple :
  - ➔ *Agent/TCP set window_ 100* ;# Change la variable de la classe
  - ➔ *$tcp set window_ 2.0* ;# Change *window_* pour l'objet *$tcp* uniquement
- Les paramètres par défaut pour chaque agent TCP peuvent trouver dans le fichier : *ns/tcl/lib/ns-default.tcl.*

Pour de nombreuses simulations, quelques-uns des paramètres de configuration sont susceptibles de nécessiter des modifications. Les plus souvent modifiés sont : *window_* et *packetSize_*. Les modifications de ces paramètres peuvent avoir un effet profond sur le comportement de TCP.

V.1.2.2.2. Reno

Un agent TCP Reno est très similaire à l'agent TCP Tahoe, sauf qu'il inclut également *fast recovery*, quand la fenêtre de congestion est gonflée par un nombre des ACKs dupliqués qu'un émetteur TCP les a reçus avant de recevoir un nouvel ACK. Un nouvel ACK se réfère à toute ACK avec une valeur supérieure à la plus haute vue jusqu'à présent. En plus, un agent TCP Reno ne retourne pas au *slow-start* durant la retransmission rapide (*fast retransmit*). Dans le cas contraire, il réduit la fenêtre de congestion à la moitié de la fenêtre actuelle et réinitialise *ssthresh_* par cette valeur.

- *fast retransmit :* Si 3 ACK identiques sont reçus, TCP n'attend pas l'expiration du timer et retransmit les paquets.
- *fast recovery* : retransmission du 1er paquet non acquitté puis attente d'un ACK, si non réception, mode *slow-start*.

V.1.2.2.3. NewReno

TCP NewReno modifie le comportement de TCP Reno lorsqu'on reçoit un ACK partiel c'est-à-dire un ACK qui acquitte au moins le segment perdu (celui qui nous a fait entrer en *fast recovery*) mais pas tous les segments envoyés. Lors de la réception d'un tel ACK, TCP Reno quitte le mode de *fast recovery* ce qui pose un problème de performance lorsque plusieurs segments d'une même fenêtre sont perdus (les trames sont souvent perdues en rafale). TCP NewReno ne quitte le mode de *fast recovery* que si l'ACK reçu acquitte tous les segments envoyés. A la réception d'un ACK partiel, NewReno retransmet immédiatement le paquet suivant le dernier paquet acquitté dans cet ACK, diminue la taille de la fenêtre du nombre de paquets acquittés par cet ACK partiel et retransmet un paquet si c'est permis par la taille de *cwnd_*.

V.1.2.2.4. Vegas

Cet agent implémente TCP Vegas. Il a été contribué par Ted Kuo. TCP Vegas est un contrôle de congestion TCP, ou éviter la congestion du réseau (*network congestion avoidance*), l'algorithme qui met l'accent sur le délai des paquets, plutôt que la perte des paquets, comme un signal pour vous aider à déterminer le taux auquel envoyer les paquets.

TCP Vegas détecte la congestion très tôt basée sur l'augmentation des valeurs RTT des paquets dans la connexion, contrairement aux autres comme Reno, NewReno, etc. qui détectent la congestion seulement après la perte des paquets. L'algorithme dépend fortement de la précision de calcul de la valeur de base RTT. Si elle est trop petite, alors le débit de la connexion sera inférieur à la bande passante

disponible alors que si la valeur est trop grande, elle dépasse la connexion. Une mise en garde est intéressante lorsque Vegas fonctionne avec d'autres versions comme Reno. Dans ce cas, les performances de Vegas se dégradent puisque Vegas réduit ses taux d'envoi avant que Reno détecte la congestion et par conséquent donne une bande passante plus grande au flux TCP Reno co-existant.

Pendant la phase *congestion avoidance*, TCP Vegas utilise la différence entre le débit estimé D_e et le débit mesuré D_m par une source afin d'estimer l'état de congestion dans le réseau. Soit RTT_{base} le RTT minimum mesuré, D_e est calculé comme suit : $D_e = W/RTT_{base}$ où W est la valeur de la fenêtre au moment du calcul du débit estimé. Par ailleurs, TCP Vegas mesure le RTT effectif d'un paquet transmis (RTT_e) et calcule le débit mesuré comme suit: $D_m = B/RTT_e$ (où B représente le nombre d'octet transmis entre l'envoi du paquet considéré et la réception de l'ACK). Soit $Diff = D_e - D_m$ (grandeur positive par définition). Soient $0<\alpha<\beta$ deux seuils définis. Si $Diff < \alpha$, TCP Vegas augmente la fenêtre de congestion linéairement pendant le prochain RTT. Si $Diff > \beta$, TCP Vegas diminue la fenêtre linéairement pendant le prochain RTT. Sinon, la fenêtre n'est pas modifiée.

V.1.2.2.5. SACK

Une amélioration de TCP, nommée acquittement sélectif (*selective acknowledgement* ou SACK), autorise le destinataire TCP à acquitter des blocs de données reçus dans le désordre. Un champ SACK contient le nombre des blocs SACK est utilisé, chaque bloc SACK mémoires un ensemble non contigus de données qui sont été reçu. L'algorithme de contrôle de congestion implémenté dans TCP SACK est une extension conservatrice de contrôle de congestion de Reno, il utilise les mêmes algorithmes pour l'augmentation et la diminution de la fenêtre de congestion. La principale différence entre SACK et Reno est dans le comportement lorsque plusieurs paquets sont perdus dans une fenêtre de données.

Comme dans Reno, le SACK entre dans *fast recovery* lorsque l'émetteur reçoit des ACKs dupliqués. L'émetteur retransmet le paquet et réduit la fenêtre de congestion en moitié. Durant *fast recovery*, SACK maintient une variable appelée *pipe*, qui représente une estimation du nombre de paquets qui sont hors (*outstanding*) chemin. La variable *pipe* est incrémenté de un lorsque l'émetteur envoie un nouveau paquet ou retransmet un vieux paquet. Elle est décrémenté lorsque l'émetteur reçoit un ACK dupliqué avec une option SACK mémoires que les nouvelles données ont été reçues au niveau du récepteur. Quand un paquet retransmis est lui-même perdu, SACK détecte la perte par un *timeout* de retransmission, il retransmet le paquet, puis commence *slow-start*. L'émetteur sort de *fast recovery* quand il reçoit un ACK qui acquitte toutes les données remarquées lors de l'entrée au *fast recovery*. L'émetteur SACK a un traitement spécial pour les ACKs partiels. Pour ACK partiel, l'émetteur décrémente *pipe* par deux paquets au lieu d'un paquet.

V.2. Configuration de la simulation

L'implémentation du XSTP est faite dans NS2. Pour implémenter XSTP il faut implémenter tout d'abord le protocole STP puis lui ajouter l'optimisation proposée (mécanisme probing). Les deux protocoles STP et XSTP sont des protocoles de transport, donc pour les ajouter dans NS2, il faut localiser ses places dans la pile protocolaire. Le protocole XSTP c'est une classe qui est dérivée de la classe STP, et comme STP est un protocole de transport, ça sera donc une classe dérivée de la classe *Agent*.

L'avantage de l'implémentation du XSTP sous NS2 est l'existence des implémentations d'autres protocoles comme TCP ce qui nous permettrons de comparer XSTP avec d'autres protocoles. Notre objectif est d'étudier les performances du XSTP et cela en moyen de leur comparaison avec le protocole STP et les clones TCP (Tahoe, Reno, NewReno, Vegas, Sack).

Nous nous sommes intéressés à comparer les performances de STP et TCP en examinant les performances des connexions persistantes (c'est-à-dire, les transferts des fichiers de grande taille/dimension) sur des topologies de réseau satellitaire simulées.

La Figure V.3 illustre la topologie de simulation avec laquelle nous avons expérimenté. Pour tous les scénarios envisagés, nous avons considéré une configuration LEO. Les liens d'accès au satellite ont un délai de propagation de *5ms* et un débit de *2Mb/s*, et les liaisons intersatellitaires ont un délai de propagation de *10 ms* et un débit de *100Mb/s*. La topologie est similaire à la proposition du système Teledesic [20].

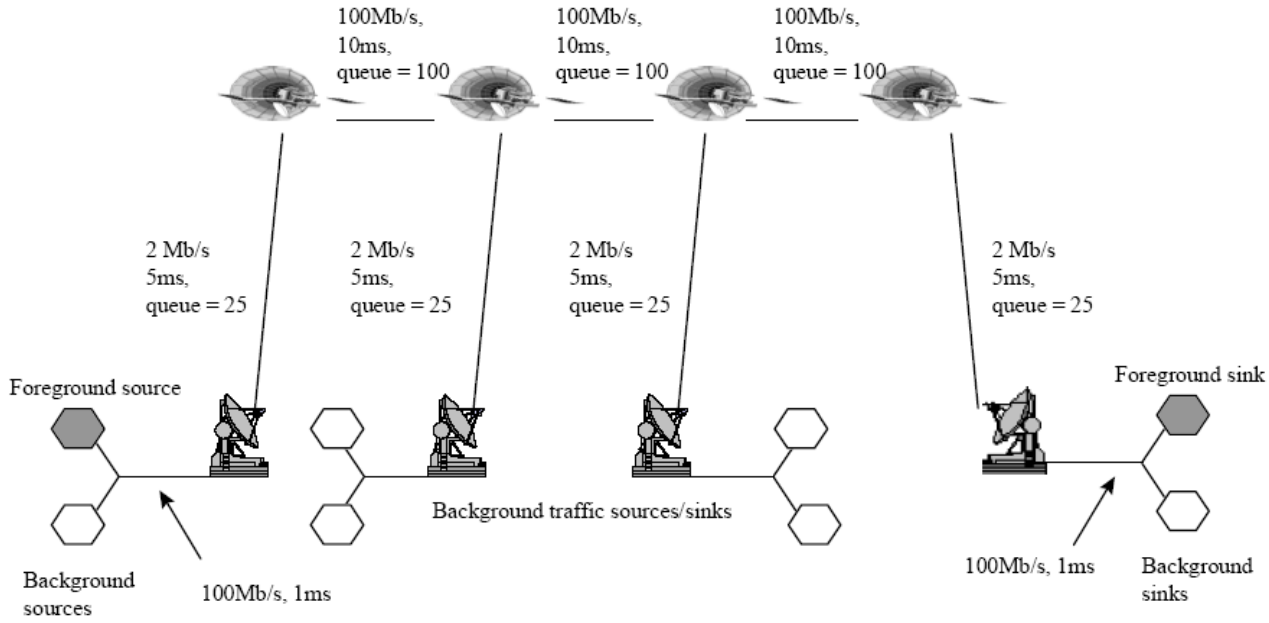

Figure V.3. Configuration d'une simulation de satellite en orbite basse (LEO)

