Rapport de stage
Concepteur Développeur en Ingénérie Logicielle
eXia 2006-2007

OBS et NS-2

Béchu Jérôme

UQAM
Montréal

Directeur de recherche : Dr. Elbiaze

A mes parents, pour leur soutien inconditionnel.

Remerciements

Je tiens à remercier Mme Halima ELBIAZE, pour avoir accepté de m'accueillir dans son équipe de recherche. Je vous remercie chaleureusement de m'avoir offert cette chance d'intégrer votre équipe de recherche ainsi que votre aide financière.

J'aimerai également remercier mes collègues Dr Wael Aly, Jaafar Wael et en particulier Mr Faten Zhani pour leurs soutiens journalier.

Enfin j'adresse également tous mes remerciements à l'équipe du CESI de Reims pour m'avoir soutenu lors de ma candidature.

Table des matières

1 Introduction

Avec environ 30 millions de Français âgés de 11 ans et plus qui se sont connectés à Internet au cours de juin 2007, cela représente environ 58 % d'internautes dans la population (source : Médiamétrie). Leur nombre a progressé de 12 % sur un an. L'engouement pour les nouvelles technologies couplé à l'apparition de nouvelles applications (la Voix sur réseau IP (VoIP), vidéo-conférences, télévision numérique ...) gourmandes en débit ont pour conséquence une demande de plus en plus importante en terme de bande passante.

La fibre optique permet de répondre à cette nouvelle attente. Cependant cette technologie a une problématique importante, elle ne dispose pas de stockage mémoire (tampon). C'est à travers ce constat que la communication par rafales sur fibre optique (Optical Burst Switching (OBS)) a fait son apparition. Elle permet de résoudre ces problèmes d'absence de tampon mémoire par une signalisation de messagerie avancée.

Dans ce rapport nous allons tout d'abord étudier cette technologie et ces principes de fonctionnement. Puis nous allons détailler le simulateur de réseau NS2 que nous utilisons dans le laboratoire. Dans la suite de ce rapport concernera ma contribution au projet. La première partie consiste à implémenter les protocoles de signalisation de OBS dans le simulateur de réseau. La seconde partie concernera la création de l'éditeur de topologie.

1.1 Présentation de l'université du Québec à Montréal

L'université du Quebec à Montréal (UQAM) a été créé en 1969. C'est cette année là que le gouvernement du Québec a fusionné l'école des Beaux-arts, le collège Sainte-Marie et trois écoles normales pour créer l'UQAM. Au départ l'UQAM accorde une grande importance aux lettres et aux arts, mais également aux sciences humaines et la formation des maîtres. Au milieu des années 70, l'UQAM se tourne vers les sciences de la gestion appuyé par la forte demande des gestionnaires. Depuis les années 80, l'UQAM donne une place importante aux sciences pures et appliquées.

Le département informatique a été quand à lui créé en 1984. Il compte plus de 40 professeurs dans différents domaines de l'informatique, dans l'enseignement et dans la recherche. Le laboratoire de recherche en télé-informatique est attaché au département informatique. Il a été fondé en 1996 par le professeur Omar CHERKAOUI. Aujourd'hui de nombreux axes de recherche sont explorés par ce laboratoire comme la virtualisation des routeurs, les réseaux sans fil ou encore les réseaux de commutation optiques.

1.2 L'équipe de recherche

La directrice et responsable de stage est le Docteur Halima Elbiaze. Sous sa direction l'équipe est consitué d'un étudiant en doctorat, Mohamed Faten Zhani, un post doctorat, Dr. Wael Aly et une étudiante en maîtrise, Tindo Sanghapi Judith Noël.

1.3 Plannification

L'objectif principal était de répondre à deux besoins au bout des six mois de stage. Il n'y avait pas de calendrier précis pour jalonner mes résultats puisque les demandes étaient en constantes évolution. Des réunions hebdomadaires nous permettaient de nous recentrer par rapport à la ligne directrice. Il était très difficile de rédiger une plannification en début de stage puisque mon travail évoluait en fonction de mes résultats et de mes capacités.

2 Les réseaux optiques

Les réseaux fibres optiques offrent de nombreux avantages tel que la largeur de la bande passante, ou encore sa capacité théorique qui peut aller jusqu'à 50 Tb/s. La fiabilité de ce type de réseau est un autre avantage par rapport aux réseaux électroniques. Cette fiabilité repose sur le fait que le signal lumineux subit moins d'atténuations que le signal électrique. La robustesse des liens permet d'installer la fibre optique quelques soient les conditions. Ces caractéristiques font de ce médium une alternative majeure dans le transport de données sur les réseaux de prochaine génération et par conséquent une technologie du futur de l'internet.

A l'heure actuelle une contrainte importante vient perturber la progression de cette technologie. La fibre optique ne dispose pas tampon mémoire permettant de réaliser les opérations de routage. Il n'est pas possible de stocker la lumière comme les protocoles de commutation des réseaux électrique le font actuellement. Une solution alternative existe : l'utilisation des conversions opto-électro-optique (O/E/O) et des mémoire de type RAM. Cependant les temps de transmission seraient alors amoindris du fait des temps de conversion ce qui ralentiraient globalement le réseau.

2.1 Architecture des réseaux

L'apparition de la technologie fibre optique dans les réseaux et notamment l'utilisation de réseau tout optique a entrainé l'apparition logique de nouvelles normes comme le démontre la figure 2.1.

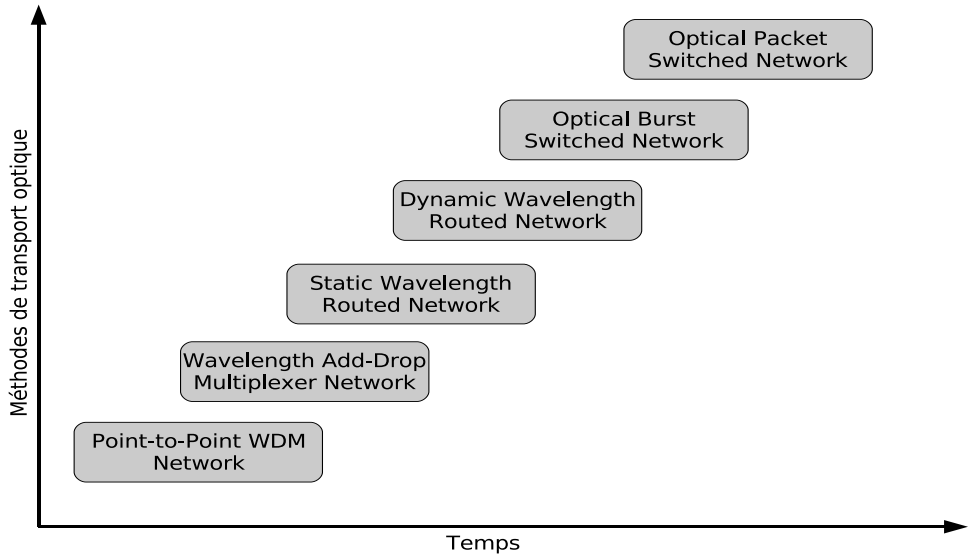

Fig. 2.1 – Evolution des méthodes de transport optique

Parmis les avancées technologiques, le multiplexage de longueur d'onde ou **Wavelength Division Multiplexing** (WDM) permet de multiplier les capacités de transmission. Avec sa largeur de bande importante, la fibre optique peut utiliser le **multiplexage spectral**. Celui-ci consiste à envoyer plusieurs signaux optiques simultanément sur la même fibre optique. Chaque signal emprunte un canal de communication différent qui sont en réalité des plages de fréquences.

point-to-point WDM links est la première génération de réseaux optiques. Ce type de réseau est constitué de noeuds reliés les uns aux autres. Pour chaque noeud le signal subit une convertion O/E/O, puis les données sont traitées puis transmises au noeud suivant. La somme de ces opérations ont un coût important en terme de temps de traitement et de complexité.

Du fait de l'immaturité matérielle des réseaux optiques, il n'est pas possible de transposer l'état de l'art des réseaux électronique vers les réseaux optiques. Dans ce domaine de nouveaux concepts ont fait leurs apparition, en voici les trois grandes familles :

- La commutation de circuits (OCS), orientée connexions ;
- La commutation de paquets (OPS), orienté pacquets ;
- La commutation par rafales (OBS), intermédiaire entre circuits et paquets.

2.1.1 Commutation de circuits optiques (OCS)

Dans un mode commuté par circuits, une connexion est établie pour toute la durée de la communication, comme dans le réseau commuté téléphonique. Un certain nombre d'opérations sont nécessaires pour ce type de commutation :

- La découverte de la topologie et des ressources (Open Shortest Path First) ;
- Le routage et l'allocation de longueurs d'ondes ;
- La contrainte de conservation des flots : proportion entre le flot rentrant et sortant ;
- La continuité des longueurs d'ondes, le signal doit utiliser la même longueur d'onde de la source à la destination ;
- La contrainte de capacité d'une longueur d'onde : la quantité totale du traffic affecté à une longueur d'onde ne doit jamais dépasser la capacité de celle-ci.

La signalisation et la réservation des ressources sont importante car elle assure la réservation de la source à la destination. Une fois la longueur d'onde réservée, la transmission est généralement tout optique et ne nécessite ni temps de calcul ni capacité de stockage. La réservation utilise un accusé de réception qui se traduit par un temps de latence pendant l'établissement des connexions. Une fois les connexions établies, le cheminement du traffic est garanti ainsi que les délais. Pour augmenter la charge du réseau, il est nécessaire d'augmenter la complexité des noeuds. Une panne sur ce type de réseau provoque des contentions. Si un lien se brise, un nouveau chemin doit être négocié et les données doivent donc attendre. Une solution existe dans l'allocation de chemins de protection. Si le traffic est variable, une partie de la bande passante ne sera pas utilisée.

2.1.2 Commutation de paquets (OPS)

La commutation de paquets consiste à faire transiter les données de proche en proche sous forme de paquets. Un paquet est un volume de données, précédé d'une entête contenant les informations nécessaires au routage. Ce type de stratégie est proche de la commutation IP, à ceci près qu'il n'y a pas de mémoire tampon. Pour fournir les informations au routeur, les entêtes du paquet sont convertis par O/E/O pendant que les données sont temporisées dans des fibres à retardements. Ce type d'opération dégrade les performances du réseau, de plus les fibres à retardements ne sont pas

encore au point.

Cette commutation offre une bonne granularité, ce qui permet une adaptabilité à la variabilité du traffic. Notons que pour chaque noeud le calcul sur la table de routage est effectué pour déterminer le chemin approprié.

2.1.3 Commutation par rafales (OBS)

La notion de rafale consiste à réserver pendant un temps donné un lien, permettant d'envoyer un paquet. Cette commutation est un compromis entre la commutation de circuits et la commutation de paquets. Il permet d'optimiser l'utilisation des ressources du réseau, il offre une bonne granularité, une robustesse aux pannes ainsi qu'une adaptabilité aux variations de traffic. OBS utilise un système de contention, qui consiste à envoyer un paquet de contrôle précédent la rafale après un temps appellé offset time. Cette contention n'utilise pas de stockage, pas de calcul et pas de conversion O/E/O.

2.1.4 Conclusion

La figure 2.2 présente ces trois modes de commutation. Des données sont émises depuis la source (S) vers la destination (D) en traversant trois noeuds (1, 2 et 3).

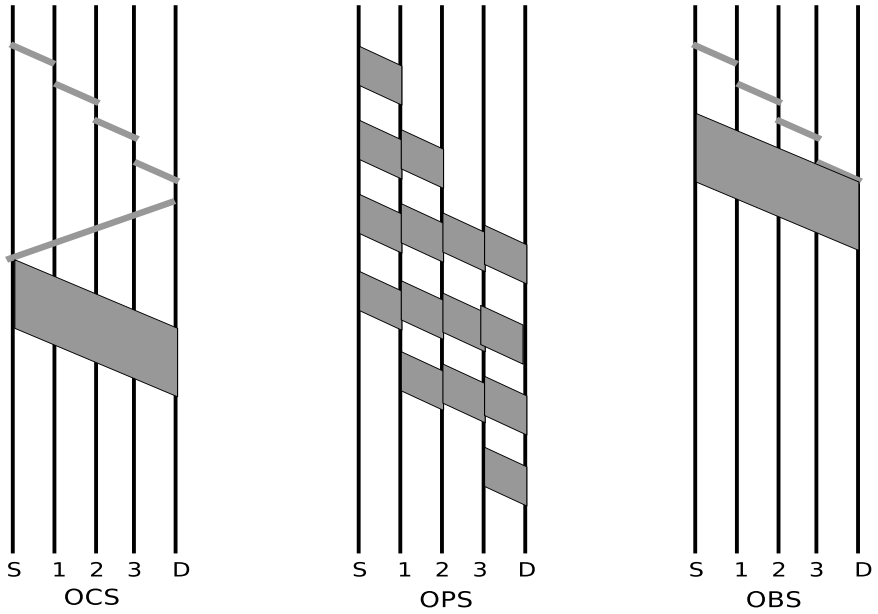

FIG. 2.2 – Comparaison de trois modes de commutation optique

Chacune de ces commutations correspond à un type de réseau et une variabilité du trafic. La mise en place des ces technologies requiert une maturité matérielles différentes pour chacune. La commutation de circuit est déjà utilisée du fait que les composants matériels existent. Cependant la demande de plus en plus importante de performances pousse la recherche à aller de l'avant. OBS s'impose comme un compromis entre complexité et granularité.

2.2 Architecture des réseaux OBS

Un réseau OBS est tout optique, c'est à dire que les routeurs ne font pas de conversion O/E/O hormis le traitement des paquets de contrôle qui transitent sur des longueurs d'ondes dédiées. OBS est consituté de noeuds (routeur ou station) qui peuvent être des **edge node** (noeud de frontière) et des **core node** (noeud de coeur). Ils sont reliés par des liens de fibre optique. Un noeud de frontière accumule du traffic électronique (IP, AIM, ...) pour construire ensuite des rafales qui seront injectées dans le réseau optique. Ces rafales sont appellées Burst. Le noeud de coeur est tout optique, il a pour rôle de commuter les paquets à l'intérieur du réseau. Ce noeud dispose de ports dédiés aux messages de contrôle, qui sont converti pour obtenir les informations permettant de définir le chemin du Burst correspondant.

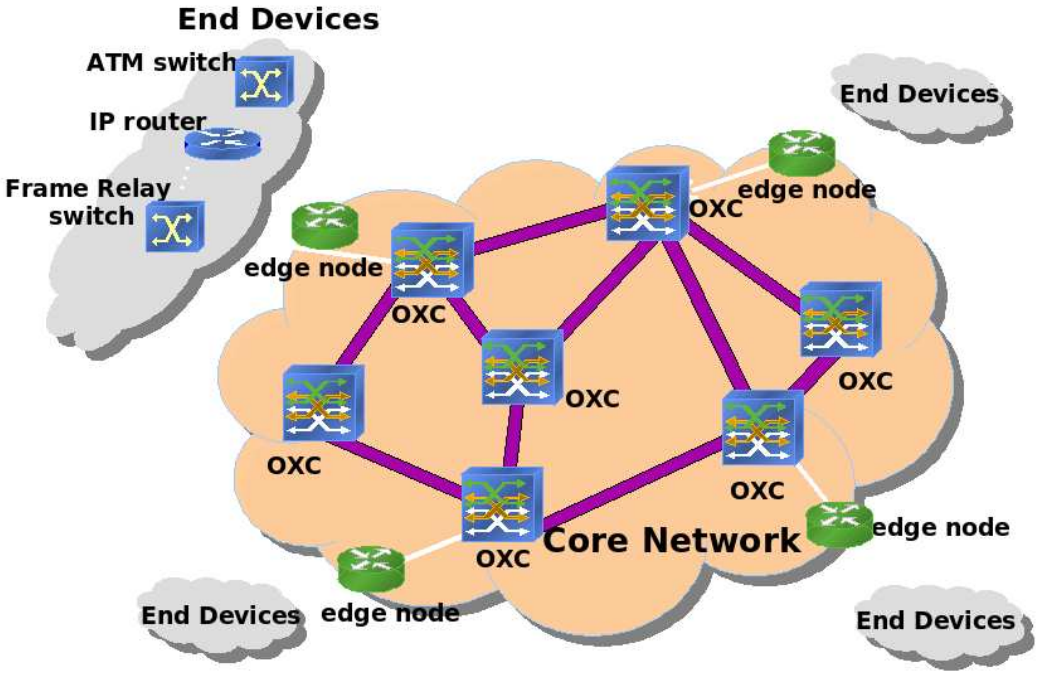

FIG. 2.3 – Les réseaux OBS

Les longueurs d'ondes de la fibre optique sont organisées en deux groupes, le premier est réservé pour les paquets de contrôle et le second groupe pour les paquets de données. L'entête des paquets est envoyé en éclaireur sur le réseau en utilisant les longueurs d'ondes de contrôle. Ce type de paquet est un **Burst Header Packet** ou BHP (parfois appellé setup). Le principe de ce paquet est de prendre les décisions concernant la commutation pour le paquet de données qui va suivre. Les préparatifs de commutation sont appellés **switch cross-connect setting**. Avant l'émission du paquet de données, l'émetteur respecte un délai permettant au paquet de contrôle de parcourir le réseau jusqu'à la destination et donc de configurer tous les routeurs intermédiaires. Ce délai est appellé **offset**. La valeur de l'offset résulte d'une addition d'un certains nombre de traitements :

- Le temps de conversion O/E/O à chaque routeur ;
- Le délai des files d'attentes ;
- Le temps de configuration du switch (configuration des mirroirs).

