UPMC - Paris VI
Master Informatique

Université de la Réunion
Master STIC

Introduction à NS

Modèle: Version 1.1

P. Anelli - LIP6

jeudi, octobre 12, 2006

Présentation

1. Motivations

Dans [1], les auteurs donnent les principales raisons qui rendent la simulation de l'Internet difficile. Elles peuvent se résumer par la grande hétérogénéité et l'évolution rapide. L'hétérogénéité commence avec les liens, comprend les protocoles et finit avec les trafics d'applications diverses et variées. L'Internet évolue rapidement en terme de taille mais également en terme d'architecture et de topologie. C'est devant ce constat que des chercheurs ont essayé d'élever l'état de l'art de la simulation de l'Internet. Le projet VINT (`http://netweb.usc.edu/vint`) est une collaboration entre USC/ISI, Xerox parc, LBNL et UCB. Il a pour objectif principal de construire un simulateur multi-protocoles pour faciliter l'étude de l'interaction entre les protocoles et le comportement d'un réseau à différentes échelles. Le projet contient des bibliothèques pour la génération de topologies réseaux, des trafics ainsi que des outils de visualisation tel que l'animateur réseau nam (network animator). L'étude des comportements à des échelles différentes d'un réseau n'est pas obtenue par la simulation parallèle (qui peut être utile pour accélérer la simulation) mais par l'utilisation de techniques d'abstraction appliquées à différents éléments de la simulation. VINT est un projet en cours qui développe le simulateur NS.

Le simulateur NS actuel est particulièrement bien adapté aux réseaux à commutation de paquets et à la réalisation de simulations de petite taille. Il contient les fonctionnalités nécessaires à l'étude des algorithmes de routage unipoint ou multipoint, des protocoles de transport, de session, de réservation, des services intégrés, des protocoles d'application comme HTTP. De plus le simulateur possède déjà une palette de systèmes de transmission (couche 1 de l'architecture TCP/IP), d'ordonnanceurs et de politiques de gestion de files d'attente pour effectuer des études de contrôle de congestion. La liste des principaux composants actuellement disponible dans NS par catégorie est:

Application	Web, ftp, telnet, générateur de trafic (CBR, ...)
Transport	TCP, UDP, RTP, SRM
Routage	Statique, dynamique (vecteur distance) et routage multipoint (DVMRP, PIM)
Gestion de file d'attente	RED, DropTail, Token bucket
Discipline de service	CBQ, SFQ, DRR, Fair queueing
Système de transmission	CSMA/CD, CSMA/CA, lien point à point

Prises ensembles, ces capacités ouvrent le champ à l'étude de nouveaux mécanismes au niveau des différentes couches de l'architecture réseau. NS est devenu l'outil de référence pour les chercheurs du domaine. Ils peuvent ainsi partager leurs efforts et échanger leurs résultats de simulations. Cette façon de faire se concrétise aujourd'hui par l'envoi dans certaines listes de diffusion électronique de scripts de simulations NS pour illustrer les points de vue.

2. Documentation et sites essentiels

Les liens les plus intéressants sont regroupés sur la page:

http://www.univ-reunion.fr/~panelli/enseignement/2-ING/2-documentation.html

Le site de base de NS est maintenu par le projet VINT à l'URL `http://www.isi.edu/nsnam/ns/`. On y trouve des liens et les informations principales sur NS. Les principales documentations sont:

- Le *manuel ns;* Il constitue la référence de NS. Il décrit les commandes courantes de l'interpréteur et les principes pour développer de nouveaux composants.
- Le code Tcl des commandes de l'interpréteur. Il est contenu dans les fichiers `ns-lib.tcl`, `ns-node.tcl`, `ns-link.tcl`, `ns-agent.tcl`, `ns-default.tcl`, `ns-packet.tcl` du répertoire `tcl/lib`.
- Le tutoriel de Marc Greis. Il présente principalement les informations permettant de construire un modèle de simulation et montre comment utiliser l'interpréteur. Il n'y a que peu d'explications sur la création de nouveaux composants et sur le fonctionnement du simulateur. A noter que le tutoriel indique des liens sur la documentation de NS, Tcl, C++.

Le tutoriel est en ligne sur le site de NS et également fournis avec la distribution de NS dans le répertoire `ns-tutorial`.