Nous attachons à la source un agent XSTP et à la destination un agent STPSink. Un agent XSTP ne génère pas de données d'application; comme TCP; pour cela, nous avons lui connecté un générateur de trafic FTP pour que nous puisse envoyer des données de grande taille. Le nœud de destination est considéré comme un puits de données. La taille des paquets envoyés par la source est *1000 octets*, c'est qui est équivalent à la taille maximale d'un segment (MSS) d'une session XSTP. La taille de la fenêtre du récepteur est fixée à *200* et la taille initiale de la fenêtre de congestion de l'émetteur c'est *1*. Le taux de polling est fixé à *3 poll* chaque RTT, mais si le mécanisme du probing est déclenché alors le taux du polling revient *1 poll* chaque RTT. Le nombre maximum des probes suivis est fixé à *4 probes*, et le nombre des RTT mesurés consécutif pour terminer le cycle de probing est fixé à *deux mesures*. Le RTT initial est fixé à *0.08 secondes*.

Pour étudier les performances du XSTP, nous allons lui comparer avec STP et les variantes de TCP (clones TCP). Pour la configuration de la simulation de TCP nous avons utilisé NS par défaut pour tous les paramètres sauf la taille de la fenêtre, qui a été ouverte à une grande valeur pour éviter de limiter l'émetteur artificiellement.

V.3. Paramètres de performance

Un des paramètres de performance pertinents dans ce contexte est le débit effectif, qui est défini comme le taux moyen des données transmis (bits/sec) et est calculé selon la formule suivante:

Débit effectif = Taille originale / temps de simulation (bit/sec)

Un autre paramètre très important, en particulier pour les dispositifs alimentés par batterie comme les satellites sur orbite basse LEO est l'énergie dépensée durant la connexion. Bien qu'il existe beaucoup de facteurs à considérer pour l'évaluation efficace des dépenses d'énergie, l'accent est mis sur la transmission d'overhead. Ce paramètre est défini comme le pourcentage d'octets supplémentaires dépensés dans la transmission des données. La transmission d'overhead est calculée selon la formule suivante :

*Overhead = ((Taille totale – Taille originale) / Taille originale) * 100 (%)*

Un autre paramètre est l'efficacité, est le ratio d'utilisation du canal. Ce paramètre nous donne le ratio de données par mémoire au nombre total de données. Il est calculé par la formule suivante :

Efficacité = Taille originale / Taille totale

Le dernier paramètre détermine l'utilisation du canal inverse. Il montre l'efficacité du protocole sur les liens asymétriques où la bande passante n'est pas la même dans les deux sens. Ce paramètre est défini comme le nombre d'octets envoyés sur le canal inverse. Il est calculé par la formule suivante :

Canal inverse = Taille totale sur le sens inverse / Temps de simulation (bits/sec)

V.4. Scénarios de tests

Pour bien évaluer l'effet du mécanisme probing du protocole XSTP sur le débit effectif et l'énergie, des tests de simulation sont effectués sur la topologie de la Figure V.3. Nous avons ajouté des trafics HTTP en *background* qui ont occupé, en moyenne, environ 10% des liens *bottleneck* [20]. Ce trafic émule l'utilisation actuelle de WWW basée sur des distributions empiriques des traces de trafic actuelles. Le but était de ne pas encombrer le réseau fortement, mais d'ajouter une variabilité à la simulation.

Le trafic *background* a été source de multiples stations, créant un *bottleneck* occasionnel soit à la pénétration ou à la sortie du réseau satellitaire. Dans certains cas, nous avons équilibré la charge de trafic dans chaque sens par la création d'une connexion TCP-Sack dans le sens inverse avec la même quantité de trafic WWW, afin de provoquer des pertes périodique dans le chemin inverse aussi.

Nous avons ensuite examiné les performances de XSTP dans un environnement dans lequel le BER est varié de $1e^{-3}$ à $1e^{-8}$.

Conclusion

L'étude des performances du protocole XSTP a nécessite son implémentation dans le simulateur de réseaux NS2. Après cette implémentation, nous avons fait des simulations sur la topologie LEO illustrée par la Figure V.3. En fin, avec les paramètres des performances que nous sommes définis dans la Section V.3 et les scénarios de tests définis dans la Section précédente nous avons exécuté nos simulations et comparé le protocole XSTP avec les variantes de TCP (clones TCP). Dans le chapitre suivant nous allons illustrer les résultats de la simulation et déduire le protocole le plus performant.

CHAPITRE VI
RESULTATS DE LA SIMULATION

Introduction

Dans ce chapitre, les résultats des tests de simulation sont présentés et expliqués. Le chapitre présente également une analyse des résultats et les observations intéressantes. Nous allons comparer le protocole XSTP avec les variantes de TCP (clones TCP), pour déterminer quel est le meilleur protocole. Nous allons regarder les paramètres des performances définies dans le chapitre précédent et qui sont : le débit effectif, la transmission sur le canal inverse, l'efficacité et le pourcentage d'overhead transmis par mémoire au total de données.

XSTP et STP sont implémentés sous NS2 puisque ce simulateur est déjà largement utilisé par la communauté de recherche en réseau, et est particulièrement bien supporté pour les simulations de TCP. Nous nous sommes intéressés en comparant les performances de XSTP et clones TCP en examinant les performances des connexions persistantes (le transfert des fichiers long) sur une topologie de réseau satellite LEO simulé. Dans ce qui suit, nous présentons les résultats d'une comparaison entre XSTP et clones TCP dans un environnement où le BER est élevé.

VI.1. Transmission dans un seul sens

Dans ce cas le transfert de données se fait d'un émetteur à un récepteur dans un seul sens (l'émetteur ne fait que de la transmission et le récepteur ne fait que de la réception). Un trafic HTTP est ajouté dans la simulation, ce trafic émule l'utilisation actuelle de WWW. La durée de toutes les simulations est 60 secondes. Dans chaque cas les tests sont répétés avec différent taux d'erreur (BER varie entre 10^{-8} et 10^{-3}).

VI.1.1. Débit effectif

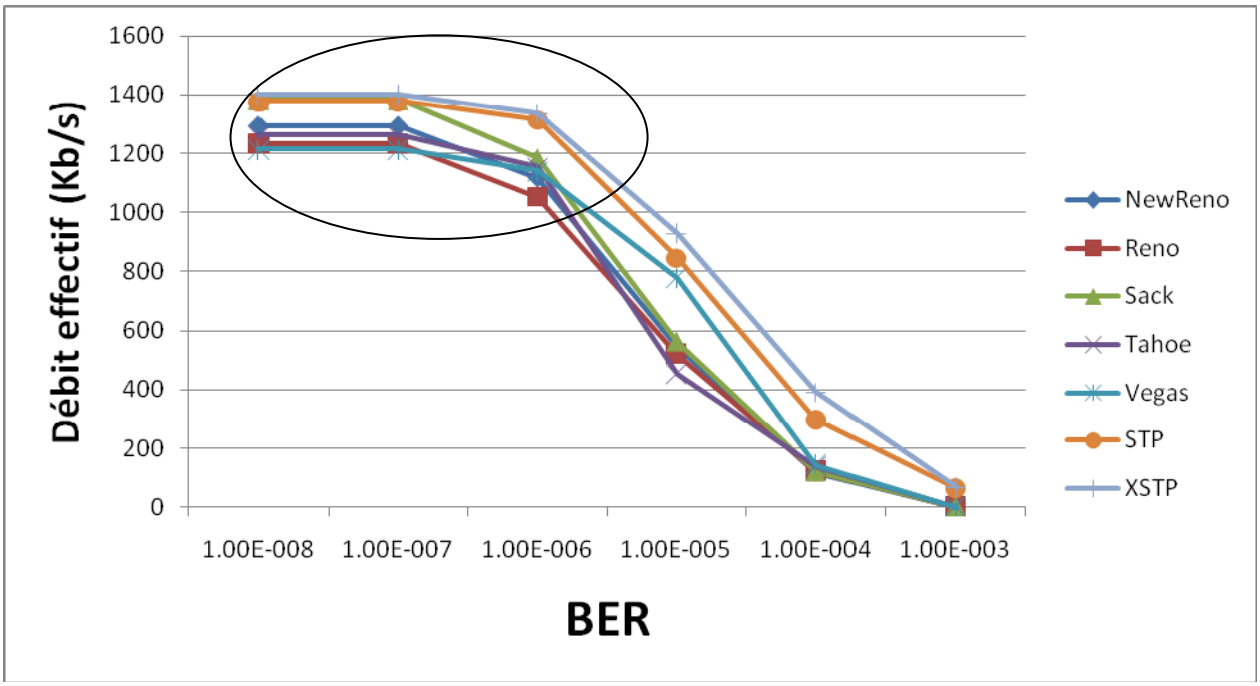

Figure VI.1. Débit effectif en fonction du BER dans le cas de transmission dans un seul sens

La Figure VI.1 illustre la variance de débit effectif atteint en fonction du BER. Le débit diminue avec l'augmentation de probabilité de perte pour tous les protocoles. Comme montre la Figure, XSTP est plus performant que STP et les variantes de TCP en terme de débit atteint. Cela est due en grande partie à la capacité du mécanisme probing de procéder à une récupération immédiate (*Immediate Recovery*) lorsque des erreurs de lien sont détectées, et donc d'éviter de réduire inutilement le taux de transmission; contrairement aux STP et clones TCP qui réduisent leur taux de transmission à chaque détection d'erreur.