Le choix de l'offset est déterminant pour les performances d'un réseau OBS. En effet si cette valeur est trop petite, des phénomènes de congestion peuvent subvenir (dans le cas ou le Burst est envoyé

par la source avant que le BHP atteigne la destination). Si la valeur est trop grande, des phénomènes de congestion peuvent être amplifier. La plupart des configurations utilisent un offset fixe, ce qui rend le réseau insensible au changement de traffic.

Les **Optical Cross-Connect** ou OXC permettent de gérer les longueurs d'ondes, de regénérer les signaux et de les router. Ce sont les routeurs des réseaux OBS, les noeuds de coeur. Parfois il dispose également d'un dispositif de convertion d'ondes.

2.2.1 La table de routage

Avec la version du module actuel, la table de routage est déterminée une seule fois au début de l'application. Cette table est ensuite fixe pour tout le reste de la simulation. Le chemin le plus court est déterminé entre deux noeuds en fonction du poids de chaque lien.

2.2.2 La création des rafales (Burst Assembly)

Dans un réseau OBS, le traffic entrant des clients provenant de différentes sources arrivent sur un edge node. Celui-ci a pour tâche d'aggréger les paquets en entrée en utilisant des d'algorithmes. Ces algorithmes dépendent de paramètres tel qu'une destination commune, un type de service ou encore la mise en place d'une qualité de service (QoS). QoS permet de classifier les paquets entrant en fonction de leurs types. Par conséquent on peut trouver dans un edge node plusieurs queues en fonction des destinations ou du QoS. Les paquets sont assemblés en fonction de seuil de tolérance, qui sont généralement une base de temps (timeout) ou un seuil basé sur la taille. Parfois l'aggrégation est hybride et basée sur les deux paramètres. Le burst est ainsi créé.

2.2.3 Contention

Dans un core node, lorsque plusieurs bursts arrivent sur un core node pour utiliser la même longueur d'onde de sortie, le noeud subit une **contention**. Généralement trois types de solutions sont proposées pour résoudre ce problème. La première solution est appellée **Wavelenght domain**, le burst en relation avec la contention est envoyé sur une longueur d'onde différente. **Time domain** est la seconde solution. Elle propose d'appliquer un délai jusqu'à la résolution du problème. Dans ce cas on a recours a un mécanisme appellé Fiber Delay Link (FDL) qui est un composant hardware permettant de temporiser un burst (honéreux, composant immature). La troisième et dernière solution est nommée, **space domain** ; utilisation des déflections qui consiste à choisir un chemin alternatif pour notre burst afin qu'il rejoigne sa destination. On peut imaginer une combinaison d'une ou plusieurs de ces types de résolutions en fonction de la topologie. En cas de non résolution du problème de contention, le burst est tout simplement jeté (drop).

2.2.4 Signalisation

Un chemin de communication optique entre deux noeuds est appellé lightpath. La Signalisation consiste à réserver un chemin pour le data burst sur un réseau OBS. Différents protocoles de signalisation existent. Nous allons en détailler les principaux. Une partie de mes travaux concernent la mise en place de ces algorithmes dans le simulateur de réseau que nous allons étudier dans le prochain chapitre.

Un protocole de signalistion du point de vue d'OBS se distingue par trois caractéristiques principales qui sont : la direction, la réservation, et la libération.

Pour utiliser les protocoles de signalisation, il est nécessaire d'utiliser trois types de BHP :

1. BHP Setup : ce paquet a pour rôle de configurer les ressources du réseau OBS pour le burst correspondant.
2. BHP Confirm : ce paquet permet de confirmer la réception du BHP Setup.
3. BHP Release : ce dernier a pour tâche de libérer les ressources.

Direction

Dans la théorie OBS, on apelle direction la phase préparatoire à l'envoi d'un burst sur le réseau.

La **direction qui a un seul sens** est schématisée par la figure 2.4. La plupart des configurations OBS utilise ce type de direction pour configurer le chemin à travers le réseau. La permière étape consiste à envoyer une message BHP Setup. Ce message contient les informations concernant le burst correspondant, et configure les cores nodes jusqu'à la destination. Le BHP est transmis sur une longueur d'onde dédiée aux paquets de contrôle. Le burst est transmis après un délai offset. Ce délai correspond aux temps additionnés du temps de parcours des liens et du temps de configuration des switchs. Le burst est ensuite transmis sur une longueur d'onde spécifique aux données. Le problème latent de cette configuration est que si le BHP Setup échoue lors de la réservation du chemin, le burst est tout de même émis avec pour résultat la suppression des données sur le noeud congestionné.

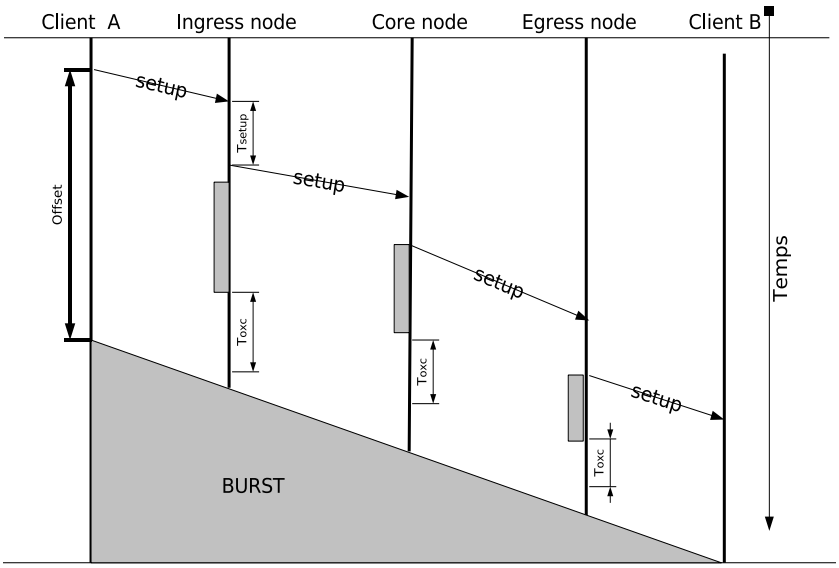

Fig. 2.4 – Direction à un seul sens

La **direction à un double sens** est schématisé par la figure 2.5. Ce type de configuration nécessite un message de contrôle supplémentaire. Ce second message doit être émis du destinataire vers l'émetteur, permettant à ce dernier d'envoyer les données. Ce type de direction est peu utilisé car il augmente de façon significative le temps d'émission d'un burst avec de surcroît des problèmes potentiels de congestion dûs à l'ajout d'un message supplémentaire à chaque émission de burst.

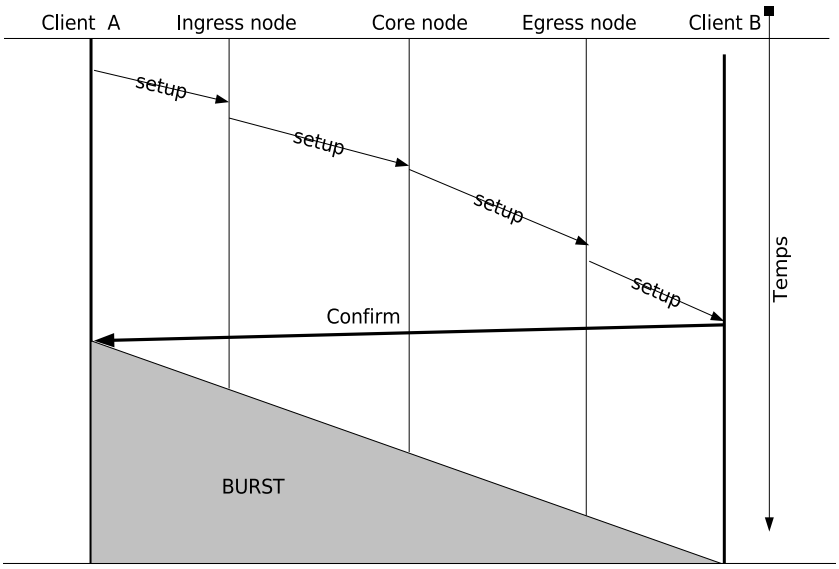

FIG. 2.5 – Direction à double sens

Libération des ressources

La libération des ressources consiste à la libérer des canaux de données alloués pour un burst. Il existe actuellement deux types de libération.

Libération implicite (Implicit Release) Dans ce premier cas, lorsque le burst est envoyé au prochain noeud, la ressource allouée est directement libre après le passage du burst.

Libération explicit (Explicit Release) A l'inverse du cas précédent, ici la destination doit envoyer un message pour libérer les ressources tout au long de ce chemin. Ce message s'appelle BHP Release, il doit parcourir le chemin en sens inverse.

2.2.5 Choix de la longueur d'onde

La choix de la longueur d'onde est primordial. Si le système ne dispose pas de conversion de longueur d'onde, le choix de la longueur d'onde à la réception du BHP sera irrémédiable. Des paramètres importants sont à prendre en compte comme la métrique du réseau, ou encore la charge des longueurs d'ondes.

Réservation immédiate

La longueur d'onde est réservée pour un burst immédiatement après l'arrivée du message BHP Setup correspondant. Si aucune longueur d'onde n'est disponible, le BHP est rejeté et le burst ne sera pas traité (drop).

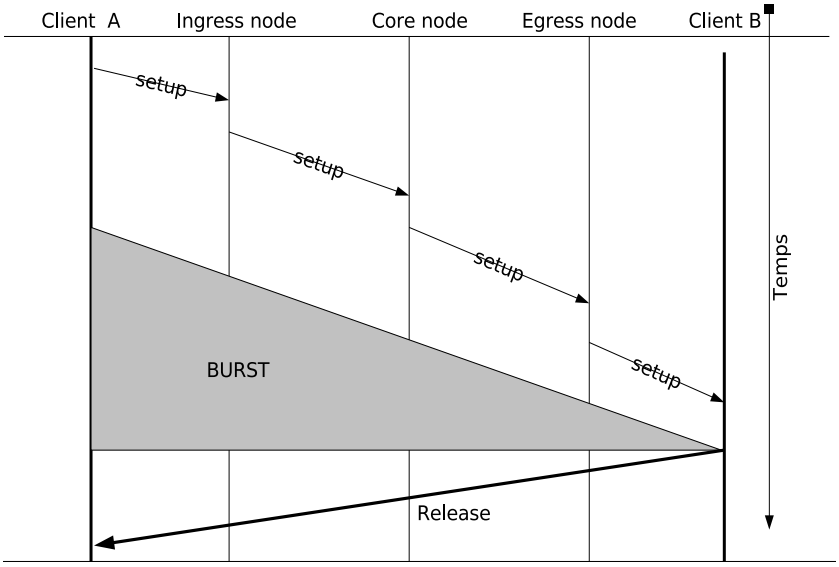

FIG. 2.6 – Libération explicite

Réservation retardée

Ce type de réservation consiste à optimiser l'utilisation des longueurs d'ondes en ayant recours à une gestion dans le temps (un calendrier de la longueur d'onde). Cette réservation intervient sur une longueur d'onde à partir d'une heure T et pour un délai D. Les algorithmes définis plus bas concernant la plannification des bursts sont alors utilisés (LAUC).

Algorithme de plannification des bursts (Burst Scheduling Algotithm)

Lorsqu'un Burst Header Packet ou message de contrôle arrive sur un core node un algorithme de scheduling est utilisé pour lui assigner une longueur d'onde de sortie. Différents types d'alogithme sont utilisés dans la littérature mais nous allons en citer deux qui sont : **Last Available Unused Channel And Schedule** (Dernier canal non utilisé et non plannifié) et **Last Available Unused Channel And Schedule Void Filling** (Dernier canal non utilisé et non plannifié avec remplissage des vides).

Last Available Unused Channel And Schedule (LAUC)

Cet algorithme a pour rôle d'assigner quel port et quelle longueur d'onde doivent être réservé pour un burst. (partie gauche de la figure 2.7) Lorsque le BHP est reçu nous sommes en mesure de déterminer le temps T dans lequel le data burst arrivera, puisque nous disposons d'informations telles que la taille du burst, le délai offset choisi ou encore la capacité du lien. LAUC dispose d'un nombre D de channel. Lorsqu'un BHP arrive l'algorithme recherche les canaux disponibles pour le temps T (c'est à dire l'arrivée du burst). Si plusieurs canaux sont libre au moment de cette réservation, le choix va s'effectuer en prenant le canal ayant la différence entre le temps T du burst précédent et ce burst le plus court.

Last Available Unused Channel And Schedule - Void Filling (LAUC-VF)

Le principe est similaire à l'algorithme précédent mis à part que l'on privilégie les espaces vides entre deux bursts (cette espace est appellé void). A l'arrivée d'un burst on regarde si un canal de données peut contenir un burst entre deux bursts prévus auparavent. (partie droite de la figure 2.7)

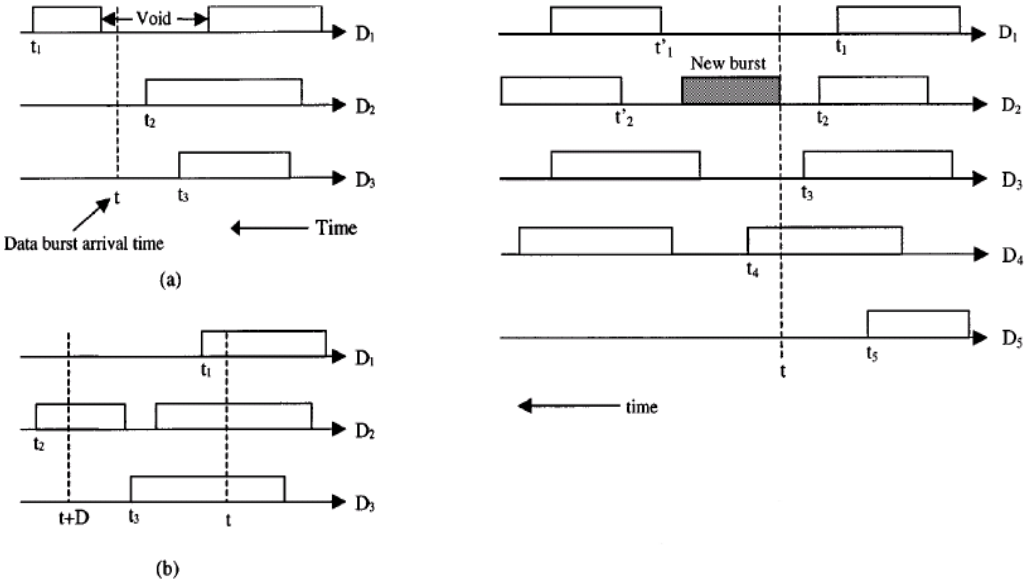

FIG. 2.7 – LAUC et LAUC-VF

3 Network Simulator 2

3.1 Introduction

NS-2 (Network Simulator, 2 pour la génération) est un logiciel de simulation de réseaux informatiques. Il est basé sur des principes de conception objets, de réutilisabilité du code et de modularité. Ce logiciel est dans le domaine publique. NS-2 est très populaire du fait qu'il est exessivement extensible. De plus l'utilisateur dispose d'une documentation exhaustive. NS-2 permet la simulation de commutation de paquets, il dispose de nombreuses fonctionnalitées comme les algorithmes de routage unicast, multicast, la simulation de procotoles de transport, de session et de réservation. Il permet également de simuler des protocoles d'application tel que le HTTP ou le FTP. Il dispose de nombreux systèmes d'ordonnancement et de politique de file d'attente.