Le cycle probing prend plus de temps pour terminer avec l'augmentation de BER, en raison des pertes des probes ou leurs ACKs.

Une autre observation est que le meilleur protocole des variantes de TCP est le protocole TCP Sack en termes de débit atteint. Celui-ci est proche du protocole STP dans le cas où le BER est faible, mais avec l'augmentation du BER (à partir de 10^{-6}) la différence entre STP et TCP Sack est très grande dans le débit. Il y a aussi le protocole TCP Vegas, qui donne un débit le plus faible dans le cas où le BER est faible, et il donne un débit plus grand que toutes les variantes TCP même TCP Sack quand le BER est élevé. Cela est due à la détection de la congestion très tôt par TCP Vegas basée sur l'augmentation des valeurs RTT des paquets dans la connexion, contrairement aux autres comme Reno, NewReno, etc. qui détectent la congestion seulement après la perte des paquets. En d'autre terme, si le BER est faible et avec la variance de RTT, TCP Vegas peut réduire sa fenêtre à cause de la variance des RTT alors que les autres variantes de TCP ne réduit pas leurs fenêtres s'il n'y a pas une perte; et dans le cas où le BER est très élevé, TCP Vegas atteint un débit intéressant par mémoire aux autres à cause de la détection de la perte très tôt.

La Figure VI.2 c'est un zoom de la partie marquée sur la Figure originale VI.1.

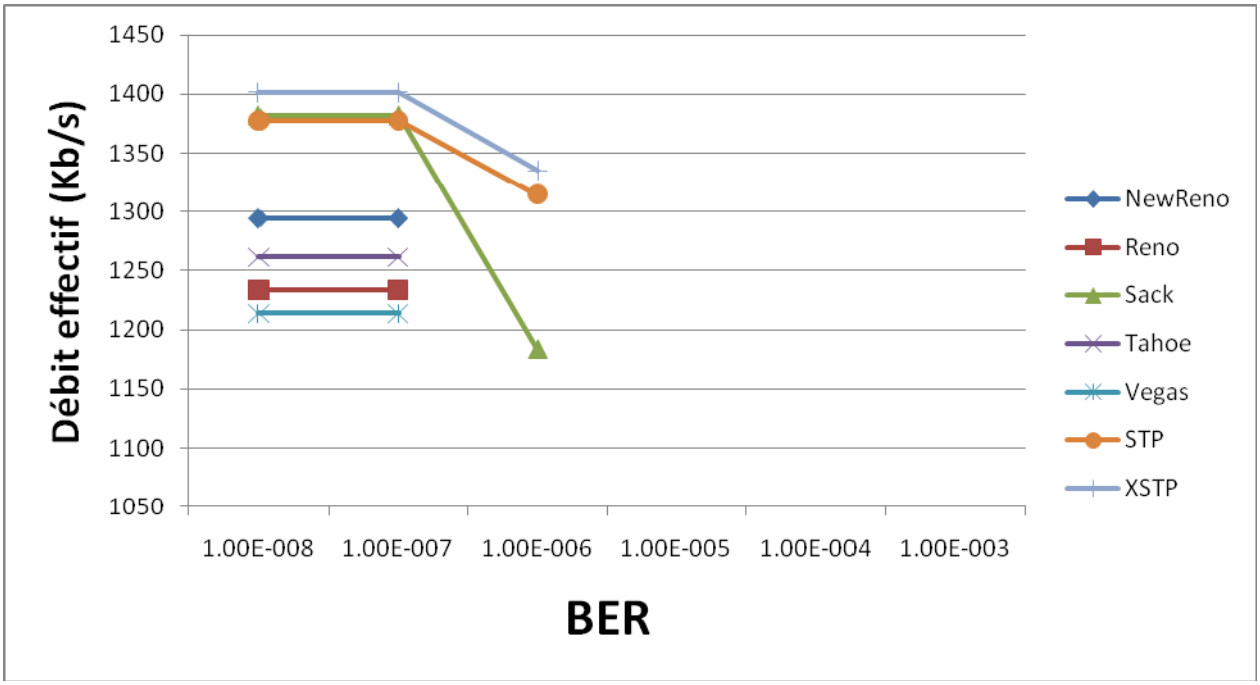

Figure VI.2. Zoom de la partie marquée sur la Figure VI.1

VI.1.2. Bande passante nécessaire pour le canal inverse

Dans ce cas de test on peut déterminer la bande passante nécessaire pour le canal inverse. Il faut noter que la bande passante dans le canal inverse doit être minimisé le plus possible, puisque les liaisons satellitaires sont asymétriques (la bande passante n'est pas la même dans les deux sens). Comme nous la montre la Figure VI.3, le protocole XSTP nécessite une bande passante très petite sur le canal inverse par mémoire aux STP et clones TCP.

Une autre observation est que la bande passante sur le canal inverse augmente avec l'augmentation du BER dans le cas des protocoles XSTP et STP, par contre elle diminue avec l'augmentation du BER pour les clones TCP. Le canal inverse est utilisé pour la transmission des ACK, et la bande passante sur le canal inverse varie selon le type des ACK, leur taille et leur nombre. Dans le cas des protocoles clones TCP, le récepteur envoie un ou plusieurs ACK à la réception des paquets, donc s'il n'y a pas de perte, le récepteur envoie beaucoup d'ACK puisqu'il reçoit beaucoup de paquets ce qui augmente la bande passante sur le canal inverse, mais si la probabilité de perte est très élevée, le récepteur

ne reçoit pas beaucoup de paquets, donc il n'envoie pas beaucoup d'ACK ce qui diminue la bande passante sur le canal inverse. Par contre, dans le cas des protocoles XSTP et STP, le récepteur envoie des paquets STAT et USTAT sur le canal inverse. Dans ce cas ce qui fait l'augmentation de la bande passante sur le canal inverse est le nombre des USTAT envoyés et la taille des STAT. Dans ce cas si la probabilité de perte est faible, alors le récepteur n'envoie pas beaucoup de USTAT et envoie des STAT avec des tailles petites (ne contiennent pas des lacunes), ce qui nécessite une bande passante petite; mais si la probabilité de perte est très élevée, le récepteur envoie beaucoup de USTAT et des STAT qui ont des tailles intéressantes (contiennent des lacunes), ce qui augmente la bande passante nécessaire sur le canal inverse.

La dernière observation est la différence entre XSTP (STP optimisé) et STP classique, quelle est causée par le nombre des STAT et USTAT envoyer par chaque protocole. XSTP nécessite moins de bande passante sur le canal inverse que le protocole STP, puisque durant le cycle probing dans XSTP, le nombre des STAT envoyés diminue à cause de la diminution de taux de polling à un POLL chaque RTT durant ce cycle (le nombre des STAT reçu égale au nombre des POLL envoyé, donc durant le probing le récepteur envoie un STAT chaque RTT); par contre, pour STP le taux de polling est toujours égale à 3, donc durant la période du probing il y a 2 STAT en plus envoyé par STP par mémoire au XSTP. Aussi, durant le cycle probing XSTP n'envoie pas des USTAT puisque il arrête la transmission de données, et comme nous savons que le USTAT est envoyé si le récepteur reçoit 2 paquets de données qui ont 2 numéros de séquences non consécutif; par contre, STP n'arrête jamais la transmission de données, donc toujours il y a la possibilité d'envoyer des USTAT, surtout dans le cas où la probabilité de perte est élevé.

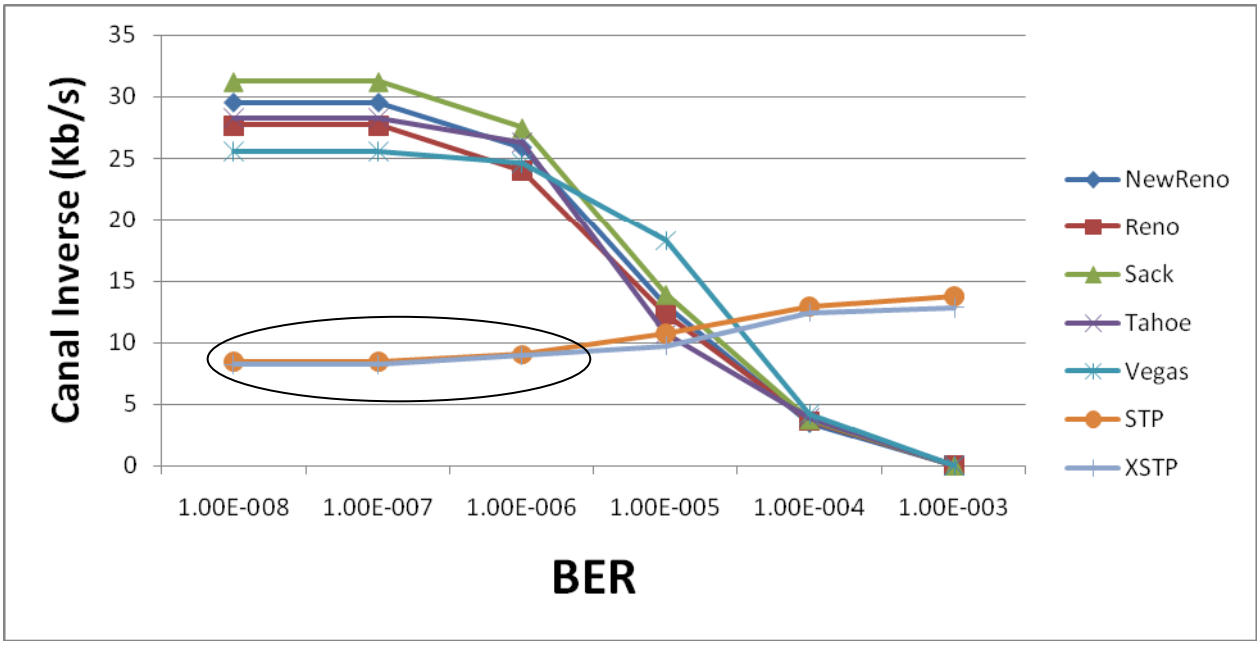

Figure VI.3. Bande passante du canal inverse en fonction du BER dans le cas de transmission dans un seul sens

La Figure VI.4 c'est un zoom de la partie marquée sur la Figure originale VI.3.