Ces caractéristiques propres et surtout son architecture modulaire explique qu'il est aujourd'hui un standard du domaine. Nous allons justement nous pencher sur cette architecture si particulière, en définissant ces composantes logicielles. Puis nous traiterons du module OBS, de ses spécificités et de son fonctionnement.

NS-2 a été créer en 1989, c'est une variante du REAL network simulator. Il a été développé par Lawrence Berkley National Laboratory. Depuis le projet est promu par plusieurs institutions de recherche en collaboration tel que USC/ISI, Xerox PARC, LBNL, et UCB sous le nom de projet **VINT** (Virtual InterNetwork Testbed) [1].

La troisième génération de ns est déjà en cours de développement depuis le 1 juillet 2006. Il est prévu de développer ce nouveau logiciel sur 4 ans.[2]

3.2 Architecture du simulateur

NS-2 est écrit en c++. Pour créer une simulation, il faut au préalable créer une topologie. Cette topologie doit être rédiger en OTcl (Object Tool Command Line). OTcl est un language de script orientė objet basė sur TCL (Tool Command Line).

NS-2 utilise ce type de document en entrée pour réaliser des simulations offrand des résultat qui pourront être étudiés par la suite.

Le simulateur est donc orienté objet. Une simulation consiste à la circulation de paquets (Objet) entre différents objets représentant les éléments du réseau. Concrètement les objets simulés utilisent un pointeur sur l'objet paquet qui navigue ainsi d'objet en objet à l'aide d'un mécanisme de calendrier d'événement (un scheduler).

[1] http ://www.isi.edu/nsnam/vint/
[2] http ://www.nsnam.org/

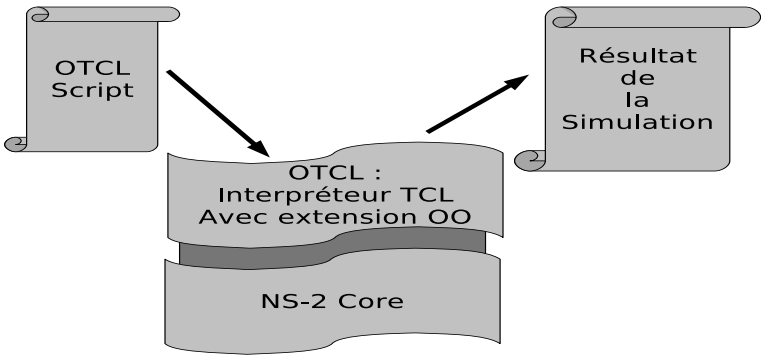

FIG. 3.1 – Vue simplifiée de ns

3.2.1 NS-2 : Une Pile logicielle

NS-2 sépare les données de l'implémentation, c'est pourquoi NS-2 regroupe un ensemble de logiciel qu'il utilise comme des poupées russes (emboîtement logiciel).

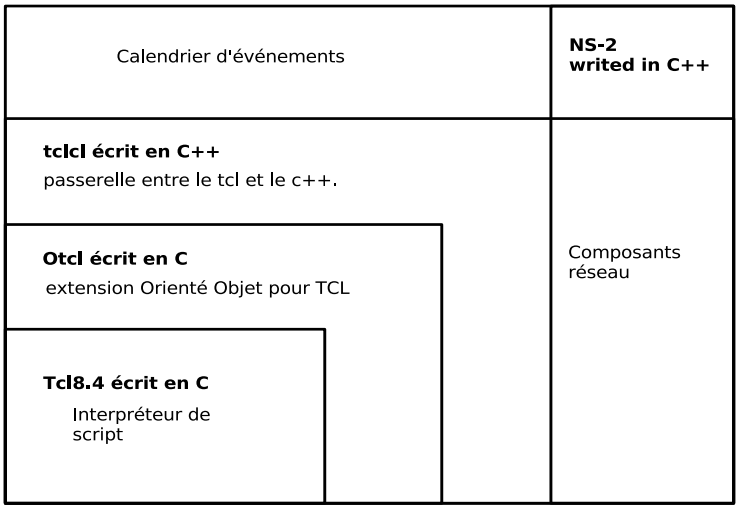

FIG. 3.2 – Architecture de NS-2

TCL

TCL pour Tool Command Line est un langage de script conçu en 1988. Ce langage est à typage dynamique, multi-plateforme et extensible. Il est facile à apprendre et repose sur 11 règles syntaxiques. TCL s'interface très facilement avec le langage C, de plus il est économe en mémoire vive.

Exemple de script :

```
% set fruits "pomme orange banane"
% puts [lindex $fruits end]
banane
```

OTcl

OTcl est une extension du langage de commande TCL, qui utilise une programmation structurée (boucles, procédures, notions de variables). Le moteur OTcl transforme les instructions TCL en instructions C++.

Exemple de script :

```
% Class Bagel
% Bagel abagel
% abagel info class
Bagel
% Bagel info instances
abagel
```

TCLcL (TCL with classes)

TCLcL est une interface TCL/C++. C'est une couche de liaison (appellé glue dans la littérature) entre le c++ de NS-2 et le TCL. Il permet de manipuler le script TCL en C++.

NS-2

NS-2 est donc le coeur du simulateur, il est écrit en c++ pour des raisons de performance. Pour créer une simulation il faut tout d'abord définir la topologie du réseau c'est à dire les noeuds et les liens. Puis définir les agents et les applications. Un agent est un objet de couche 4 du modèle OSI ; cela peut être du TCP, UDP ou TCP Sink. Une application est génératrice de traffic sur le réseau. Nous pouvons par exemple utiliser du FTP, telnet ou autre. L'étape suivante consiste à créer les connexions entre les agents puis de lancer les générateurs de traffic. Enfin la dernière étape consiste à définir le début et la fin de la simulation.

La liste des principaux composants acutellement disponible dans NS-2 sont :
- Application : Classe mère de toutes les applications (FTP, TELNET, ...).
- Agent : Classe mère des protocoles de niveau 3 et 4 (TCP, UDP).
- Node : Représente l'ensemble des noeuds du réseau. Chaque noeud contient un queue de données (classifier) pour envoyer un paquet arrivant d'un agent ou d'une interface vers la bonne sortie.
- Queue : Classes mères des buffers (RED, DropTail).
- LinkDelay : Simule les délais de propagations.
- Packet : Classe mère des paquets contient le header.
- TimeHandler : Classe mère des timers.

Voici un tableau listant quelques composants actuellement prit en charge par NS-2 :

Application	Web, ftp, telnet, générateur de traffic
Transport	TCP, UDP, RTP, SRM
Routage	Statique, dynamique
Gestion des files d'attentes	RED, Drop Tail, Token bucket
Mac	CSMA/CD, CSMA/CA, point to point link

Il existe des passerelles entre le c++ et le tcl. Comme le montre l'image 3.4, les objets crées dans OTcl sont également présents dans le c++. Il existe également un mécanisme de partage des

variables entre le language de script et le code c++ appellé **bind**. En utilisant ce procédé nous pouvons modifier directement la valeur d'une variable en c++ depuis le script.

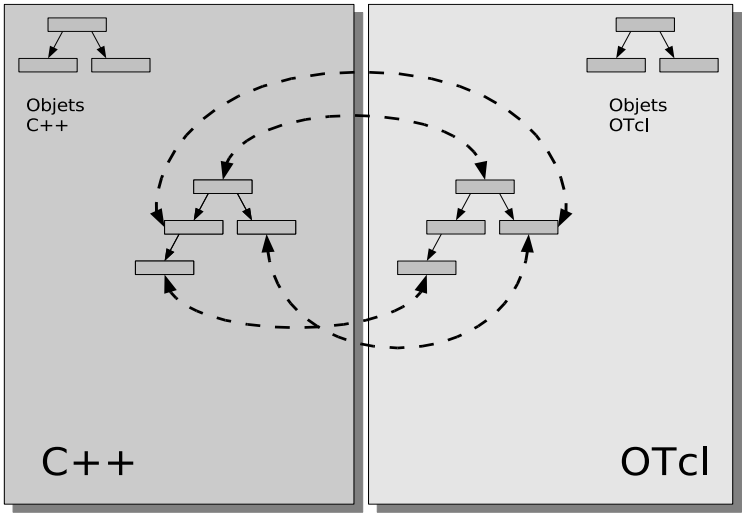

FIG. 3.3 – C++ et OTcl : Dualité

3.2.2 Description d'une topologie simple

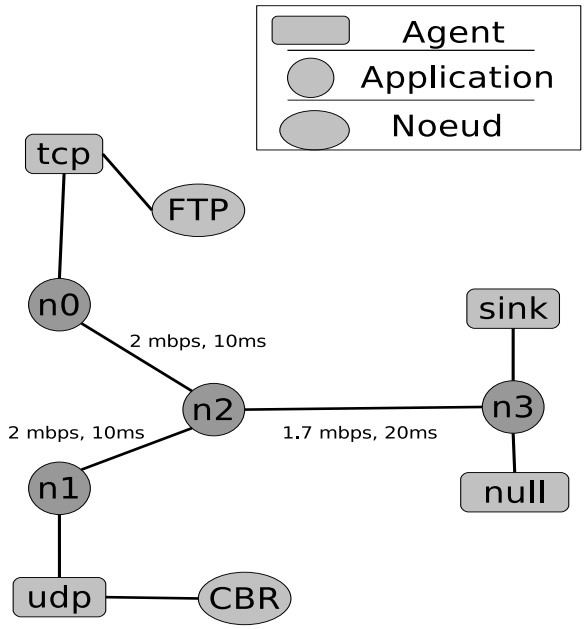

FIG. 3.4 – Exemple d'une topologie simple

Nous allons détailler ici la création d'une topologie simple. Celle-ci va comporter deux générateurs de traffic. Ce réseau consiste en 4 noeuds (n0, n1, n2, n3). Les liens double entre n0 et n2, n1 et n2 ont 2Mpbs de bande passante et 10 ms de délai. Le lien entre n2 et n3 a 1.7Mpbs de bande passante et 20ms de délai. Chaque noeud utilise une queue de type DropTail (Discipline de traitement des files d'attentes que l'on peut comparer à FIFO, de plus en plus remplacé par RED).

Un agent TCP et reliée à n0, et une connexion est établit vers le TCP Sink attaché au noeud n3. Un agent TCP Sink génère des acquittements de réponses. Un agent UDP est attaché au noeud n1 et relié à l'agent nul attaché au noeud n3. Un agent nul permet de libérer les paquets à la réception. Un générateur de traffic FTP et CBR est attaché respectivement aux agents TCP et UDP. La dernière étape de la simulation consiste à définir les temps de départ et de fin des générateur de traffic. Vous trouverez une copie de ce script en Annexe A.

3.2.3 Description d'un noeud

Comme le montre la figure 3.5, un noeud est composé d'une entrée. Cette entrée reçoit tous les messages à destination du noeud. Lors de la réception d'un message l'élément appellé **Addr Classifier** permet de trier les messages en plusieurs destinations. Si le paquet a pour destination ce noeud dans ce cas celui-ci est transféré au module **Port Classifier**. Ce dernier a pour rôle de transférer le paquet à l'agent (et ensuite parfois l'application) pour laquelle il est destiné. Dans le cas ou le paquet n'a pas pour destination notre noeud, il est alors redirigé vers le lien approprié en fonction de la table de routage construite en début de la simulation.

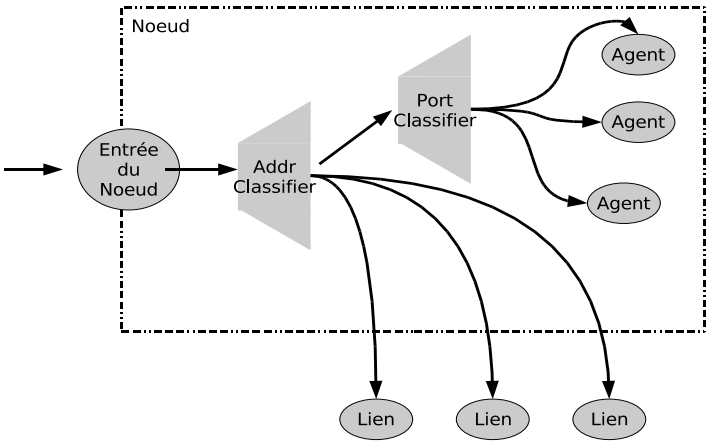

Fig. 3.5 – Vue d'un noeud NS-2

3.2.4 Classifier

Un classifier est une sorte de démultiplexeur. Il a pour objectif de retrouver la référence d'un objet à partir du contenu du paquet. Par exemple le Port Classifier permet de transmettre le paquet à l'agent approprié.

3.2.5 Description d'un lien

Un lien dans NS-2 est un objet qui relie deux noeuds. Pour que les paquets puissent aller et venir dans les deux sens, il est nécessaire de créer un lien d'un noeud vers un autre et inversement. Fort heureusement des routines permettent de créer facilement ce type de lien double. Un lien a donc pour propriétés :

– un délai de propagation : c'est à dire le temps de parcours du lien par un paquet (latence)
– une capacité : la bande passante
– un traitement de la file d'attente (DropTail par exemple).

Voici un exemple de création :

```
% lien entre n0 et n1
$ns duplex-link $n0 $n1 2Mb 10ms Droptail
```

3.2.6 Description de l'ordonnanceur (Calendrier)

Ce composant est appellé **Scheduler**, il permet d'exécuter des événements dans le futur. Lors de la création de la simulation, l'utilisateur définit un certains nombre d'événements. Pendant la simulation les objets créent eux-même de nouveaux événements. Tous les événements sont envoyés sur le scheduler qui les stocke dans un échéancier. Le scheduler a pour tâche d'organiser ces événements et de les exécuter. NS-2 se base sur ces événements et leurs génération.

3.2.7 Exemple d'un flux

Dans cette exemple nous allons détailler le principe de fonctionnement d'une simulation basique de NS-2. La simulation comporte seulement deux noeuds reliés par un lien double. Un agent et une application FTP sont présents sur le noeud n0 tandis qu'un agent de type TCPSink est présent sur le second noeud.

Au tout début de la simulation, l'application FTP envoi un paquet à destination du noeud n1. L'application envoi ce paquet à son agent associé. L'agent ne conaissant pas les liens extérieurs du réseau, envoi ce paquet à l'entrée du noeud n0. Celui-ci à l'aide du Addr Classifier identifie le lien approprié pour envoyer le paquet à la destination. En l'occurence c'est le lien n0-n1. Une fois à l'entrée du noeud n1, le Addr Classifier du noeud évalue le paquet et détermine qu'il lui est destiné. Il ne reste plus qu'à transmettre ce paquet au Port Classifier qui va transmettre ce paquet à l'agent TCPSink destinataire.

La deuxième partie de la simulation est la réponse à l'application FTP. En effet ce type d'application a besoin d'un acquittement. La réponse est donc émise par l'agent TCPSink qui l'envoi à sa propre entrée n'ayant pas conaissance du reste du réseau. L'entrée donne le paquet à l'Addr Classifier du noeud n1, qui détermine que le paquet n'est pas pour lui. Il l'envoi donc sur le bon lien qui s'avère ici être le lien n1-n0. Une fois parcouru le lien le paquet passe par le Addr Classifier du noeud n0 puit le Port Classifier de n0 pour terminer sa course à l'agent TCP. Le paquet s'arrête ici, puisque l'acquittement se situe au niveau de la couche TCP. Nous avons donc vu comment un paquet circule de noeud en noeud.

3.3 Utilisation du module Optical Burst Switching

Le module OBS que nous utilisons provient de The Regents of the University of California [3]. Ce module contient déjà un certain nombre de fonctionnalitées que nous allons détaillées ; il permet de simuler les réseau de type OBS.

[3] http ://wine.icu.ac.kr/ obsns/index.php

3.3.1 Description des fonctionnalitées

OBS-NS est écrit en c++ et Otcl. La partie en C++ contient deux classifier, une classe de gestion de statistique, une classe de scheduling, un agent, un démultiplexeur propre à l'agent et d'autres classes inhérente au fonctionnement comme une classe de bufferisation des bursts.

Le déclenchement de l'envoi d'un burst est hybride, si le timeout est écoulé alors le burst est envoyé ; si la taille maximum d'un burst est atteinte le burst est envoyé.

Si l'algorithme d'ordonnancement ne trouve pas de longueur d'onde disponible le burst est tout simplement jeté.

OBS-NS comporte son popre module de gestion de statistique appellé StatCollector. Ce module permet de stocker des informations importantes tel que le taux de perte des bursts ou encore le nombre de bits émis en TCP. De plus OBS-Ns utilise un système de qualité de service permettant de prioriser certains paquets que nous n'utiliseront pas et un système de temporisation de burts (FDL pour rappel).

3.3.2 Le schéma de signalisation utilisé

OBS-ns utilise une direction à un seul sens, c'est à dire que le module envoi un message BHP setup, attend un offset et envoi le Burst correspondant.

La réservation quand à elle est retardée avec remplissage des void. Le scheduler optimise donc l'utilisation des longueurs d'ondes dans le temps en fonction des disponibilitées.

La libération est implicite ; cela signifie que lorsque le burst est parti du core node, la longueur d'onde est automatiquement libéré.

3.3.3 core node

Le core node (figure 3.6) a pour rôle de commuter les paquets à travers le réseau. Il ne dispose que d'un mécanisme Add Classifier qui s'appelle **CoreClassifier**. Ce classifier utilise le scheduler **LaucScheduler** pour sélectionner la longueur d'onde adéquate. Les liens sont des **OBS-FiberLinkDelay** ; ils permettent de simuler des temps de bufferisation.

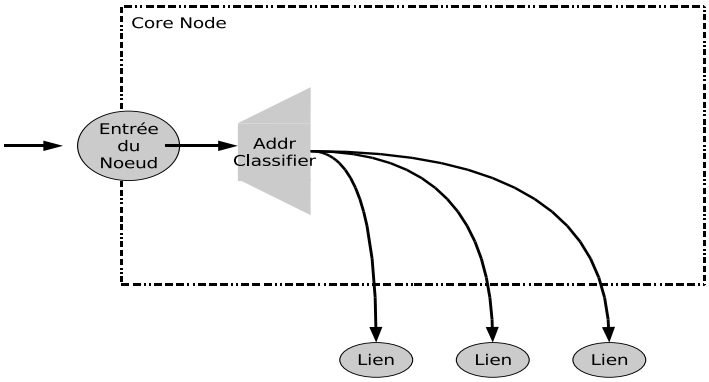

FIG. 3.6 – Un core node

3.3.4 edge node

Le edge node (figure 3.7) a deux rôles, le premier est de burstifier, c'est à dire assembler les paquets du monde exterieur en Burst. Le second est de réaliser la déburstification, c'est à dire le désassemblage. Au point de vue structure le edge node comporte un Addr Classifier appellé **EdgeClassifier**, un Port Classifier **OBSPortClassifier**. L'agent chargé de la gestion des bursts est **IPKTAgent**. Cet agent dispose d'autent d'instance de **BurstManager** qu'il a de destination. Cette classe permet de gérer les files d'attentes propre à l'agent OBS.

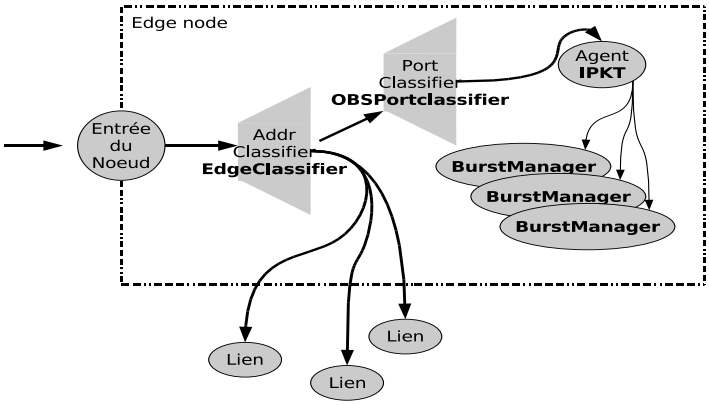

FIG. 3.7 – Un edge node

4 Schéma de Signalisation dans NS-2

Mon premier travail au sein du laboratoire a été de modifier le module OBS-NS existant afin de mettre en place les principaux schémas de signalisation. Cette modification doit être souple et flexible d'utilisation. A partir du script TCL, nous pourrons modifier chaque paramètre permettant de modifier le comportement des signaux. Ainsi nous pourrons paramétrer les protocoles directement depuis le script. Les valeurs par défaut doivent correspondre au résultat d'une simulation actuelle.

Les auteurs du livre [1] ont définis une hiérarchisation des caractéristiques des schémas de signalisation pour aboutir à une architecture commune.

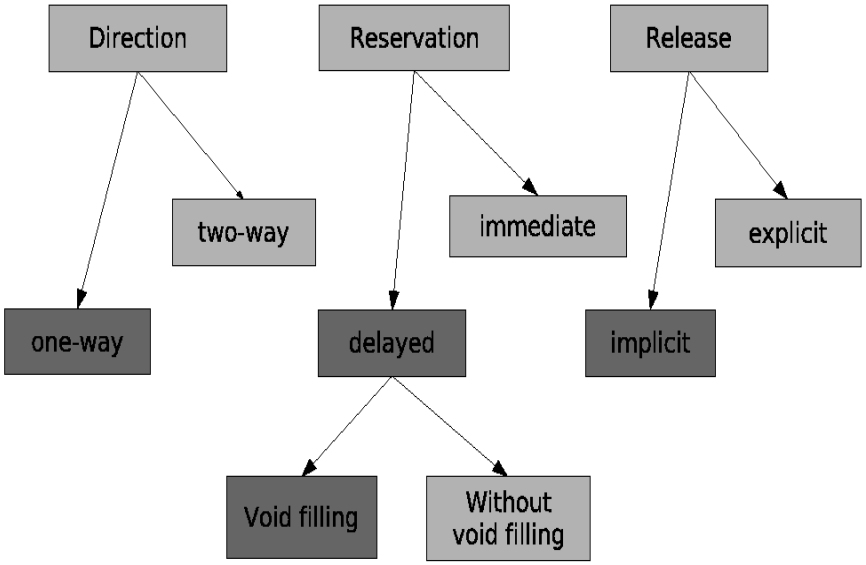

FIG. 4.1 – Caractéristiques des schémas de signalisation

La figure 4.1 permet de visualiser cette arbre de propriétés. Un schéma de signalisation se caractérise donc par trois propriétés (cf. section 2.2.4) :

1. La direction (Direction),
2. La réservation (Reservation),
3. La libération (Release).

Nous allons étudier dans une première partie, la structure logicielle du module OBS et justifier les choix de modifications apportés. Puis nous étudirons la mise en place d'une validation analytique ; qui comportera le contexte de la simulation ainsi que sa réalisation.

4.1 Implémentation

Les schémas de signalisation à implémenter sont **Just In Time (JIT)**, **Just Enough Time (JET)**, **Tell And Wait (TAW)**, **Tell And Go (TAG)**, et **HORIZON**. Nous allons détailler leurs compositions ci-dessous.

Le **TAW** utilise une direction à double sens, la réservation est du type retardé dans le temps et la libération explicite.

Le **TAG** possède une direction à un seul sens, la réservation est immédiate et la libération implicite.

Le **JET** est à direction en un seul sens, la réservation est immédiate et la libération implicite.

Le **JIT** a une réservation dans le temps avec remplissage des blancs, il utilise une direction à un seul sens tandis que la libération est explicite. Le JIT+ utilise les même caractéristiques mis à part que la libération est implicite.

Enfin **HORIZON** est similaire à JIT+ à la différence que la réservation est dans le temps sans remplissage des blancs.

Voici un tableau récapitulatif de ces différents protocoles :

Protocoles	Direction	Réservation	Libération
TAW	double sens	retardée	explicite
TAG	un seul sens	immédiate	implicite
JIT	un seul sens	retardée VF	explicite
JIT+	un seul sens	retardée VF	implicite
HORIZON	un seul sens	retardée	implicite
JET	un seul sens	immédiate	explicite

Le module OBS précédent utilisait une direction à un seul sens, une réservation retardée avec remplissage des blancs et libération implicite. Ce sont les caractéristiques de **JIT+**.

La demande initiale indiquait d'implémenter la capacité de spécifier les caractéristiques des protocoles de signalisation à notre convenance. Par exemple pour simuler **TAW** il suffirait de définir les paramètres suivants à partir d'un script TCL :

```
Direction = one-way
Reservation = delayted
Release = explicit
```

Nous allons développer chaque caractéristiques indépendamment de façon à pour pouvoir activée l'une ou l'autre en fonction des besoins de la simulation. La première étape va être de typer le **BHP**, puisque nous allons avoir besoin de trois messages setup différents (cf. section 2.2.4) : **BHP setup**, **BHP confirm**, **BHP release**. En fonction du type de BHP que l'agent recevra, le logiciel pourra agir en conséquence.

Un des caractéristiques d'un schéma de signalisation est la **direction**. Une direction peut être de deux types : **Direction one-way Direction two-way**. La différence se situe au niveau de la gestion de l'offset.

La **Direction one-way** consiste à envoyé un **BHP setup**, d'attendre un offset puis d'envoyer le burst. La **Direction two-way** requiert l'utilisation d'une second message BHP appellé **confirm** 2.5. Le principe d'implémentation est de créer un type de direction puis de l'affecter en début de simulation. Si la simulation est configuré en une seule direction, aucune modification n'est

effectuée. Dans le cas inverse lors de l'envoi d'un Burst, l'agent doit tout d'abord envoyé un **BHP setup** puis attendre le message de **BHP confirm** désiré (figure 4.2). Le message **BHP confirm** doit utiliser le même chemin que le message qui l'a généré. Actuellement la table de routage du module est statique ce qui signifie que le chemin d'émission de l'émetteur vers le destinataire est identique en sens inverse. Dans le cas d'une configuration à double sens d'autres cas particulier doit être gérer. Dans ce type de configuration nous pouvons rencontrer un problème lorsque le Core Node ne dispose pas des ressources pour que le burst puisse continuer de parcourir son chemin. Dans ce cas le Core Node jette le message concerné ; l'émetteur dispose toujours des données du burst. La solution de ce problème est l'utilisation d'un timer. Si un message **BHP confirm** n'est pas reçu dans un délai paramétré dans ce cas l'éxpéditeur envoi de nouveau un **BHP setup**. Et ceci de façon limité. On peut donc ajouté deux variables qui vont être :

- Le nombre de réémission d'un burst,
- La durée du timeout.

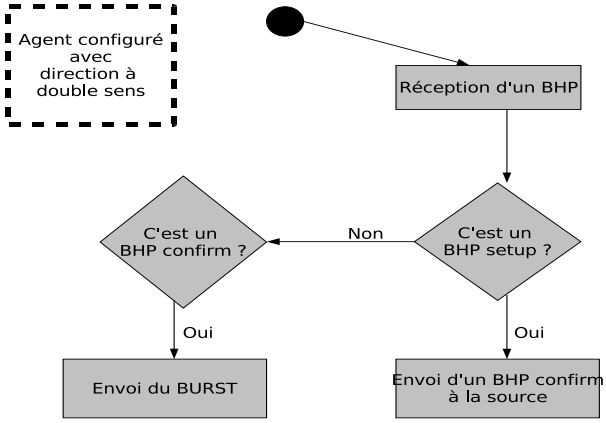

FIG. 4.2 – Algorithme direction à double sens

La **libération** des ressources doit être également activable depuis le script. Deux types de libérations sont à mettre en place, la libération implicite et la libération explicite. Le principe est de libérer explicitement chaque ressource allouée du réseau pour en disposer de nouveau. Cette libération s'effectue à l'aide du dernier type de message : **BHP release**. Un problème d'allocation peut subvenir dans le cas où le message **BHP release** ne parvient pas à destination. Avec l'utilisation d'un système de timeout, nous pouvons solder le problème en libérant cette ressource (longueur d'onde) au bout d'un délai fixé dans le script TCL.

Dans le module OBS actuel la **réservation** des ressources est réalisée par une classe C++ comme expliqué dans la section 3.2.3. Il y a autant d'instances que de ports ; chaque instance exploite les canaux de contrôle et les canaux de donnée. Le Module utilise l'algorythme de LAUC-VF (section 2.2.5). Nous utilisons ici l'héritage lors de la création des instances afin de préciser le type de calendrier utilisé. La figure 4.3 permet de visualiser cette structure. Chacune des classes qui héritent de OBSScheduler surchargent les méthodes schedControl et schedData. shedControl recherche un canal de contrôle libre. schedData recherche un canal de données libre. La figure 4.4 est une représentation graphique de l'algorithme de LAUC-VF. Lorsqu'un canal est recherché qu'il soit de contrôle ou de donnée, nous disposons de deux informations de temps, le temps T d'arrivé du burst et sa durée. Si le temps T d'arrivé est supérieur à StartTime et que sa différence avec SchedTime est supérieur à la durée, la longueur d'onde est sélectionnée. L'alorithme parcourt toutes les longueurs d'ondes afin de trouver le temps le plus court. Si aucun void n'est trouvé, dans ce cas il recherche des longueurs d'ondes disponible après UnScheduleTime.

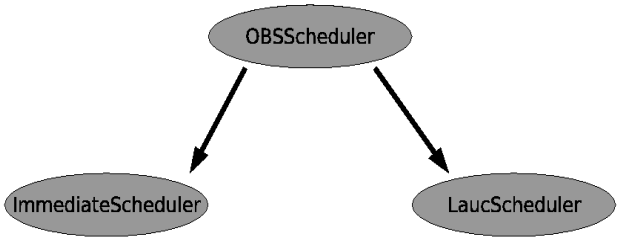

FIG. 4.3 – Héritage pour l'ordonnancement des longueurs d'ondes

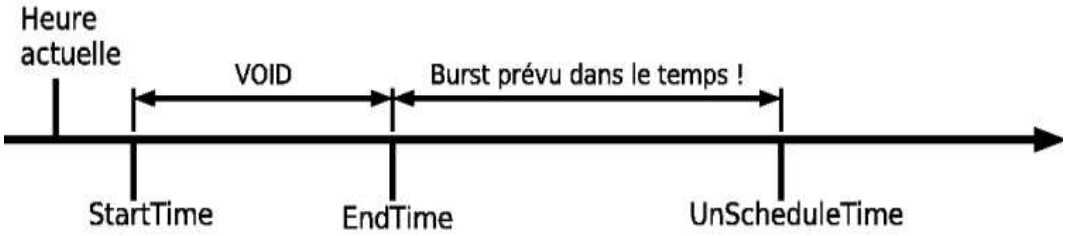

FIG. 4.4 – Planification d'une longueur d'onde avec l'algorithme LAUC-VF en C++

Lorsque que la longueur d'onde est sélectionné, StartTime récupère la valeur de UnScheduleTime, UnScheduleTime est affecté par le temps T de l'arrivé du burst additionné à sa durée, et EndTime est affecté par le temps T de l'arrivé du burst.