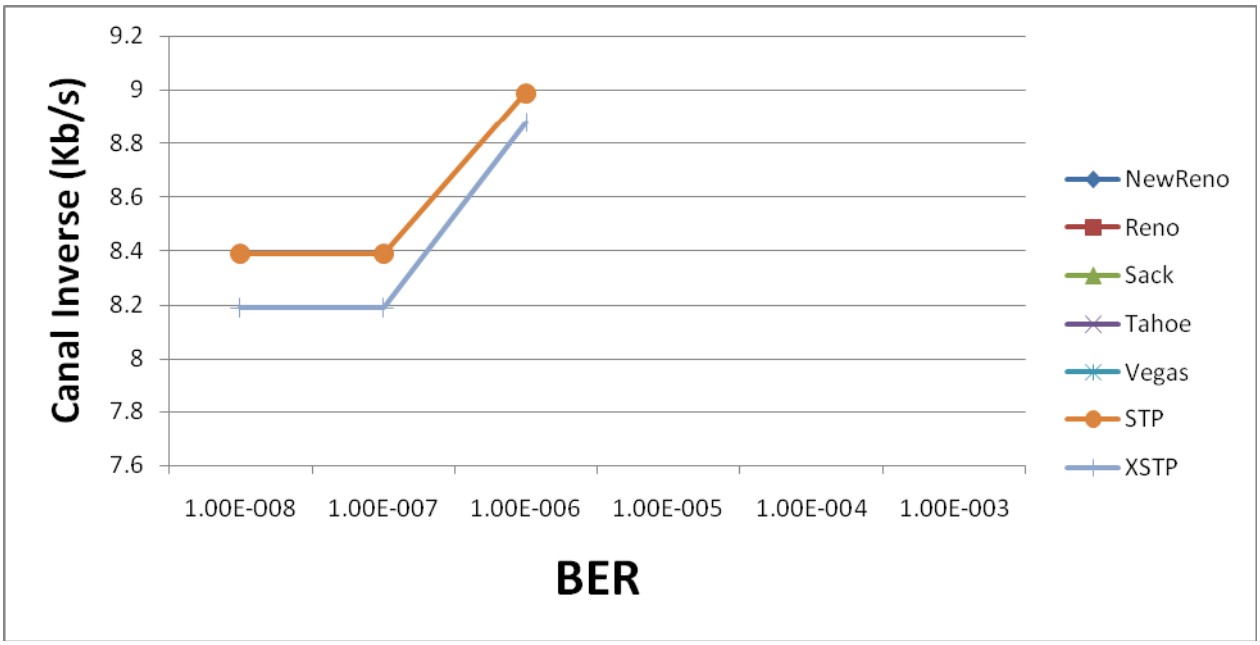

Figure VI.4. Zoom de la partie marquée sur la Figure VI.3

VI.1.3. Efficacité

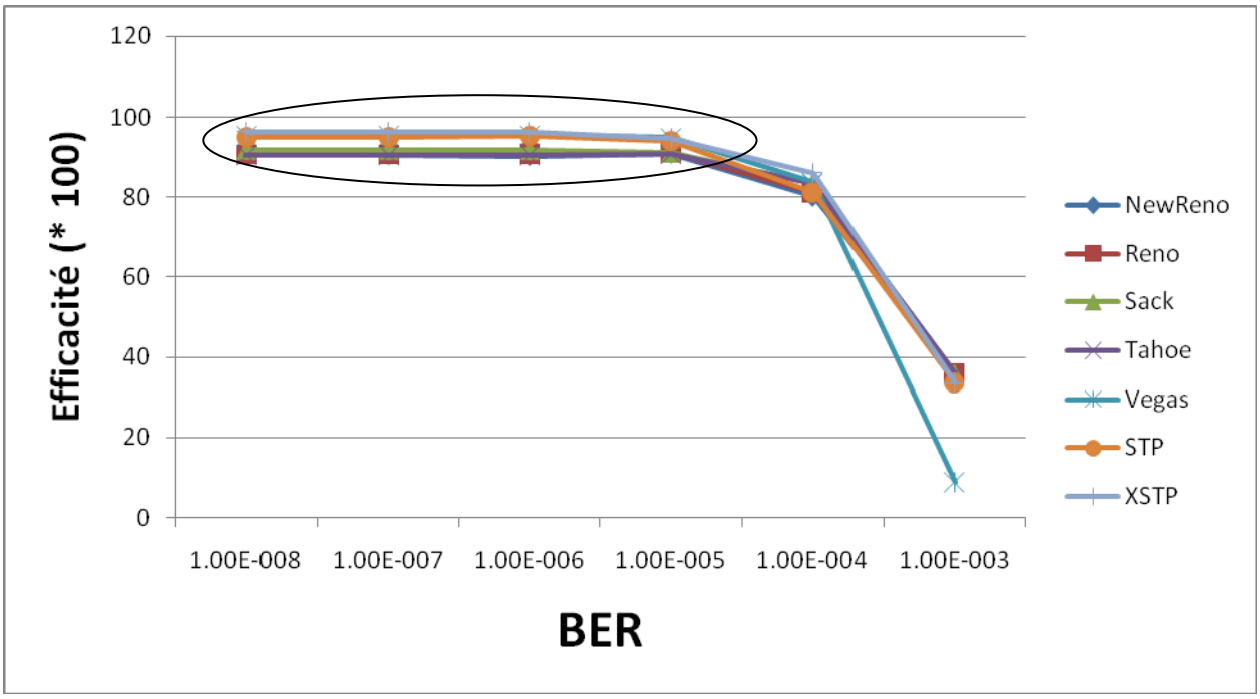

Figure VI.5. Efficacité de la transmission de données en fonction du BER dans le cas de transmission dans un seul sens

Dans ce test, nous avons calculé l'efficacité du protocole qu'elle représente le ratio entre les données de l'utilisateur et le total des données envoyées (la valeur de l'efficacité est inférieur à 1). Cette valeur diminue avec l'augmentation de BER (comme le débit effectif), puisque le nombre de données utilisateur diminue avec l'augmentation de la probabilité de perte. La Figure VI.5 illustre que le protocole XSTP est le plus efficace que STP et les clones TCP, ce qui nous montre que le protocole XSTP ne perte pas beaucoup de données, donc ça montre l'efficacité de l'optimisation intégrée au protocole STP. Une

autre chose, si la probabilité de perte est élevée, l'efficacité de tous les protocoles diminue d'une façon importante (dans le cas où le BER égale 10^{-3}).

La Figure VI.5 montre aussi que le protocole TCP Vegas est plus efficace que STP (voir zoom), ce qui signifie que TCP Vegas à moins de perte par mémoire à STP, mais le problème de TCP Vegas est qu'il n'envoie pas beaucoup de données (son débit effectif est très faible qui est *1213.7 Kb/s* par mémoire à STP qui est *1377.79 Kb/s*, voir Figure VI.1), donc il n'exploite pas bien la bande passante offerte. Mais, si la probabilité de perte est très élevée, l'efficacité de TCP Vegas est la plus faible par mémoire aux autres protocoles.

On peut dire que cette efficacité détermine l'efficacité d'utilisation du canal, puisque si le nombre de données utilisateur est important, l'efficacité est très proche de *1*, donc le canal est bien utilisé, sinon elle est proche de *0*, ce que signifie que le canal n'est pas bien exploité.

La Figure VI.6 c'est un zoom de la partie marquée sur la Figure originale VI.5.