4.1.1 Test de l'implémentation

La simulation dont est issue la figure 4.8 est d'une durée de 10 secondes. Elle permet de mettre en évidence l'utilisation des différents BHP en fonction des schémas de signalisation. Voici une explication des thermes utilisés sur ce graphe :

- BHPDROP : messages BHP jetés
- BHPRCV : messages BHP reçues
- BHPSND : messages BHP envoyés
- BHPRLSSND : messages BHP de type libération
- BHPCFRSND : messages BHP de type confirmation

La figure 4.8 est le résultat d'une simulation mettant en évidence la gestion des BHP. Bien que les BHHPSEND sont constants hormis pour le TAW. Ce même protocole rejette également beaucoup de BHP. Le JIT émet deux fois plus de BHP avec la même fiabilité de réception.

La figure 4.6 affiche le résultat d'une simulation avec un trafic faible. Cette topologie est composée de deux edge node et d'un core node. Un agent est créé sur le premier edge node et envoi le traffic sur le second. Les trois noeuds sont reliés entre eux de tel sorte que le core node est au milieu. Une seule connexion est créé basé sur une taille de pacquet de 120 000 octets et d'un taux d'émission de paquets de 3000 bursts/seconde. Chaque lien dispose de 3 canaux de donnés et d'un canal de contrôle. Notons que mis à part le TAW le nombre de burst envoyés correspond exactement au nombre de burst reçus. Le TAW parait le moins fiable des protocoles. Remarquons également que JIT envoi deux fois plus de BHP que de burst ; ce qui le rend propice à la contention.

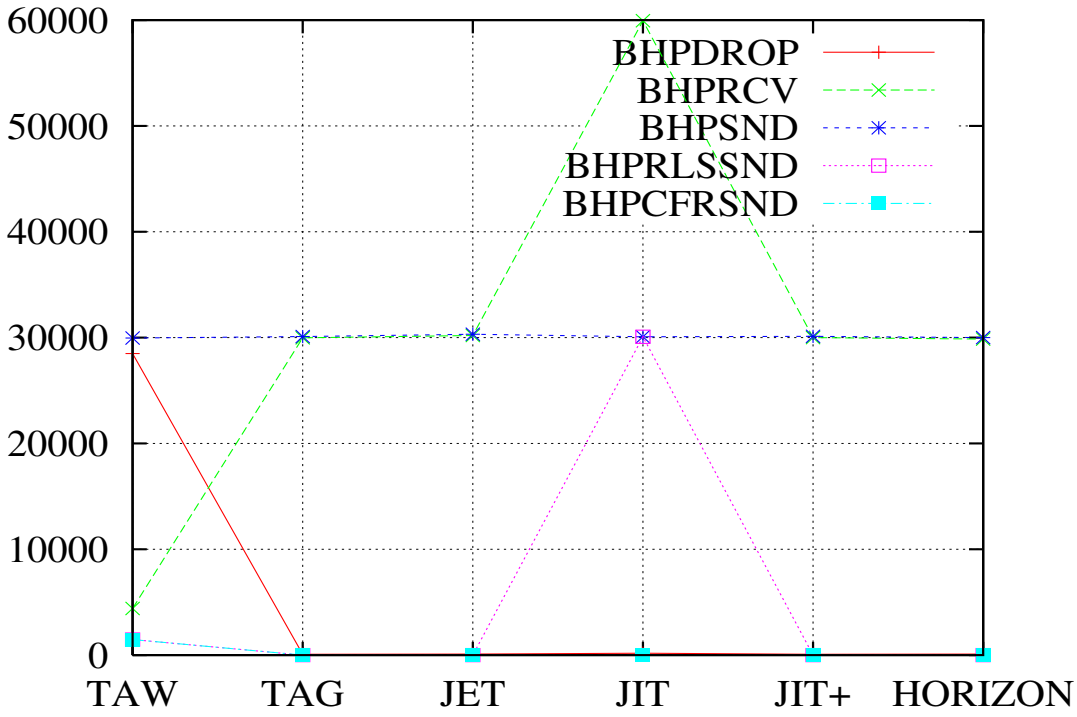

FIG. 4.5 – Affichage des BHP envoyés en fonction du type de signalisation

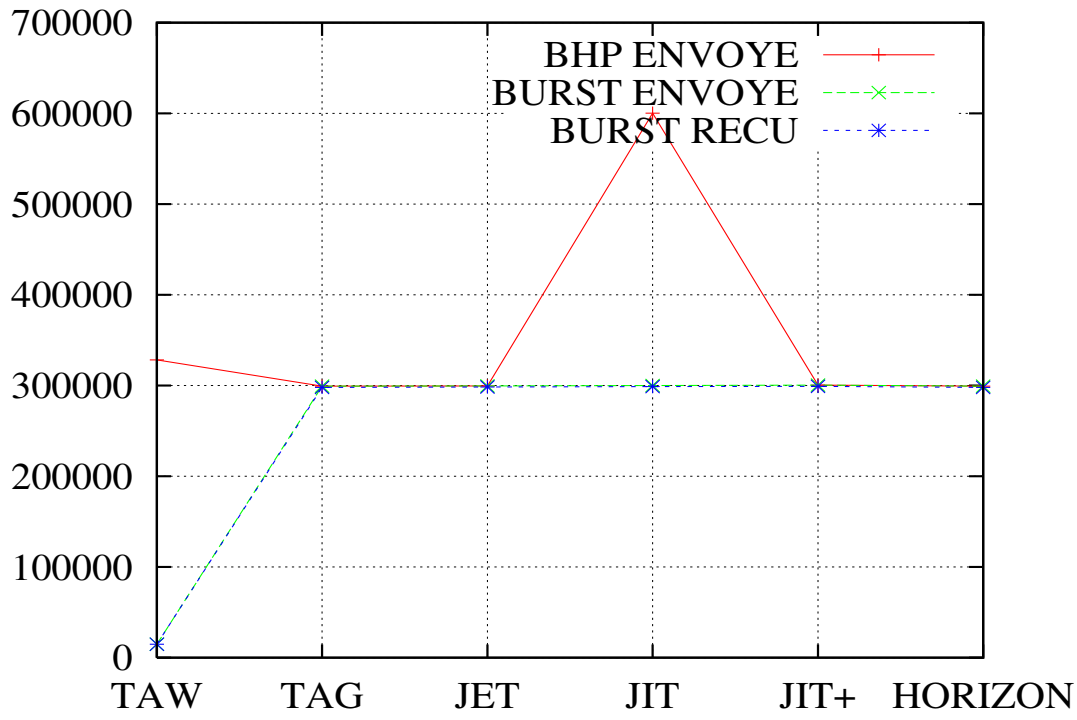

FIG. 4.6 – Statistiques réseau d'émmission de burst avec un traffic faible.

4.2 Validation analytique

Pour valider le développement logiciel de la signalisation dans le module OBS de NS-2, nous avons eu recours à un article qui propose de comparer JIT, JET et HORIZON [2]. Cette article

propose de calculer le taux probable de perte **Burst Lost Probability** (BLP), c'est à dire la probabilité de perte des bursts. La formule utilisée pour calculer cette valeur est Erlang. Erlang est une unité de mesure d'intensit du trafic. 1 Erlang correspond à l'occupation maximale sur une ligne. l'aide de la simulation nous allons calculer le taux de perte ou **Burst Lost Rate** (BLR) qui doit être similaire au BLP. Dans la première partie de ce chapitre nous allons présenter les caractéristiques du modèle à utiliser. Puis dans une seconde partie nous allons mettre en place une simulation en utilisant le module développé. La dernière partie présentera les résultats des simulations mais également une comparaison des résultats attendus. Nous conclurons par une discussion sur ces résultats.

4.2.1 Etude du modèle

L'objectif de ce modèle est d'étudier et de comparer les différents modèles de réservation de longueur d'ondes dans les réseaux OBS. L'article ne précisant pas si le JET utilisé utilise une libération de type explicite, il est décidé d'ajouter JET+ qui n'utilisera pas cette fonctionnalitée.

Nous considérons un réseau OBS qui comporte W+1 longueur d'ondes. Une seule longueur d'onde est utilisée pour la signalisation (les messages BHP) ; les autres sont réservées pour transporter les données. Voici la définition de trois paramètres important pour la simulation (figure 4.7) :

Toxc correspond à la somme de temps que prend un OXC pour configurer un chemin depuis une entrée vers une sortie. Physiquement ce temps correspond au temps utilisé pour la configuration des miroirs (l'ordre de temps est de 0.1 ms). Pour la simulation ce temps sera constant et définit en début de test.

Tsetup(X) correspond au temps de parcours du message de setup en fonction de X (JIT, JET, HORIZON), schéma de signalisation. Nous allons supposer que Tsetup(X) est une constante définit en début de simulation pour tous les bursts.

Toffset(X) correspond au temps d'offset de l'envoi des rafales en fonction du schéma de signalisation (JIT, JET, HORIZON). La valeur de cet offset dépend du schéma de signalisation et du nombre de noeud. Il peut également dépendre d'autres facteurs comme la qualité de service que nous n'allons pas utiliser ici. Supposons que **k** est le nombre de Core Node que le burst doit traverser pour aller de la source à la destination. L'article propose de calculer l'offset avec la formule suivante :

```
Toffset(X) = k * Tsetup(X) + Toxc
```

La simulation requiert également d'autres caractéristiques précises. Les messages BHP doivent suivre une génération de type processus de Poisson avec un taux de λ. Le processus de Poisson est une loi exponentiel sur des cas rares. La longueur d'un burst suit également cette loi de poisson avec pour paramètre la taille moyenne d'un burst. Enfin l'offset suit également cette loi avec comme paramètre la moyenne de Toffset.

Pour calculer le Burst Drop Probability, l'auteur a choisi d'utiliser la formule d'Erlang-B. Les deux paramètres de Erlang ici sont le nombre de longueur d'ondes transportant des données et la puissance du trafic. Puis pour chaque type de schéma de signalisation, un modèle mathématique est établit pour obtenir cette puissance de trafic. Pour plus de détail sur ces formules se référer à l'article.

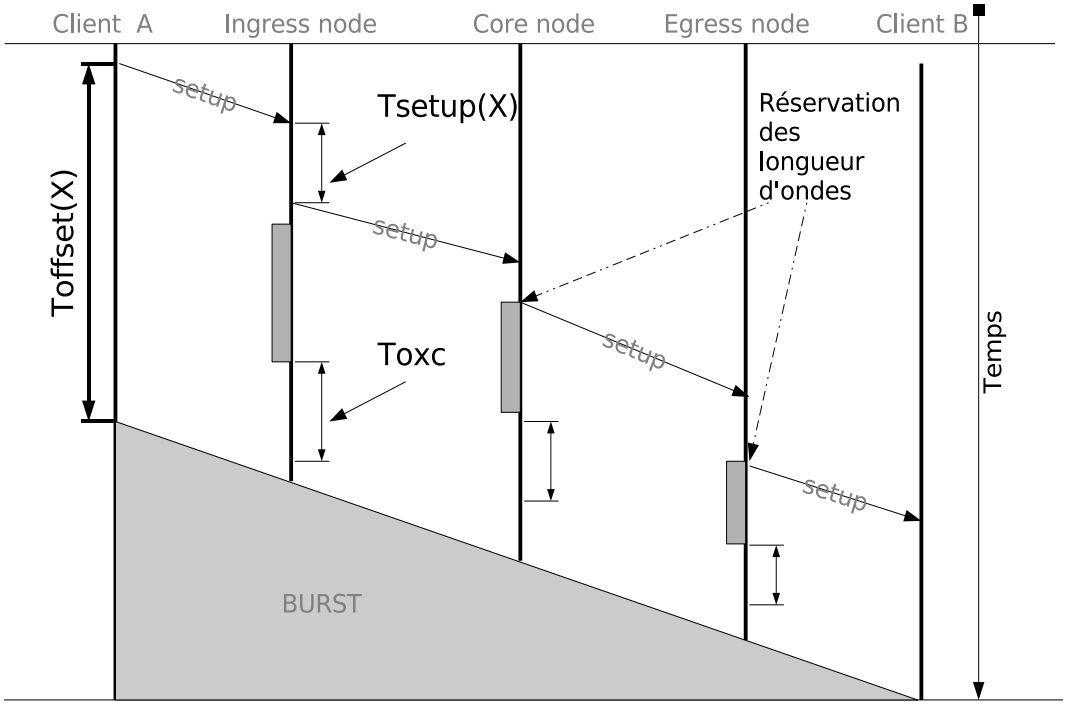

FIG. 4.7 – Présentation de Toxc, Tsetup(X) et Toffset(X)

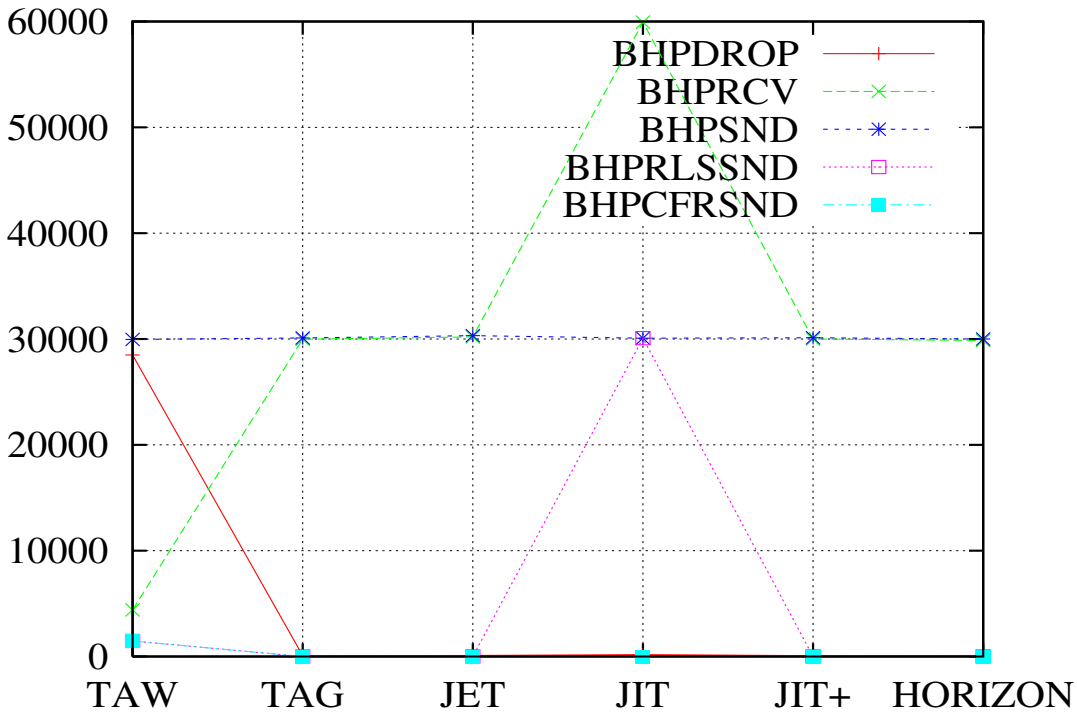

FIG. 4.8 – Affichage des BHP envoyés en fonction du type de signalisation

Discussion

La figure 4.9 nous permet de visuliser les résultats obtenus par les modèles mathématiques. Le premier graphe est réalisé à partir de valeurs qui correspondent aux réalitées matérielles actuelles.