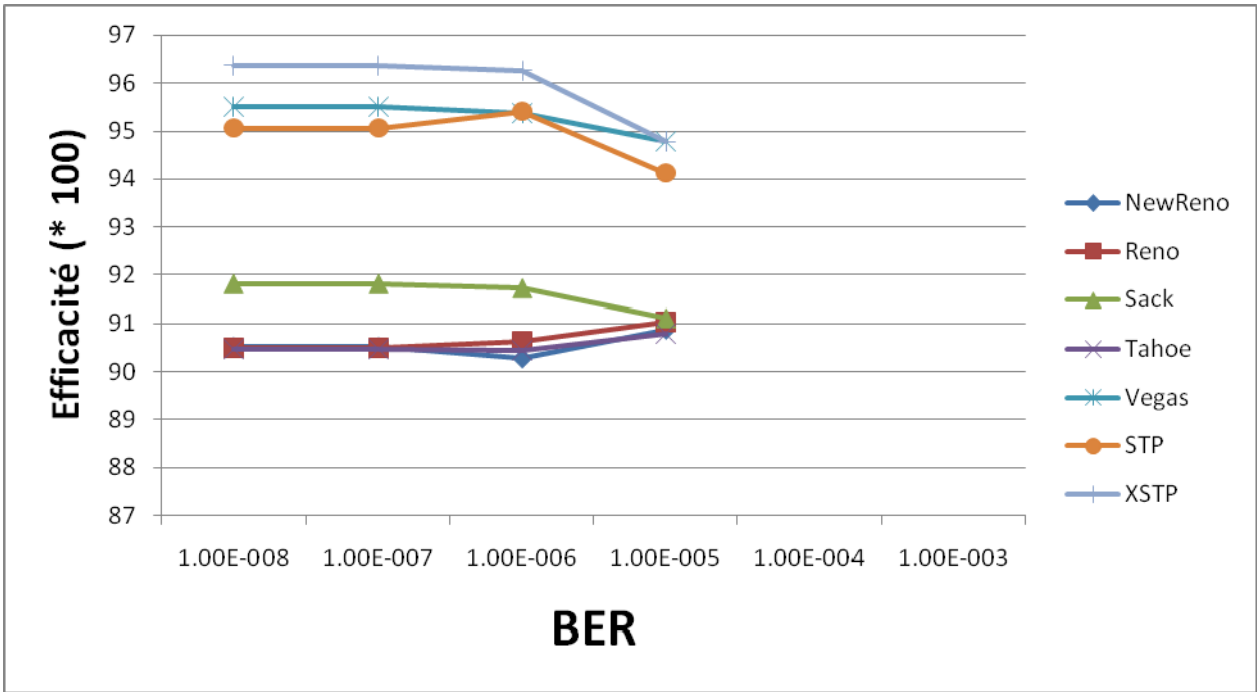

Figure VI.6. Zoom de la partie marquée sur la Figure VI.5

VI.1.4. Overhead

L'overhead détermine les données supplémentaires (qui n'appartiennent pas aux données utilisateur) envoyées. Nous avons représenté l'overhead sous forme de pourcentage. Ce test représente le complément de l'efficacité, puisque c'est le nombre d'octets supplémentaires envoyés sur le nombre total d'octets envoyés multiplier par *100*. A l'inverse de l'efficacité, l'overhead augmente avec l'augmentation de BER. La Figure VI.7 illustre que XSTP n'envoie pas beaucoup de données supplémentaires par mémoire aux autres protocoles, ce qui montre l'efficacité de l'optimisation du protocole STP (mécanisme probing). Durant le cycle probing, un émetteur XSTP envoie un seul POLL chaque RTT et arrête la transmission de données pour protéger la perte de nouvelles données. Aussi, à la fin de cycle probing, s'il n'y a pas de congestion, l'émetteur ne réduit pas sa fenêtre de congestion, ce qui donne la possibilité d'envoyer plus de données utilisateur; donc la réduction d'overhead est atteinte par la suspension de la transmission durant le cycle probing et d'avoir une fenêtre de congestion grande s'il n'y a pas de congestion, ce qui diminue le nombre de données utilisateur perdues.

Dans les variantes de TCP, TCP Vegas qui envoie moins d'overhead si la probabilité de perte n'est pas élevée, mais si la probabilité est très élevée, c'est celui-ci qui envoie plus d'overhead que les autres protocoles. Ce qui signifie que la stratégie de contrôle de congestion de TCP Vegas est bien adaptée dans le cas où la probabilité d'erreur n'est pas trop élevée.

L'un des choses les plus importantes dans le domaine des satellites est la consommation de l'énergie. Les chercheurs essaient toujours de minimiser l'énergie dépensée par les satellites dans la transmission de données. L'un des paramètres qui influe sur cette énergie est le nombre d'overhead transmis, l'énergie dépensée dans la transmission d'overhead considérer comme une énergie perdue. La Figure VI.7 montre que le protocole XSTP consomme moins d'énergie que les autres protocoles dans la transmission d'overhead, donc il bénéfice d'une énergie supplémentaire qui l'utilise dans la transmission de données.

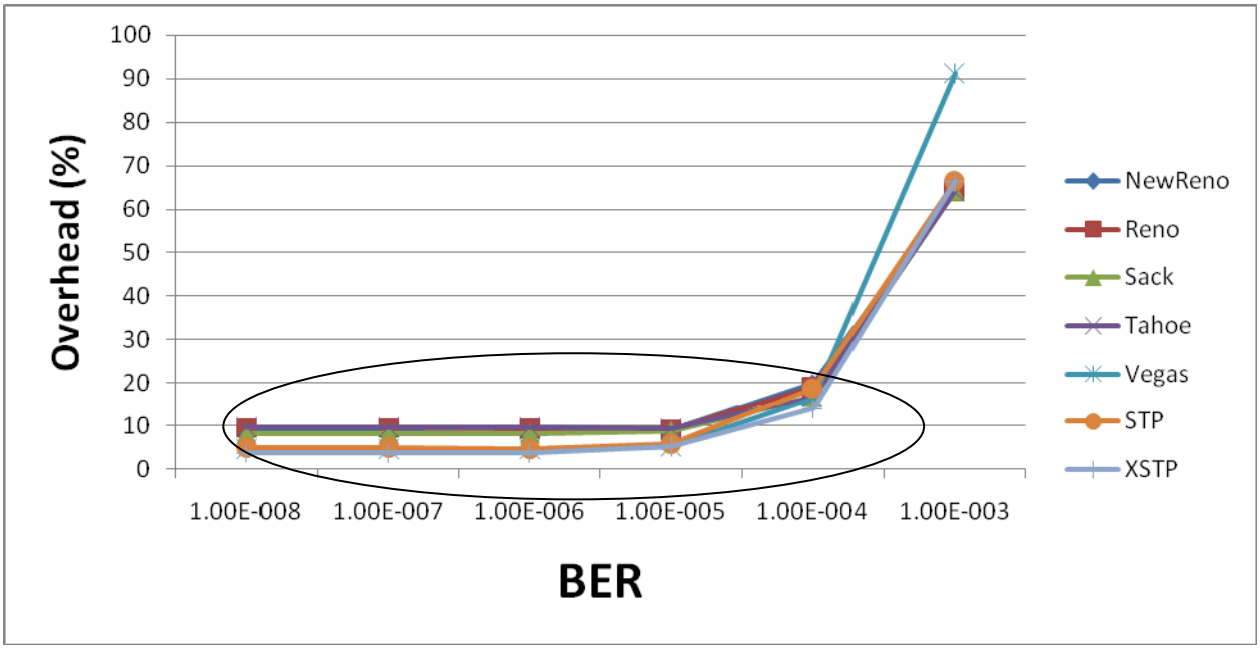

Figure VI.7. Overhead transmis en fonction du BER dans le cas de transmission dans un seul sens

La Figure VI.8 c'est un zoom de la partie marquée sur la Figure originale VI.7.

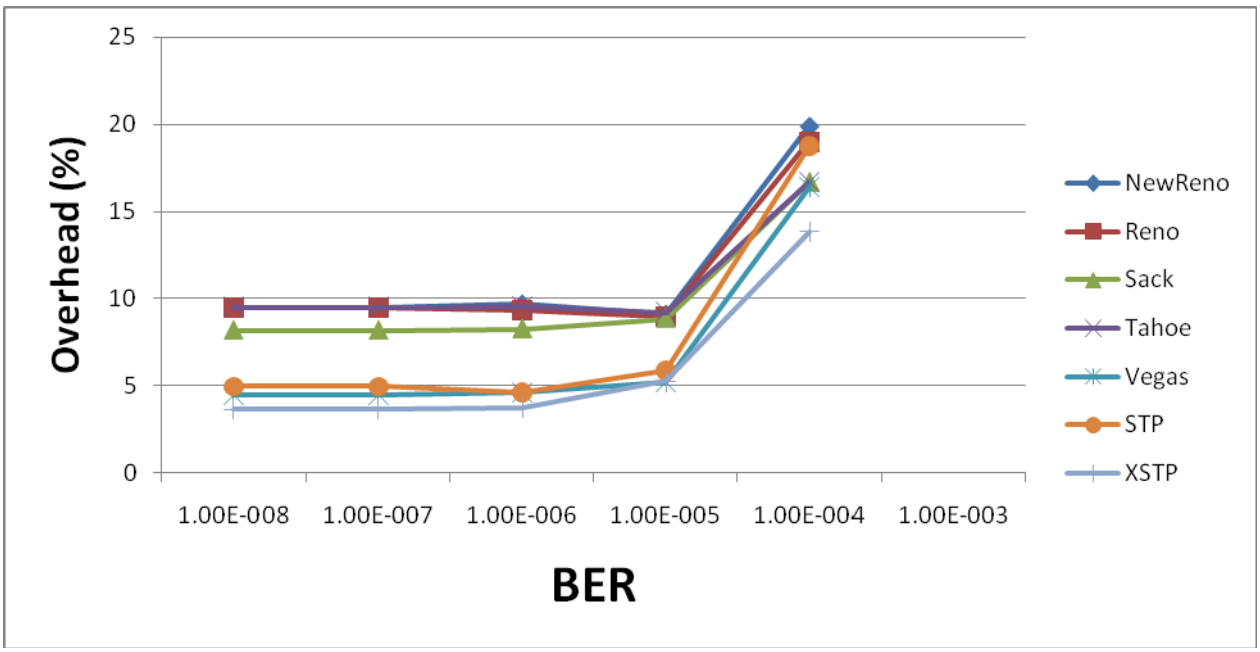

Figure VI.8. Zoom de la partie marquée sur la Figure VI.7

VI.2. Transmission dans les deux sens

Dans ce cas, le transfert de données se fait dans les deux sens (un nœud est un émetteur et récepteur en même temps). Un trafic HTTP est ajouté dans la simulation; ce trafic émule l'utilisation actuelle de WWW. Dans le sens inverse, le protocole TCP Sack est accordé à l'émetteur qui réceptionne les données transmises par un émetteur qui utilise le protocole XSTP ou l'un des variantes de TCP (clones TCP). La durée de toutes les simulations est 60 secondes. Dans chaque cas, les tests sont répétés avec différent taux d'erreur (BER varie entre 10^{-8} et 10^{-3}).

VI.2.1. Débit effectif

Dans ce test, le débit diminue avec l'augmentation du BER et XSTP reste toujours le meilleur en termes de débit atteint comme dans le test d'un seul sens. Le débit de transmission atteint par chaque protocole est diminué par mémoire au cas de la transmission dans un seul sens, cela est dû à la transmission dans le sens inverse. La Figure VI.9 illustre quelques différences par mémoire à la transmission dans un seul sens, et qui sont les suivantes :

- le débit effectif de XSTP ne diminue pas comme celui des autres protocoles, ce qui montre l'efficacité du mécanisme probing (l'optimisation du protocole STP).
- STP est meilleur que TCP Sack, si le BER est faible, puisque STP n'utilise pas un *timeout* pour l'arriver des ACK.
- TCP Vegas est influé beaucoup par la transmission dans le sens inverse, il donne un débit très faible par mémoire aux autres protocoles même dans le cas où la probabilité de perte est très faible. Cela signifie que TCP Vegas est moins performant s'il fonctionne avec d'autre protocoles, à cause de la réduction de ces taux d'envoie avant la perte ce qui donne plus de bande passante au protocole de l'autre côté (TCP Sack dans notre cas).

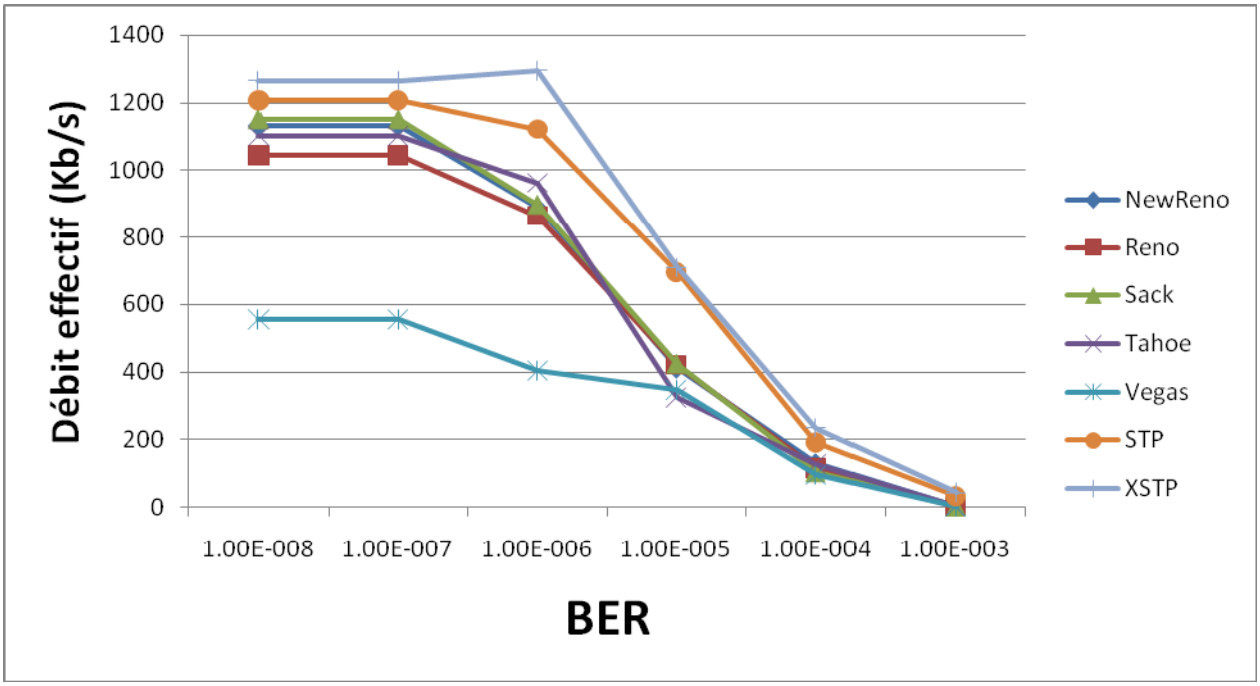

Figure VI.9. Débit effectif en fonction du BER dans le cas de transmission dans 2 sens

VI.2.2. Bande passante nécessaire pour le canal inverse

Dans ce cas, c'est les mêmes observations que dans le cas de la transmission dans un seul sens, la bande passante nécessaire sur le canal inverse est plus faible pour les protocoles STP et son optimisation par mémoire aux variantes de TCP qui nécessitent une grande bande passante dans le cas où le BER est faible. Cela à cause de nombre des ACK envoyés par les variantes de TCP qui est très important. La seule différence dans ce cas par mémoire au cas de transmission dans un seul sens est : TCP Vegas ne nécessite pas une grande bande passante sur le canal inverse, cela est due au débit très faible de celui-ci, comme il est expliqué dans le paragraphe précédent.

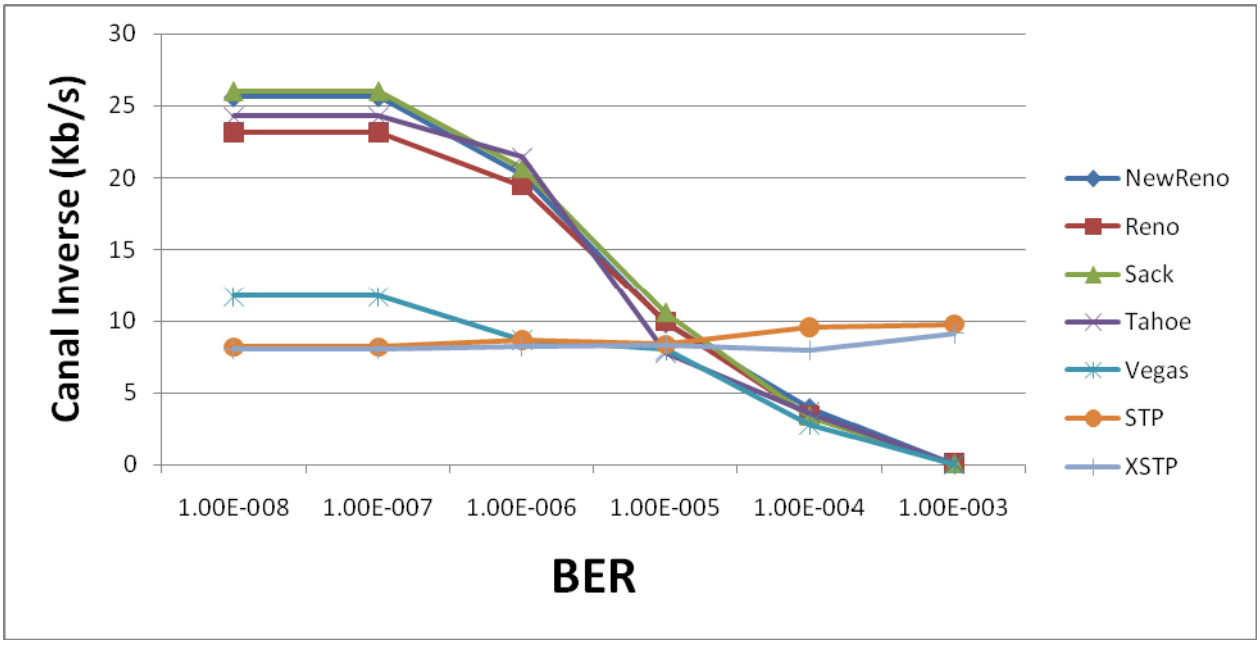

Figure VI.10. Bande passante du canal inverse en fonction du BER dans le cas de transmission dans 2 sens

VI.2.3. Efficacité

Dans la transmission dans les deux sens, XSTP (STP optimisé) est toujours le plus efficace par mémoire aux autres, comme dans le cas de la transmission dans un seul sens. La seule différence est que cette efficacité ne diminue pas d'une façon très importante, comme dans la transmission dans un seul sens si la probabilité de perte est très élevée. Cela est due à la diminution des taux d'envoie par les protocoles, s'il y a d'autres transmissions dans le sens inverse.

L'efficacité du XSTP dans le cas de la transmission dans un seul sens, pour un BER 10^{-8} était *96.37*; par contre, dans le cas de la transmission dans les deux sens, pour le même BER, l'efficacité du XSTP touche une valeur de *95.79*.

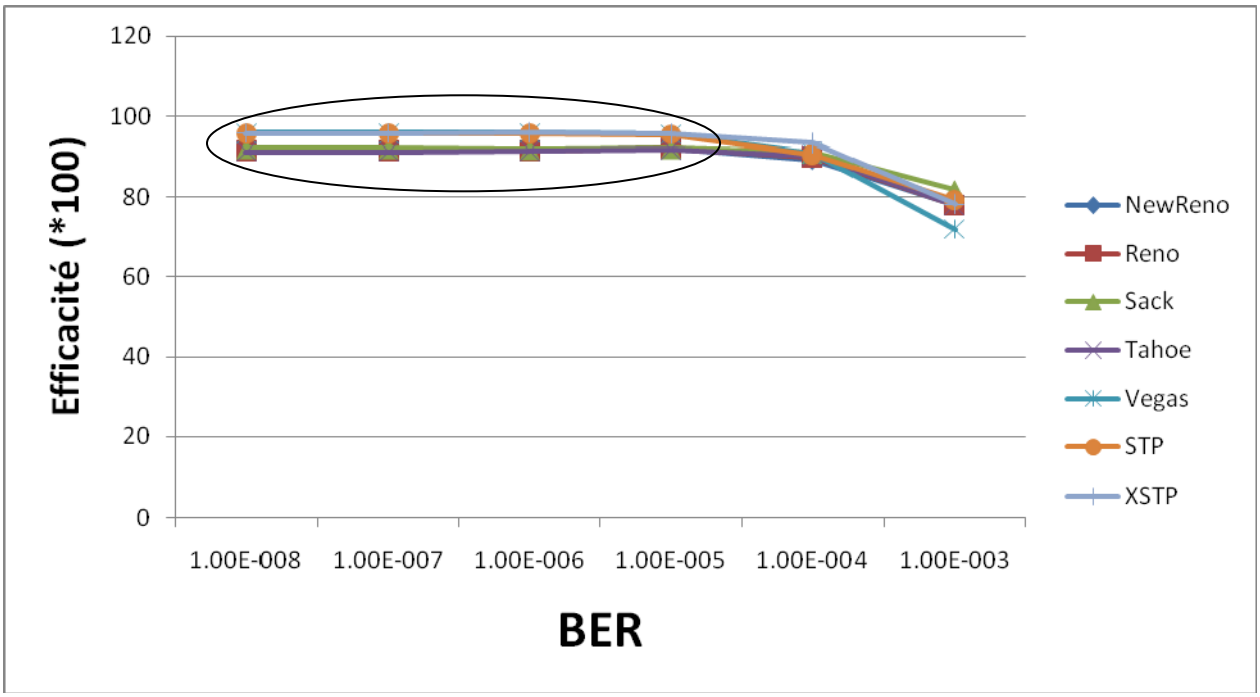

Figure VI.11. Efficacité de la transmission de données en fonction du BER dans le cas de transmission dans 2 sens

La Figure VI.12 c'est un zoom de la partie marquée sur la Figure originale VI.11.