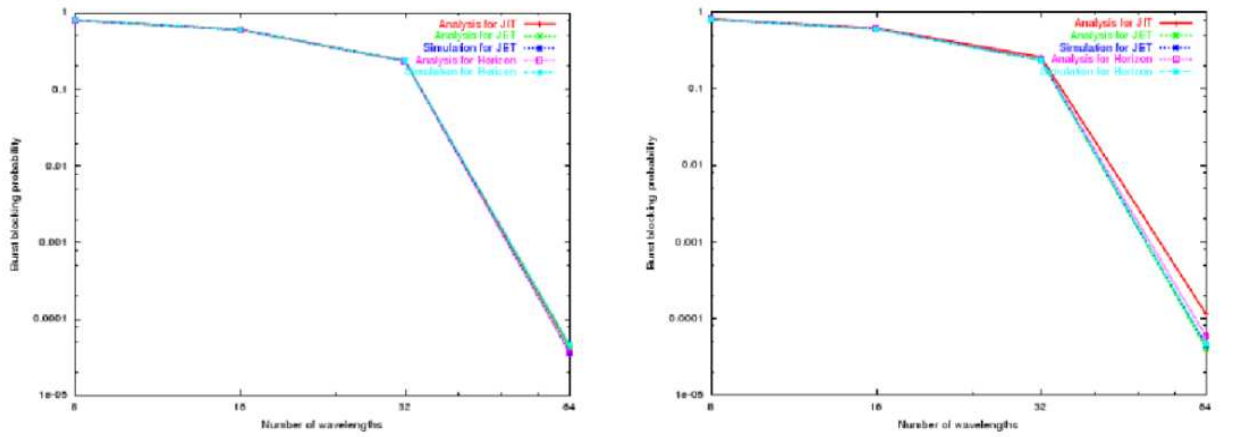

FIG. 4.9 – Résultats de l'article

Le second graphe est créé à partir de valeurs supposées des prochains équipements.

Ces deux graphes présentent la variation du Burst Lost Probability en fonction d'un nombre de longueur d'ondes différentes (8, 16, 32 et 64).

Nous pouvons observer que les courbes sont similaires que ce soit dans un ou l'autre des graphes, en dépit du fait que les constantes choisies sont très différentes. Les trois schémas de réservations sont identique mise à part lorsque le nombre de longueur d'ondes équivaut à 64 dans le scénario du future (Dans ce dernier cas JET et HORIZON ont un meilleur résultats que JIT).

4.2.2 La simulation

Pour respecter les prérequis de la simulation nous allons utiliser deux Edge Node et un Core Node. Un des deux Edge Node comportera un agent qui va générer du traffic en suivant le processus de Poisson. La figure 4.10 présente la représentation graphique de la topologie.

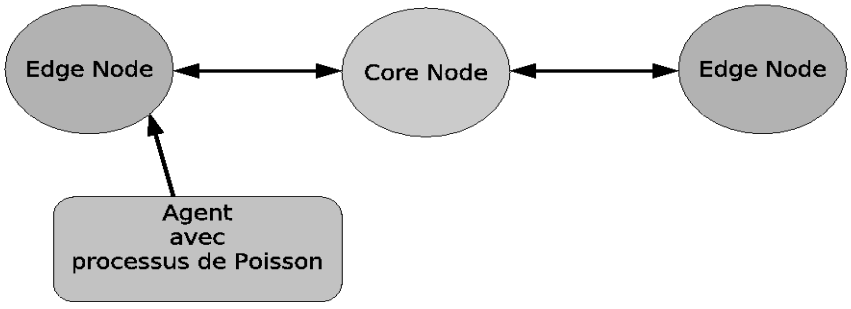

FIG. 4.10 – Topologie utilisé pour la simulation

L'annexe B contient la totalité du script réalisé.

Nous allons tout d'abord mettre en place le scénario 1 (Current scenarion) de la figure 4.9. Pour déterminer λ nous savons que λ/μ=32. Nous savons également que 1/μ=50ms c'est à dire 0.05s. Nous pouvons donc en déduire que λ = 32/0.05 soit 640. Pour déterminer la taille moyenne d'un burst nous savons que le temps de transmission = la taille d'un burst/capacité. Donc la taille moyenne = capacité/μ = capacité*1/μ. Si nous utilisons une capacité de 100000000 bit cela donne : 5000000. Une simulation équivaut à un résultat sur le graphe, il faut donc 16 simulations pour produire un graphe ce qui correspond au 4 signalisation (JIT, JIT+, JET et HORIZON) et 4 nombres de longueurs d'ondes (4, 8, 16 et 32). L'objectif étant de produire un graphique similaire à celui présent dans l'article, nous allons utiliser une script linux permettant d'automatiser ces tests :

```
for Delay in 20 100 200 500
        for Signaling in JIT+ JIT JET HORIZON
                for NumberOfWaveLength in 8 16 32 64
                        Run Simulation (Delay Signaling NumberOfWaveLength)
                done
        done
done
```

La figure 4.11 est le résultat de la simulation **Current Scenario**. Si nous comparons ce graphique avec le résultat de l'article, nous pouvons voir que la courbe est identique. Celle-ci est identique quelque soit le type de procoles mis en place.

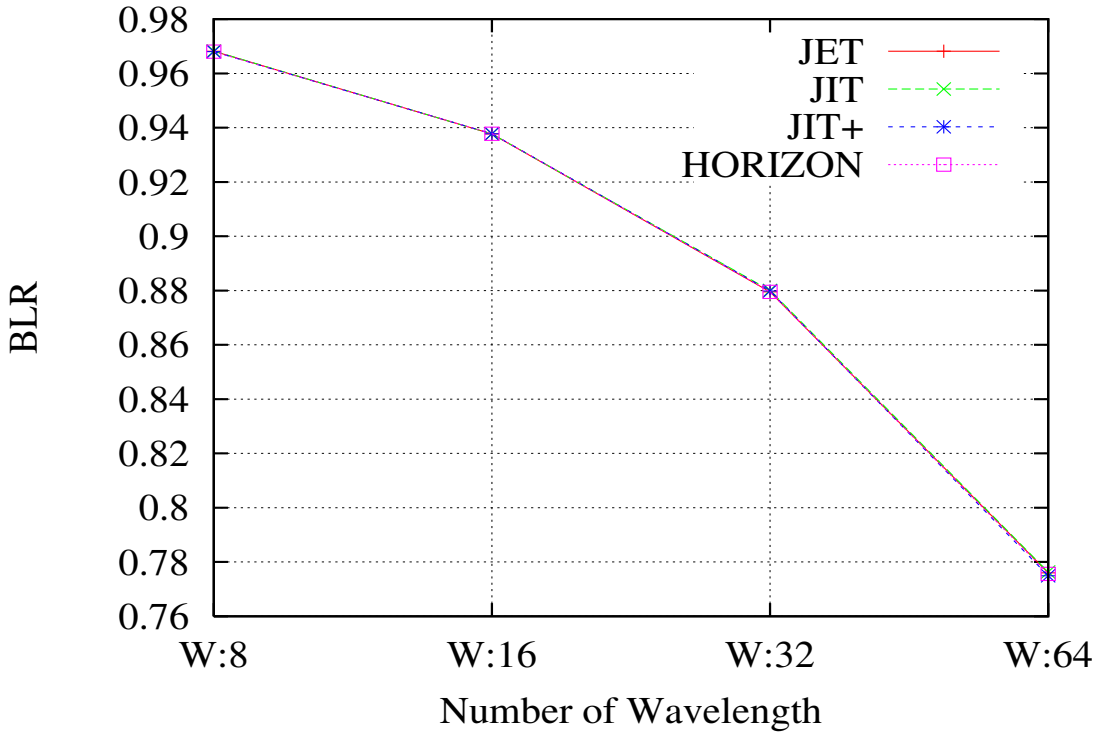

FIG. 4.11 – Résultat de la simulation Current Scenario

La figure 4.12 est le résultat de la simulation **Future Scenario**. Si nous comparons ce graphique au graphique Future Scenario de la figure 4.9, il apparait que la courbe est légérement similaire. Sans toutefois avoir le détachement pour la dernière valeur avec les 64 longueurs d'ondes. Sur le graphe de l'article, le protocole JIT est moins performant. Dans nos résultats tous les protocoles possèdent une courble similaire.

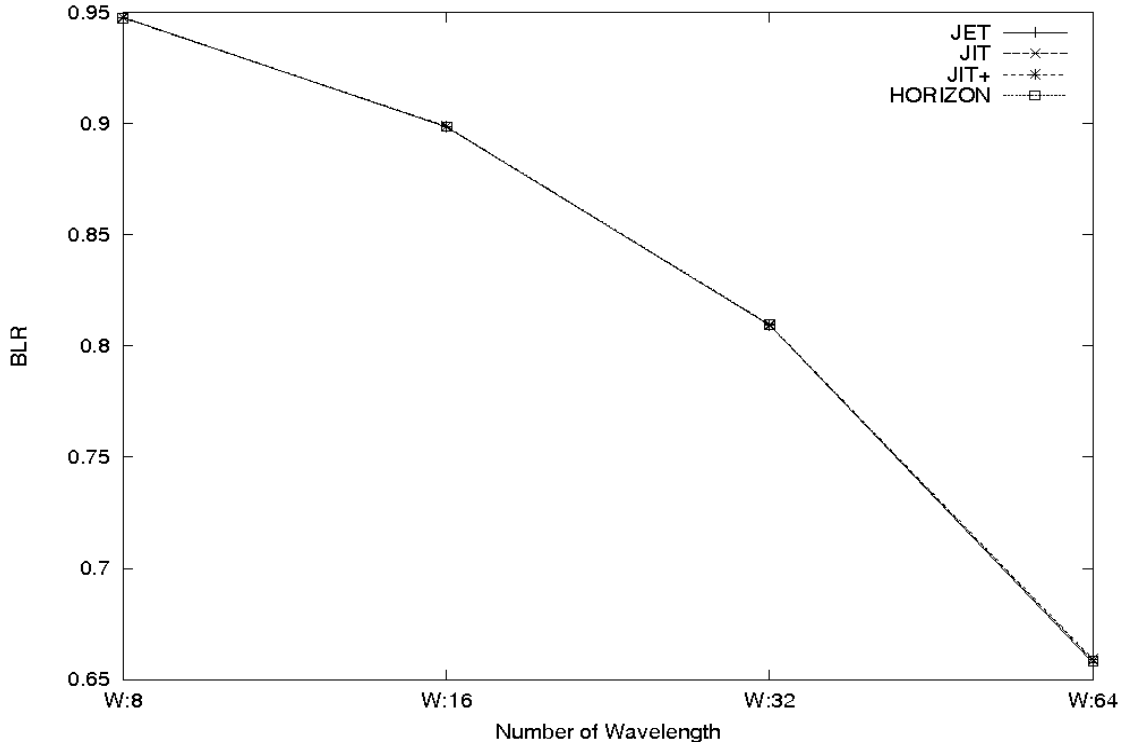

FIG. 4.12 – Résultat de la simulation Future Scenario

5 Editeur de topologie

L'éditeur de topologie constitue la seconde partie de mon travail. L'objectif est de réaliser un outil graphique permettant de simplifier la création de topologie pour NS2-OBS. A partir de cette interface l'utilisateur doit être capable de créer et de paramétrer chaque élément de la simulation.

Dans ce chapitre, nous allons tout d'abord étudier les fontionnalitées nécessaires du logiciel. Puis nous allons détailler les choix technologiques et d'écrire l'architecture du développement. Pour clore ce chapitre nous allons présenter l'outil obtenu.

5.1 Etude des fonctionnalitées

Ce logiciel s'inspire de nam (network animation) qui est l'outil actuel permettant de créer des toplogies pour NS. nam (figure 5.1) propose des outil de dessin des noeuds, des agents et des liens entre eux. Il dispose pas à l'heure actuelle de fonctionnalitées pour dessiner des réseaux OBS. C'est pourquoi ce nouvel outil baptisé **jnam4obs** (java nam pour OBS) est destiné exclusivement à la création de topologie NS2-OBS.

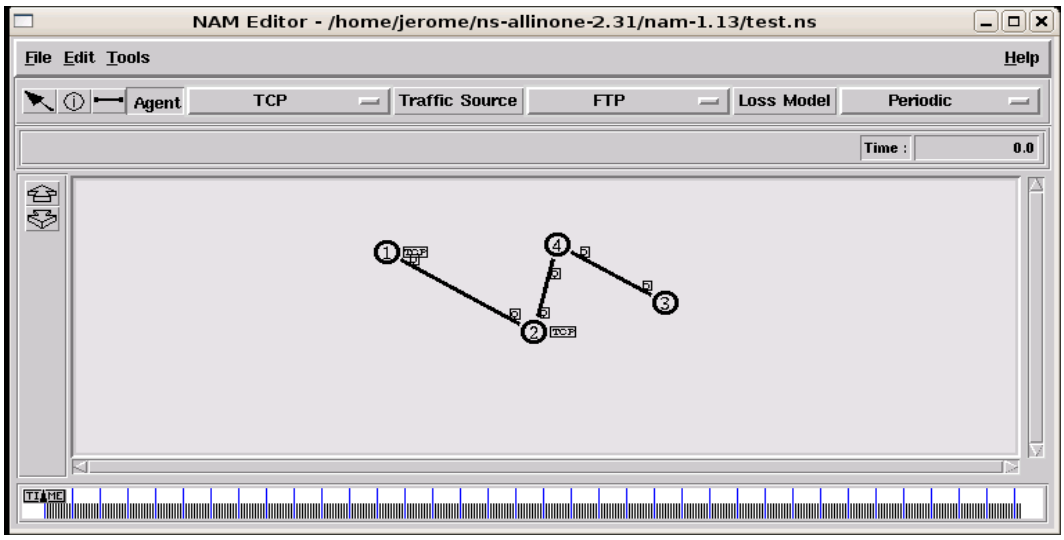

FIG. 5.1 – nam

jnam4obs doit comporter trois parties : la première est l'éditeur de topologie, la seconde un module de gestion des variables globales ; enfin la dernière partie concerne la gestion des statistiques. La figure 5.2 affiche le diagramme de fonctionnalité de jnam.

Fig. 5.2 – Diagramme de foncionnalité

5.1.1 Le module graphique

Une topologie NS2-OBS est structuré autour de **Core Node** et de **Edge Node**. Ces noeuds sont ensuite reliés entre eux.

Voici la liste exhaustive des composants d'une topologie NS2-OBS :
- Core Node : Un noeud de coeur
- Edge Node : Un noeud d'interface.
- Physical Link (PL) : Un lien physique entre deux noeuds de type duplex (dans les deux sens).
- Agent : Générateur de traffic.
- Software Link (SL) : Un lien entre deux agents.

Les cores nodes, edges nodes et agents doivent être représenter par un symbole graphique qui pourra être dèplacer ou supprimer. Les liens physiques seront représentès par une flêche. Les liens logiciels relient deux agents, l'information apparaitra sous la forme d'une pastille sur les agents.

Les cores nodes, edges nodes et agents ont une autre caractéristique commune, ils sont identifiés par une chaîne de caractère qui doit être éditable.

Pour simplifier l'affichage graphique afin de réaliser des réseaux complexes, nous allons utiliser un paramètre permettant de créer plusieurs instances sur un même élément graphique. Ce paramètre va s'appliquer aux edge node et aux agents.

Un agent est l'élément permettant de générer ou de recevoir le traffic du réseau. Les générateurs de traffic pour nos types de réseau peuvent être du FTP, POISSON ou PARETO. Une fois le type de générateur sélectionné il faut également spécifier la taille du paquet. Il est également nécessaire de spécifier l'agent cible du générateur ainsi que le temps T de lancement et d'arrêt du générateur.

Un lien physique permet de relier deux noeuds de type core ou edge. La configuration du lien comporte la bande passante, le délai, le nombre de canaux de contrôle et le nombre de canaux de donnés. L'implémentation du module dispose d'un mécanisme de temporisation Fiber Delay Link (FDL). Il sera donc interessant d'intégrer une possibilité de saisie d'une temporisation FDL.

Pour relier un agent à un core node, nous utiliserons également un système de flêche.

5.1.2 Le module variable globale

Les variables globales sont une partie importante d'un script TCL de simulation. En effet, il représente à la fois la flexibilité du système mais aussi la complexité de celui-ci. Nous pouvons déterminer de manière précise chaque paramètre du réseau OBS afin d'être le plus proche possible

de la réalité.

Chaque variable doit être modifiable depuis l'interface graphique. Un commentaire de description est nécessaire pour chacune d'elle afin d'aider l'utilisateur lors de modifications. Les variables seront regroupés dans des groupes afin de simplifier l'utilisation du logiciel.

Lorsque les propriétés d'un élément graphique possèdent pour valeur 0, c'est la valeur de la variable globale correspondante qui sera utilisée. Ainsi par exemple si le nombre de canaux de contrôle n'est pas spécifié pour un lien physique lors de la génération jnam4obs va alors appliquer la valeur de la variable globale correpondante : ncc (number of control channel).

5.1.3 Le module statistique

Ce module doit permettre de spécifier les statistiques qui doivent être généré pour les résultats de la simulation. Ces statistiques concernent principalement le nombre de paquets jetés, le nombre de paquets reçus ou encore le nombre d'octets généré par la simulation. La dernière version de la simulation permet de générer des statistiques entre deux noeuds. Il est donc nécessaire de pouvoir spécifier les noms des noeuds.

5.1.4 Génération du script TCL

Comme la figure 5.3 l'indique, un script TCL pour OBS s'instancie dans un ordre définit. La définition des variables globales consiste à paramétrer la simulation par l'affectation de temporisations ou la spécification de paramètres important. La création des noeuds consiste à l'instansciation des core et edge node du réseau. La création des noeuds consiste à la mise en place de liens physiques entre les différents noeuds. La table de routage est créer une seule fois au début du script puis elle est fixe pendant toute la durée de la simulation. L'étape suivante consiste à mettre en place les générateurs de traffic puis de lancer la simulation.

Fig. 5.3 – ordre des actions d'un script TCL

5.1.5 La notion de répertoire de travail

L'environnement d'exécution du script utilise un ensemble de sous script propre au module NS2-OBS de l'équipe. Cette version du module a la particularité de générer des statistiques en fonction de chaque noeud source vers chaque noeud de destination. Le répertoire d'exécution du script contient ces fichiers :

- controller.tcl : Script de lancement de la simulation
- stats.tcl : Contient quelques fonctions de statistiques

Le répertoire contient également un sous répertoire INCLUDING. Ce dernier contient les scripts suivants :

- myarguments.tcl : Gestion des arguments passer en paramètre
- initialization.tcl : Initialisation des variables globales
- helper.tcl : Affichage du menu d'aide
- config.tcl : Affiche la configuration du script
- common.tcl : Quelques fonctions TCL utile à la simulation

Le logiciel de génération du TCL doit générer les trois fichiers suivant :

- topology.tcl : Contient les fonctions de création/gestion de la topologie
- variables.tcl : Contient les variables globales
- StatsParameters.tcl : Contient les paramètres de génération des statistiques

5.1.6 Stockage des données

jnam4obs doit posséder une fontion de sauvegarde des topologies éditées. Cette sauvegarde doit enregistrer à la fois la topologie et ces différentes caractéristiques mais aussi l'emplacement des noeuds sur le graphe.

Il est également nécessaire de stocker la liste des variables globales ainsi que leurs valeurs respectives. Comme nous l'avons vu plus haut, ce type de sockage est propre au répertoire de travail et peut donc être stocké dans ce répertoire. Ce même répertoire doit également stocker la liste des statisiques utilisable et les statistiques sélectionnées.

5.1.7 Fonction de vérification du graphe

Une fonction de vérification de la cohérence du graphe doit être implémenté pour guider l'utilisateur lors de la réalisation d'une topologie. Cette fonction vérifira toutes sortes d'erreurs auquel l'utilisateur pourrait être confronté lors du lancement de la simulation. Voici la liste des erreurs détectables par cette fonction :

- Abscence de noeud
- Abscence d'agent
- Noeud orphelin
- Agent orphelin
- Nombre d'occurence d'un noeud incohérent avec le noeud correspondant
- Nombre d'agent incohérent avec l'agent relié

5.2 Choix des technologies

NS-2 est utilisé sous un environnement linux. Certaines membres de l'équipe utilse Cygwin pour utiliser NS-2 sous Windows. Le logiciel doit donc être compatible avec Unix et Windows. Le choix s'est donc porter sur Java [1]. Ce language en plus d'être multiplateforme dispose de nombreuses API répondant à nos besoins.

5.2.1 Module graphique

La partie principale de l'application est la création graphique de la topologie. L'ergonomie étant très importante, l'utilisation d'un module graphique complet est nécessaire. l'API JGraph présente ces caractéristiques [2]. Ce composant permet de créer des diagrammes, des graphes à état, ou toute sorte de graphe basé sur des principes de noeuds et de liens. De plus les entités peuvent prendre n'importe quelle forme afin de correspondre aux besoins du développeur. Cette API dispose de fonctions de sauvegarde qui ne seront pas utilisés puisque le logiciel a besoin de stocker d'autres informations propres à chaque entité.

5.2.2 Sauvegarde des données

Pour la gestion des sauvegardes de graphes ainsi que des sauvegardes de configuration le choix s'est porté sur l'utilisation de **Extensible Markup Language** (XML). L'API retenue pour cette tâche est JDom, car cette API propose une interface d'exploitation simple [3].

5.3 Diagramme de fonctionnalité

La figure 5.4 illustre les principales fonctions du logiciel. La structure mémoire de la topologie sera stockée dans un ensemble de classes pévu à cet effet.

La génération du TCL doit comporter les fonctions d'écritures des fonctions statistiques, et de script TCL.

Le module graphique doit comporter les fonctions nécessaire aux quatres éléments du schéma :

Dessin d'un graphe Un core node, edge node et agent

Vérification du graphe fonction d'analyse de la cohérence

Sauvegarde du graphe Enregistrement du graphe dans un fichier

Ouverture du graphe Ouverture du fichier et affichage sur le graphe

Une structure mémoire spécifique doit être créer pour accueillir la topologie :

Agent Classe qui gère agent.

Node Classe qui gère un noeud.

SoftwareLink Classe qui gère un lien logiciel.

[1] http ://www.java.com/fr/
[2] http ://www.jgraph.com/
[3] http ://www.jdom.org/

PhysicalLink Classe qui gère un lien physique.

NSTopology Cette classe stocke une topologie (à l'aide de vecteurs) et possède les fonctions d'exploitation de celle-ci.

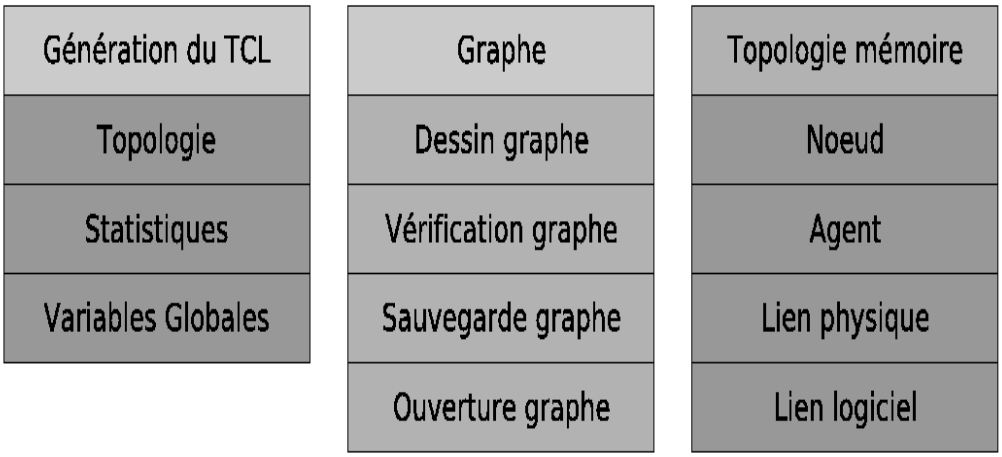

FIG. 5.4 – Diagramme de fonctions de JNAM

5.4 L'application

jnam4obs est écrit en java. L'interface utilise les composants de Swing. Vous pouvez observer l'interface présente à l'annexe C.

Au lancement de l'application le logiciel affiche une fenêtre permettant de spécifier l'environnement de travail de jnam. Nous avons fait le choix de séparer le répertoire de travail pour pouvoir spécifier les paramètres de simulation pour un répertoire donné.

Le logiciel se compose d'un menu global, d'une barre d'outil contenant des boutons de création, ouverture et sauvegarde d'un fichier jnam. Il contient également des fonctions de vérification de graphe, de modifications des variables globales, des statistiques. Il y a également un bouton pour générer le script TCL et en dernier lieu un bouton lançant la simulation.

Un système d'onglet global à l'application permet l'exploitation de 5 fenêtres. La première fenêtre concerne la création du graphe, la seconde l'édition des variables, la sélection des statistiques, une fenêtre permettant de visualiser du TCL et la dernière une fenêtre permettant de visualiser la progression de la simulation.

Quelques fonctions sont exécutées en utilisant un thread. C'est le cas pour l'ouverture/enregistrement d'un fichier, le chargement du workspace, la génération du TCL, le lancement de la simulation et la vérification du graphe. Une barre de progression est utilisée pour voir l'état d'avancement des différents threads (figure 5.5).

FIG. 5.5 – Exemple de la barre de progession : exécution d'une simulation

La figure 5.6 représente une topologie simple. Elle est composée de deux core node, de deux edge node, d'un agent générateur de traffic et d'un autre agent pour donner les réponses au premier agent. Vous pouvez remarquer que les liens physiques sont représentés par des traits et des flêches. Les liens entre agent et edge node utilisent des traits et des losanges. Les liens logiciels entre agent sont représentés par des icones de couleurs aléatoires.

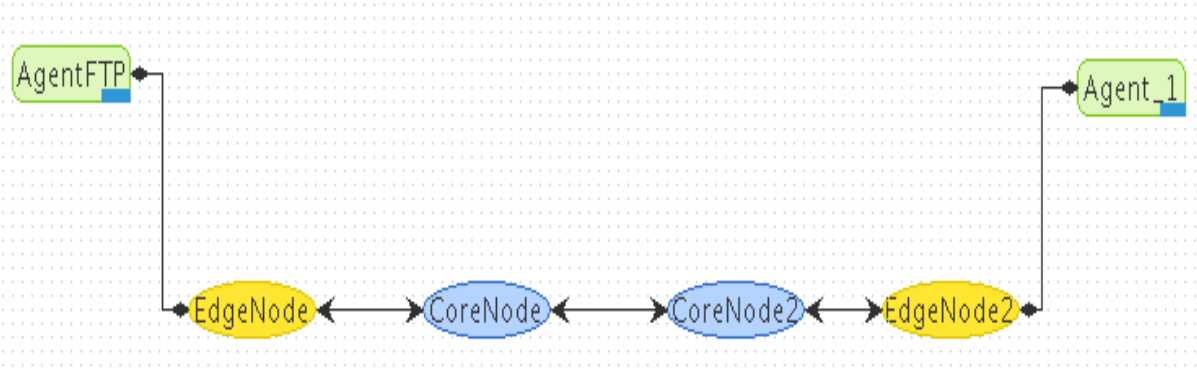

Fig. 5.6 – Exemple d'une topologie JNAM

Vous trouverez en Annexe E la topologie NSFNET (National Science Foundation NETwork) créé avec jnam4obs. NSFNET est le réseau américain de la recherche en science.

5.5 Conclusion

Ce travail a pour objectif de réaliser plus facilement des topologies complexes pour le simulateur de réseau NS. Bien que les objectifs du travail ont été clairement exprimés par l'équipe un gros travail a éeté effectué pour répondre aux besoins. En effet les demandes d'évolution du logiciel ont été spécifier au fur et à mesure de la création du logiciel ce qui a entrainé des temps de développement beaucoup plus important que prévu initialement.

6 Conclusion

Ce stage m'a permis d'appréhender une nouvelle technologie : Optical Burst Switching. Cette dernière, bien que complexe au premier abord, offre des perspectives très intéressantes pour le développement de bande passante importante. J'ai également eu l'opportunité de découvrir le simulateur de réseau NS qui offre une solution souple et extensible dans le domaine de la simulation réseau. Tant que des mémoires pour fibre optique n'existeront pas, OBS sera la solution a utilisé. Je pense que l'affinement de cette technologie réside dans l'adaptabilité dynamique des paramètres en fonction du type de réseau matériel et des données qui y transitent.

Ce projet m'a permis de découvrir plusieurs aspects du monde de la recherche. L'un des aspects marquant est que le chercheur ne peut pas perdre de temps à mettre sur papier ses théories, ou du moins tant qu'il n'est pas certain du résultat. C'est une démarche très éloignée du monde du travail, où tout doit être pensé avant de commencer le développement.

D'un point de vue humain, ce stage m'a permis de découvrir un nouveau pays : le Quebec. J'ai eu la chance de découvrir un autre continent avec ses coutumes, son climat et d'échanger avec des personnes de toutes origines.

Bibliographie

[1] Jason P. Jue and Vinod M. Vokkarane. *Optical Burst Switched Networks*. ISBN : 0387237569.

[2] Jint Teng and George N. Rouskas. *A Comparison of the JIT, JET, and HORIZON Wavelength Reservation Schemes on A Single OBS Node*

A Topologie Simple en TCL

```
#Create a simulator object
set ns [new Simulator]

#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red

#Open the NAM trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Define a 'finish' procedure
proc finish {} {
        global ns nf
        $ns flush-trace
        #Close the NAM trace file
        close $nf
        #Execute NAM on the trace file
        exec nam out.nam &
        exit 0
}

#Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail

#Set Queue Size of link (n2-n3) to 10
$ns queue-limit $n2 $n3 10

#Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
```

```
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right

#Monitor the queue for link (n2-n3). (for NAM)
$ns duplex-link-op $n2 $n3 queuePos 0.5


#Setup a TCP connection
set tcp [new Agent/TCP]
$tcp set class_ 2
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1

#Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP


#Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n3 $null
$ns connect $udp $null
$udp set fid_ 2

#Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 1mb
$cbr set random_ false


#Schedule events for the CBR and FTP agents
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr stop"

#Detach tcp and sink agents (not really necessary)
$ns at 4.5 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n3 $sink"
```

```
#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"

#Print CBR packet size and interval
puts "CBR packet size = [$cbr set packet_size_]"
puts "CBR interval = [$cbr set interval_]"

#Run the simulation
$ns run

}
```