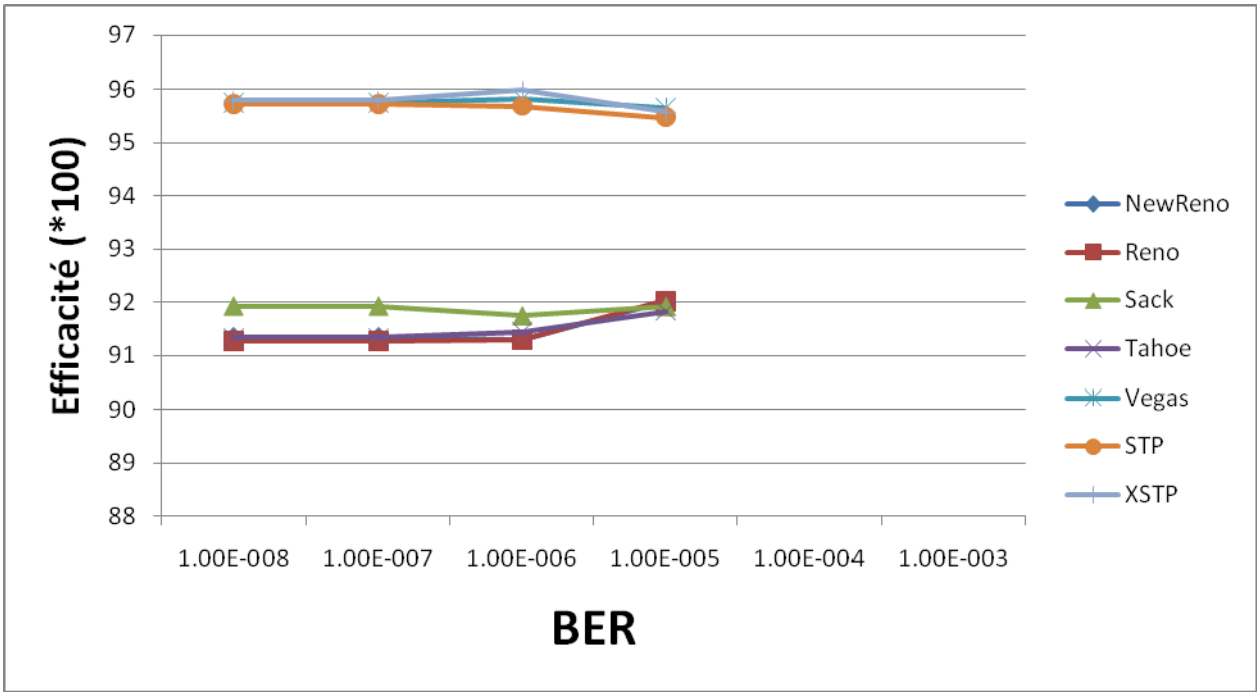

Figure VI.12. Zoom de la partie marquée sur la Figure VI.11

VI.2.4. Overhead

Dans la transmission dans les deux sens, XSTP est toujours le plus performant en termes de pourcentage d'overhead transmis par mémoire aux autres, comme dans le cas de la transmission dans un seul sens. La seule différence est que le pourcentage d'overhead n'augmente pas d'une façon très importante, comme dans la transmission dans un seul sens si la probabilité de perte est très élevée. Cela est due à la diminution des taux d'envoie par les protocoles, s'il y a d'autres transmissions dans le sens inverse.

L'un des choses les plus importantes dans le domaine des satellites est la consommation de l'énergie. Les chercheurs essaient toujours de minimiser l'énergie dépensée par les satellites dans la transmission de données. L'un des paramètres qui influe sur cette énergie est le nombre d'overhead transmis, l'énergie dépensée dans la transmission d'overhead considérer comme une énergie perdue. La Figure ci-dessous montre que le protocole XSTP consomme moins d'énergie que les autres protocoles dans la transmission d'overhead, donc il bénéfice d'une énergie supplémentaire qui l'utilise dans la transmission de données.

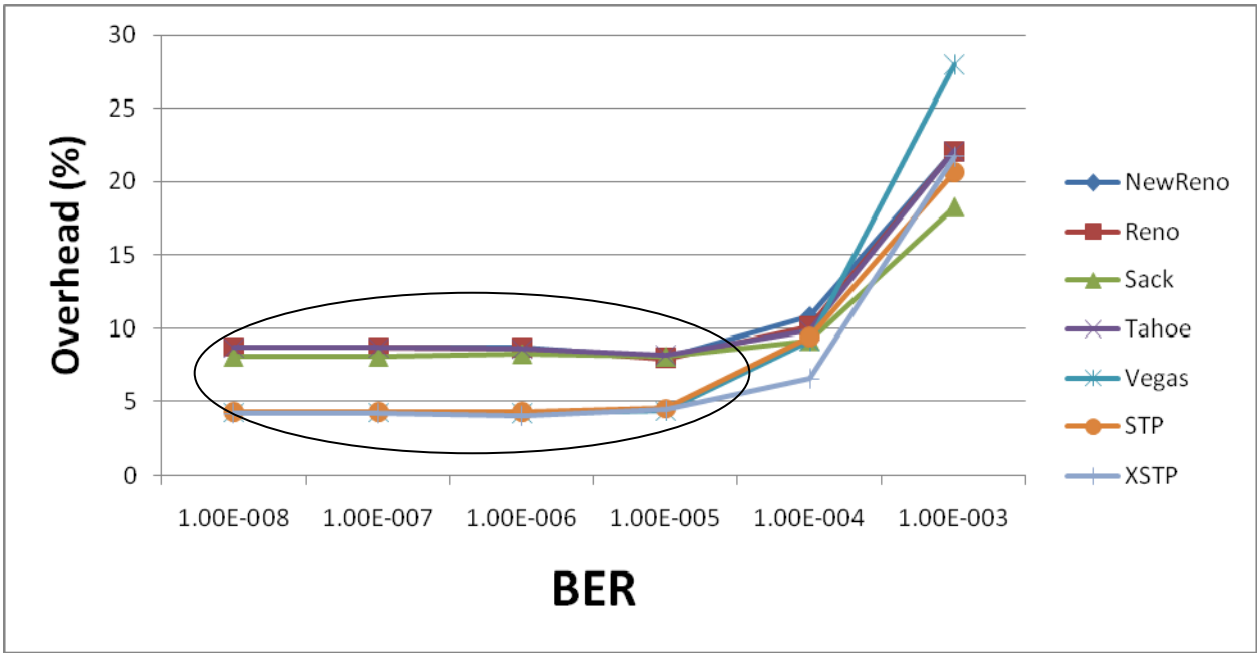

Figure VI.13. Overhead transmis en fonction du BER dans le cas de transmission dans 2 sens

La Figure VI.14 c'est un zoom de la partie marquée sur la Figure originale VI.13.

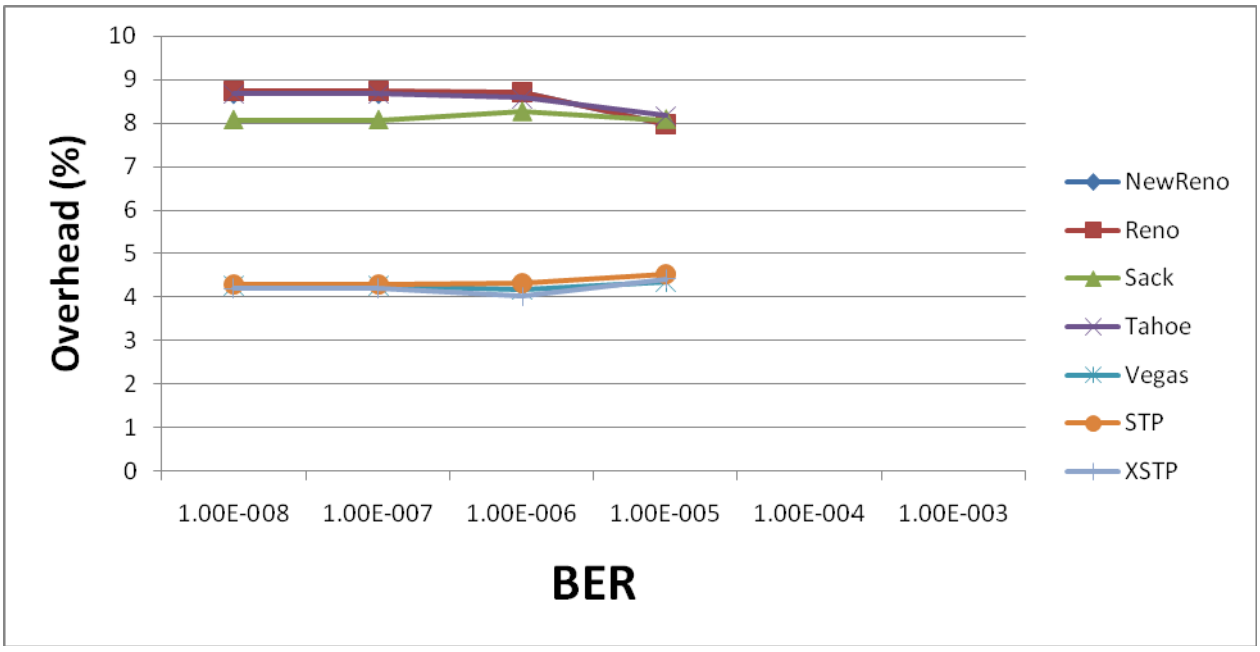

Figure VI.14. Zoom de la partie marquée sur la Figure VI.13

Conclusion

L'étude des performances du protocole XSTP a nécessité son implémentation dans le simulateur de réseaux NS2. Cette implémentation a été réalisée avec succès, malgré toutes les difficultés qui accompagnent en général une telle implémentation (maîtrise des langages C++ et OTcL, connaissance de la hiérarchie de classes de NS2). Nous avons basé dans cette étude sur la comparaison du protocole XSTP avec son protocole original STP et les variantes de TCP (clones TCP). Selon nos résultats de simulation, nous pouvons conclure que l'utilisation du protocole XSTP dans les nanosatellites, permet d'avoir un meilleur débit effectif atteint dans la transmission, un bénéfice dans l'énergie dépensée dans la transmission et une nécessité d'une très petite bande passante sur le canal inverse.

Le Tableau VI.1 contient les intervalles des valeurs atteint par le protocole XSTP pour tous les paramètres des performances, et cela dans la transmission dans un seul sens et deux sens.

	Débit effectif (Kb/s)	Bande passante nécessaire pour le canal inverse (Kb/s)	Efficacité	Overhead (%)
Transmission dans un seul sens	[68.49-1401.95]	[8.19-12.78]	[33.64-96.37]	[3.63-66.36]
Transmission dans deux sens	[42.59-1265.63]	[8.01-9.10]	[78.30-95.79]	[4.2-21.70]

Tableau VI.1. Résultats du protocole XSTP

CONCLUSION ET PERSPECTIVES

Conclusion et Perspectives

Les réseaux d'accès satellitaires LEO présentent un défi pour un transport efficace et fiable des données. La principale raison attribuée à l'échec des protocoles de transport standards est la diversité des caractéristiques des liaisons hétérogènes, y compris les liaisons satellitaires LEO. Plus précisément, ces protocoles sont centrés sur la congestion et ils manquent d'une stratégie de contrôle d'erreur qui est adaptable aux différentes conditions d'erreur dans le réseau. *Satellite Transport Protocol* (STP) est une proposition qui traite de nombreuses questions avec le transport des données sur des liaisons satellitaires. Mais, STP hérite le contrôle de congestion qui existe dans ses protocoles ancêtres [20].

Notre objectif était l'étude des performances d'un protocole de transport pour les réseaux nanosatellites. Ce protocole est une optimisation du protocole STP. Cette optimisation consiste en l'ajout d'une stratégie de contrôle d'erreur au protocole STP qui s'adapte à la condition d'erreur disponible dans le réseau. La stratégie est basée sur un mécanisme probing de bout-en-bout installé à l'émetteur STP et activé lorsque une perte d'un segment est détectée. La perte est détectée soit explicitement (réception d'un USTAT ou STAT qui contient des lacunes), soit implicitement (dépassement d'un *timeout*). L'idée du mécanisme probing est de suspendre la transmission de données quand une perte est détectée et attendre jusqu'à que la situation de l'erreur s'améliore avant d'ajuster les taux de transmission. Le mécanisme compare les deux RTT de la connexion avant et après le probing, afin de déterminer si la condition d'erreur est liée à la congestion ou liée au lien. Après cette comparaison, le mécanisme retransmet les segments perdus et reprend la transmission de nouvelles données.

Pour étudier les performances de l'optimisation du protocole STP, ce protocole est implémenté dans le simulateur de réseaux NS2. NS2 est beaucoup utilisé par la communauté de recherche dans les réseaux, aussi il contient les implémentations des clones TCP, ce qui nous a permis de faire des comparaisons. Le mécanisme probing est aussi implémenté comme une option dans l'émetteur STP. L'optimisation du protocole STP a donné naissance à un nouveau protocole de transport qui s'appelle *eXtended Satellite Transport Protocol* : XSTP.

Les tests effectués s'appuient sur la comparaison des performances du protocole XSTP avec le protocole STP et les clones TCP. Ces tests sont effectués dans un environnement simulé d'un réseau d'accès satellitaire LEO. Quatre paramètres de performances sont mesurés dans chaque test : le débit effectif; la bande passante nécessaire sur le canal inverse; l'efficacité d'utilisation de canal; et le pourcentage d'overhead transmis. Dans chaque test le BER est varié entre 10^{-8} et 10^{-3}.

Le mécanisme probing fonctionne essentiellement en investissant la transmission de certains overhead et de l'énergie pour enquêter sur la nature de l'erreur avant d'ajuster le taux de transmission. Cependant, ces overheads sont récupérés sous forme de gains de débit et des économies d'énergie. Les résultats des tests montrent que l'optimisation du protocole STP est plus performante que le protocole STP sans différenciation de types d'erreurs et les clones TCP. Ces résultats prouvés qu'il est inutile d'invoquer directement le contrôle de congestion (donc une dégradation dans le débit), ou d'envoyer plus de segments de données quand il y a une probabilité de les perdre. Dans tous les tests, le mécanisme probing aide le protocole à atteindre des niveaux plus élevés de débit et des niveaux inférieurs des overheads.

Un point important qu'il faut noter est que le mécanisme probing peut mélanger entre les conditions d'erreur de lien et de la congestion, en particulier lorsque des erreurs de lien sont plus fréquentes. Cet effet peut influencer sur les gains de débit atteint par le mécanisme probing pour les erreurs de non congestion. La même situation peut se produire aussi bien pour les overheads.

Comme perspectives à notre travail, il faut améliorer la stratégie de contrôle d'erreur, pour qu'elle ne puisse pas mélanger les conditions d'erreur de lien et de congestion, quand les erreurs de lien sont fréquentes. Il faut aussi trouver une décision plus sophistiquée qui met les critères de la fin du cycle probing, qui fait éventuellement appel aux éléments suivants : l'historique des décisions précédentes; les schémas des échanges de probing et quelques autres statistiques de la connexion. L'étude des performances du mécanisme probing sur des connexions qui ont de longue RTT (disponible dans les chemins des satellites GEO) sera l'objectif des prochains travaux.

BIBLIOGRAPHIE

[1] A. Benslimane, A. Abouaissa, *XTP specification and validation with LOTOS.*

[2] C. Caini, R. Firrincieli, M. Marchese, T. Cola, M. Luglio, C. Roseti, N. Celandroni, F. Potorti, *Transport layer protocols and architectures for satellite networks*, Int. J. Satell. Commun. Network, 2006.

[3] CCSDS Blue Book : *Space Communications Protocol Specification Transport Protocol (SCPS-TP) : Recommended Standard*, CCSDS 714.0-B-2, Octobre 2006.

[4] DC Palter, *Satellites And The Internet : Challenges And Solutions : Chapter 18 : Satellite Protocols,* 2008, pp 120-133, www.SatelliteTCP.com

[5] E. Altman, T. Jiménez, *NS Simulator for beginners*, Notes de cours, Univ de Los Andes, Mérida, Venezuela et ESSI, Sophia-Antipolis, France, Décembre 2003.

[6] J. W. Atwood, Y. Zhang, *A Definition of the XTP Service and its Formal Specification*, IEEE, 1995, pp 459-468.

[7] K. Fall, K. Varadhan, *The ns Manual (formerly ns Notes and Documentation)*, Le Projet VINT, Août 2000.

[8] M.E. Elaasar, Z. Li, M. Barbeau, E. Kranakis, Z. Li, *Satellite Transport Protocol Handling Bit Corruption, Handoff and Limited Connectivity*, IEEE Transactions on Aerospace and Electronic Systems, Avril 2005, pp. 489-502.

[9] M.E. Elaasar, Z. Li, M. Barbeau, E. Kranakis, *The eXtended Satellite Transport Protocol : its design and evaluation*, Proceedings of 17th Annual AIAA/USU Conference on Small Satellites, 2003, Logan, Utah.

[10] M. E. Elaasar, *XSTP: eXtended Satellite Transport Protocol*, thèse, 2003, Ottawa-Carleton Institute for Computer Science, School of Computer Science, Carleton University, Ottawa, Ontario.

[11] M. M. Burlacu, *A study concerning the nanosatellite systems*, Mémoire de stage, TELECOM & Management SudParis, 2007, Evry, France.

[12] M. M. Burlacu, J. Kohlenberg, *An analysis of the nanosatellites launches between 2004 and 2007*, Mémoire de recherche, TELECOM & Management SudParis, 2007, Evry, France.

[13] *NSP Packet Protocol for Sinclair Interplanetary Hardware*, Rev 1.0, Février 2008, www.sinclairinterplanetary.com

[14] P. Anelli, E. Horlait, *NS-2: Principes de conception et d'utilisation.* UPMC-LIP 6 : Laboratoire d'informatique de Paris VI, Septembre 1999.

[15] R. C. Durst, G. J. Miller, E. J. Travis, *TCP Extensions for Space Communications*.

[16] R. M. Sanders, A. C. Weaver, *The Xpress Transfer Protocol (XTP)*, A Tutorial, Computer Networks Laboratory, Department of Computer Science Thornton Hall University of Virginia.

[17] R. Wang, N. C. Aryasomayajula, A. Ayyagari, Q. Zhang, *An Experimental Performance Evaluation of SCPS-TP over Cislunar Communications Links*, WCNC proceedings, 2007, pp 2605-2609.

[18] T. Estier, *Nanosatellites*, Travail pratique de diplôme, Février 1999, Ecole Polytechnique Fédérale de LAUSANNE.

[19] T. R. Henderson, R. H. Katz, *Transport Protocols for Internet-Compatible Satellite Networks*, IEEE Journal on Selected Areas of Communications, 1999

[20] T. R. Henderson, R. H. Katz, *Satellite Transport Protocol (STP) : An SSCOP-based Transport Protocol for Datagram Satellite Networks*, 2ème International workshop on Satellite-based Information Services, 1997, Budapest, Hungary.

[21] T. S. Tuli, N. G. Orr, R. E. Zee, *Low Cost Ground Station Design for Nanosatellite Missions*, AMSAT, North American Space Symposium, 2006.