B Script de la simulation

```
source ../lib/ns-obs-lib.tcl
source ../lib/ns-obs-defaults.tcl
source ../lib/ns-optic-link.tcl

#=================================================================
# ARTICLE PARAMETERS
#=================================================================

# amount of time it takes the OXC to configure is switch fabric to se up a
# connection form an input port to an output port
set Toxc 0.01

# amount of time it takes a OBS node to process the setup message under reservation scheme
set TsetupJIT  0.0000125
set TsetupJET  0.000050
set TsetupHORIZON  0.000025
set Tsetup $TsetupJIT

# the number of OBS nodes in the path of a burst from source to destination
set k 1

# setup message Poisson processus rate
set lambda 3000

# avarage of burst length in ms
set AverageBurstLengthMs 0.05

# total bandwidth/channel (1mb = 1000000) core-core
set bwpc 100000000000



#################################################################
# Controller Parameters
#################################################################
#COP Choose the Controller Output Parameter (0 REFBLN,1 REFBLRTOT,2 REFBLRAC,3 REFBLRNT,4
set COP 0
#Choose the Tuning Parameter (TPBR:0Burtification Rate,1:TPOFFSET)
```

```
set TP 0
#ControllerID could take 0 or 1: 0 means controller OFF and 1 means ON
set ControllerID 0
#if ControllerK_ =0 and Controller ON Admission control is through steps else the controll
set ControllerK_ 0.5
# Intial value for the TuningParam_ (parameter to be tuned by the controller)
set TuningParam_ 3000.0
#statisticsTime is the time interval sampling/occurs
set statisticsTime 0.1
#stepTime is the  time  that the controller takes an action afteer (when the controller up
set stepTime 0.1
# Reference for the controlled ouput parameter
set Ref_ 0.3
# Max for the admission controller : the BLRac must not exceed this max(must be 1.0 if not
set MaxBurstLossRatioAC_ 1.0
#if ControllerK_ =0 Admission control using steps,TPStep is the length of the step in the
set TPStep 500
# maximum and minimum value of the tuning parameter
set MaxTuningParam 5500
set MinTuningParam 0
# sets the normalized factor of the TP to a big value to fill the gap between the burstifi
set normalizedFactor 5000
# sets the normalized TP to an initial value of zero (not used)
set normalizedTP 1
# ControllerFilter =0 if we don't consider the average iAVGactualBurstLossRatio_
# AVGMeasuredCOP=(1-ControllerFilter)*MeasuredCOP+ControllerFilter*AVGMeasuredCOP
set ControllerFilter 0.8

#====================================================================#

# BurstManager offsettime $AverageToffset
# timeout typically 0.1,0.2,.trace-all..1.0s
set timeout 0.1
# v6 uses 1 - 70 mu s for offset time
# must take into account the number of nodes
set offset 0.1
# duration of the simulation in seconds
set duration 1

# delay of link
set delay 5.0
# debug messages are not shown (1 if shown)
set debugOK_ 0
# 0 if no traces
set tr 0

# burst length
set maxburstsize 120000
```

```
# Poisson or parameter Parameters (average burstification rate)
set lambda 3000.0

set load 1
source args.tcl

# offset value of a burst under reservation scheme X (JIT, JET, HORIZON)
set AverageToffset [expr ($k * $Tsetup) + $Toxc]
# Holds packetSize
set AveragePacketSize [expr $bwpc*$AverageBurstLengthMs]
set AveragePacketSize [expr $AveragePacketSize/8]

# node trace files
set tracefile "tracefile.tr"
set ndtrace "nodetrace.tr"

#===================================================================#
# total number of channels per link
set maxch [expr $W + 1 ]
#===================================================================#
source INCLUDINGS/initialization.tcl

#===================================================================#
source INCLUDINGS/common.tcl
source Info.tcl

source config.tcl

Config_SaveContext
#Config_Display

proc finish {} {
global ns nf sc tf ndf nam Configfile duration burstsize navailcc Capacity ncc
global verybasic_edge_count SignalingScheme W load
$ns flush-trace
$ns flush-nodetrace
close $nf
close $tf
close $ndf
#$sc display-sim-list
set burst [$sc get-stat BURSTSND]
puts "Burst sended : $burst"
set burstdrop [$sc get-stat BURSTDROP]
puts "Burst dropped : $burstdrop"
set burstrcv [$sc get-stat BURSTRCV]
puts "Burst received : $burstrcv"
set udpsended [$sc get-stat UDPBYTESSND]
```

```
set Info [open Info.txt a]
puts $Info "- $SignalingScheme W:$W BS:$burst BD:$burstdrop UDPBS:$udpsended\n"
flush $Info
close $Info

set pathfile "output/$SignalingScheme.txt"
set data [open $pathfile a]
set blr [expr $burstdrop/$burst]
set xindice 1

if { $W == 8 } {
set xindice 1
}
if { $W == 16 } {
set xindice 2
}
if { $W == 32 } {
set xindice 3
}
if { $W == 64 } {
set xindice 4
}

puts $data "$xindice\t$blr\t$burst\t$burstdrop"
close $data
putsdebug "Simulation complete"
exit 0
}

proc MySendPoissonpacket { udp InterArrivalTime pktSize src dest VariablePacketSize Variab
global PARETO ns NumberOfBurstSended NumberOfBurstToSend AveragePacketSize

# change the offset
set offset [$VariableOffsetSize value]
set offset [expr $offset*1]
Agent/IPKT set iOffsettime_ $offset

# change the packet size
set ps [$VariablePacketSize value]
set ps [expr $ps*1]
$udp send $ps

set time [$ns now]
set inter [$InterArrivalTime value]
if { $inter <= 0 } {
set inter [expr -1.0*$inter]
}
$ns at [expr $time + $inter ] "MySendPoissonpacket $udp $InterArrivalTime $pktSize $src $d
```

```
}

#=============================================================================================
# create a poisson connection with offset length and burstsize length
Simulator instproc My_create_Poisson_connection { lambda pktSize udp null src dest start0

upvar 1 $udp udpr
      upvar 1 $null nullr
      upvar 1 $src srcr
      upvar 1 $dest destr
upvar 1 $InterArrivalTime  InterArrivalTimer
upvar 1 $VariablePacketSize  VariablePacketSizer
upvar 1 $VariableOffsetSize VariableOffsetSizer
global run Toffset AveragePacketSize AverageToffset
 # seed the default RNG
global defaultRNG
$defaultRNG seed $sd

set rng2 [new RNG]
set rng3 [new RNG]
set rng4 [new RNG]
 # create the RNGs and set them to the correct substream
set run [expr $run+1]
for {set j 1} {$j < $run} {incr j} {
$rng2 next-substream
}
set run [expr $run+1]
for {set j 1} {$j < $run} {incr j} {
$rng3 next-substream
}
set run [expr $run+1]
for {set j 1} {$j < $run} {incr j} {
$rng4 next-substream
}

# generate random interarrival times and packet sizes
set InterArrivalTimer [new RandomVariable/Exponential]
$InterArrivalTimer set avg_ [expr 1/$lambda]
#[expr 1.0/$lambda]
$InterArrivalTimer use-rng $rng2

# generate random packet size
set VariablePacketSizer [new RandomVariable/Exponential]
$VariablePacketSizer set avg_ $AveragePacketSize
$VariablePacketSizer use-rng $rng3

# generate random offset size
set VariableOffsetSizer [new RandomVariable/Exponential]
```

```
$VariableOffsetSizer set avg_ $AverageToffset
$VariableOffsetSizer use-rng $rng4

      set udpr [ new Agent/UDP]
      $self attach-agent $srcr $udpr
      $udpr set packetSize_ $pktSize
      set nullr [ new Agent/Null ]
      $self attach-agent $destr $nullr
      $self connect $udpr $nullr

      #$self at $stop0 "$selfsimr stop"
 putsdebug "Poisson traffic stream $udpr between $src = $srcr and $dest = $destr created"
 #$self at $start0 "$self sendPoissonpacket $udpr $InterArrivalTimer $pktSize $src $dest"
 $self at $start0 "MySendPoissonpacket $udpr $InterArrivalTimer $pktSize $src $dest $Varia
}

puts "========================================="
puts "Creating  topology ..."
set AgentE1 1
set AgentE2 1
set E1 [$ns create-edge-node 2 AgentE1]
set E2 [$ns create-edge-node 2 AgentE2]
set C1 [$ns create-core-node 1]
$ns createDuplexFiberLink $E1 $C1  $bwpc $delay 1 $W $maxch
$ns createDuplexFiberLink $C1 $E2  $bwpc $delay 1 $W $maxch

# building table
$ns build-routing-table

puts "========================================="
puts "Creating connections  ..."
set k 0
while {$k < $load} {
        $ns My_create_Poisson_connection $lambda $AveragePacketSize udp0($k) null0($k) E1
incr k
}

puts "========================================="
puts "Running  ..."

$ns at [expr $duration ]  "finish"
# start ns
$ns run
```

C Impression écran de JNAM

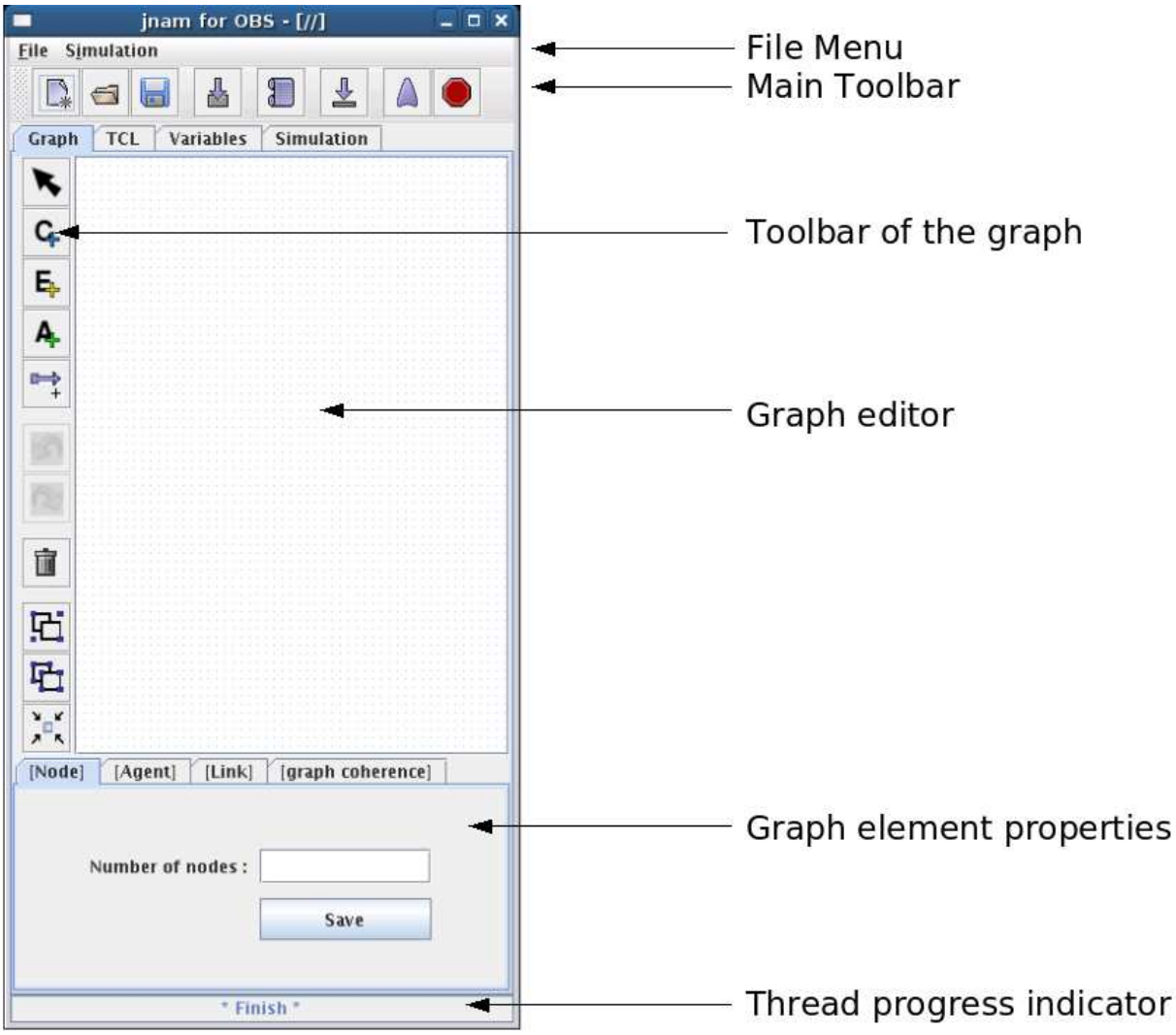

FIG. C.1 – JNAM Interface

D Méthode de configuration du schéma de signalisation

```
# set direction to agent
#       0 : one-way     ** default value **
#       1 : two-way
#Agent/IPKT set direction_ 0
# type of reservation
#       0 : immediate
#       1 : delayted (LAUC)
#       2 : delayted void filling (LAUC-VF)     ** default value **
#Classifier/BaseClassifier set reservation_ 2
# 0 = implicit release
# 1 = explicit release
#Agent/IPKT set release_ 0
# 0 = implicit release  ** default value **
# 1 = explicit release
#Classifier/BaseClassifier set release_ 0
Simulator instproc set-configuration { {typeconf ""} } {
        if { $typeconf == "TAW" } {
                Agent/IPKT set direction_ 1
                Classifier/BaseClassifier set reservation_ 1
                Agent/IPKT set release_ 1
                Classifier/BaseClassifier set release_ 1
        }
        if { $typeconf == "TAG" } {
                Agent/IPKT set direction_ 0
                Classifier/BaseClassifier set reservation_ 0
                Agent/IPKT set release_ 0
                Classifier/BaseClassifier set release_ 0
        }
        if { $typeconf == "JET" } {
                Agent/IPKT set direction_ 0
                Classifier/BaseClassifier set reservation_ 0
                Agent/IPKT set release_ 0
                Classifier/BaseClassifier set release_ 0
        }
        if { $typeconf == "JIT" } {
                Agent/IPKT set direction_ 0
```

```
                Classifier/BaseClassifier set reservation_ 2
                Agent/IPKT set release_ 1
                Classifier/BaseClassifier set release_ 1
        }
        if { $typeconf == "JIT+" } {
                Agent/IPKT set direction_ 0
                Classifier/BaseClassifier set reservation_ 2
                Agent/IPKT set release_ 0
                Classifier/BaseClassifier set release_ 0
        }
        if { $typeconf == "HORIZON" } {
                Agent/IPKT set direction_ 0
                Classifier/BaseClassifier set reservation_ 2
                Agent/IPKT set release_ 0
                Classifier/BaseClassifier set release_ 0
        }
}
```

E Topologie NSFNET

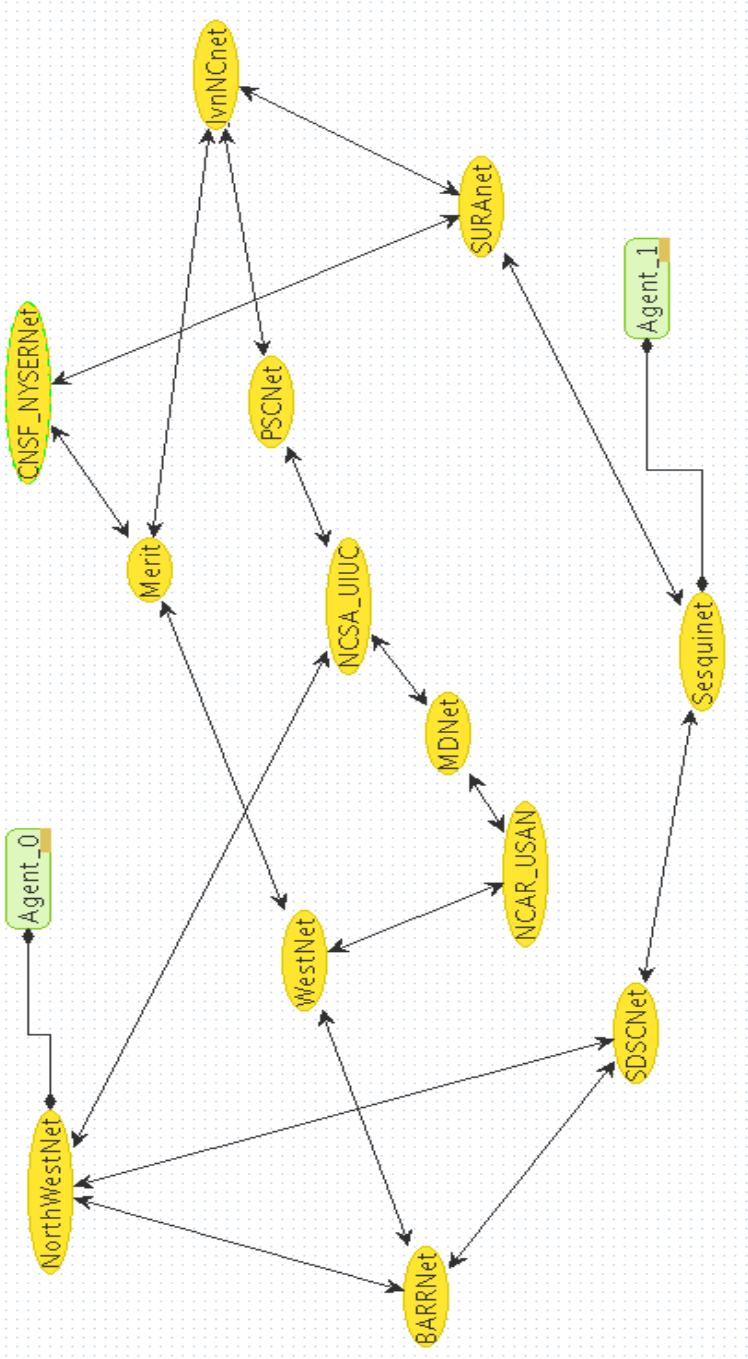

Fig. E.1 – Topologie NSFNET