3. Notions pour l'utilisation de l'interpréteur

Dans ce paragraphe nous allons donner les bases du langage Tcl, les principes de l'OTcl et les explications sur le mécanisme qui permet à un programme C d'utiliser un interpréteur Tcl. La référence dans ce domaine est l'ouvrage [2] dont une deuxième édition est maintenant disponible. Pour OTcl, la page à consulter est localisée à l'URL `ftp://ftp.tns.lcs.mit.edu/pub/otcl/doc/tutorial.html`

3.1. Tcl

3.1.1. Présentation

Tcl est un langage de commande comme le shell UNIX mais qui sert à contrôler les applications. Son nom signifie *Tool Command Language*. Tcl offre des structures de programmation telles que les boucles, les procédures ou les notions de variables. Il y a deux principales façons de se servir de Tcl: comme un langage autonome interprété ou comme une interface applicative d'un programme classique écrit en C ou C++. En pratique, l'interpréteur Tcl se présente sous la forme d'une bibliothèque de procédures C qui peut être facilement incorporée dans une application. Cette application peut alors utiliser les fonctions standards du langage Tcl mais également ajouter des commandes à l'interpréteur.

3.1.2. Lancement

Toutes les applications qui utilisent Tcl créent et utilisent un interpréteur Tcl. Cet interpréteur est le point d'entrée standard de la bibliothèque. L'application `tclsh` constitue une application minimale ayant pour but de familiariser un utilisateur au langage Tcl. Elle ne comporte que l'interpréteur Tcl. On retrouve cet interpréteur dans l'application NS. Lorsque l'on tape ns, une série d'initialisations est faite puis l'application passe en mode interactif. On peut alors entrer les commandes Tcl.

3.1.3. Concepts de base

Chaque commande consiste en un ou plusieurs mots séparés par des espaces ou des tabulations. Le langage n'est pas typé. Tous les mots sont des chaînes de caractères. Le premier mot de la commande est le nom de la commande, les autres mots sont les arguments passés à la commande. Chaque commande Tcl retourne le résultat sous forme d'une chaîne de caractères. Le caractère de retour à la ligne termine une commande et lance son interprétation. Le caractère de séparation de commandes sur une même ligne est ";". Une fois la commande exécutée et terminée, l'application retourne en mode interactif c'est à dire que l'interpréteur est de nouveau prêt à recevoir une nouvelle commande.

Tcl n'est pas un langage compilé, c'est un langage interprété. Il n'y a pas de grammaire fixe qui traite l'ensemble du langage. Tcl est défini par un interpréteur qui identifie les commandes par des règles d'analyse syntaxique. Seules les règles d'analyse des commandes sont fixées. Tcl évalue une commande en deux étapes: analyse syntaxique et exécution. L'analyse syntaxique consiste à identifier les mots et effectuer les substitutions. Durant cette étape, l'interpréteur ne fait que des manipulations de chaînes. Il ne traite pas la signification des mots. Pendant la phase d'exécution, l'aspect sémantique des mots est traité comme par exemple déduire du premier mot le nom de la commande, vérifier si la commande existe et appeler la procédure de cette commande avec les arguments.

3.1.4. Substitutions

Substitution de variable : `$<variable>`
Le contenu d'une variable est obtenu en faisant précéder le nom de variable par le symbole $

```
>set a 20
>expr $a*2
< 40
```

Substitution de commande : `[<commande>]`
La substitution de commande permet de remplacer la commande par son résultat. Les caractères entre les crochets doivent constituer un script Tcl valide. Le script peut contenir un nombre quelconque de commandes séparées par des retour à la ligne et des ";".

Anti-slash substitution: "\"
Il est utilisé devant les caractères retour à la ligne, $, [pour indiquer de les interpréter comme des caractères ordinaires. *newline* transforme le retour à la ligne en espace. L'anti-slash en fin de ligne indique par conséquent que la suite de la ligne est sur la ligne suivante. Devant des caractères ordinaires, il indique des caractères speciaux comme \\ pour \, \xhh pour une notation héxadécimale, \n pour un retour à la ligne.

3.1.5. Inhibitions

Il existe plusieurs façons d'empêcher l'analyseur syntaxique de donner une interprétation spéciale aux caractères tel que $ ou ;. Ces techniques se nomment inhibitions. Tcl fournit 2 formes d'inhibitions:

- **double quote** (") inhibe les séparations de mots et de commandes. Il reste les substitutions de variable et de commande. Les caractères espaces, tabs, retour à la ligne et point-virgule sont traités comme des caractères ordinaires.

```
>set date "22 Décembre"
>set msg " Aujourd hui: $date "
>set msg
<Aujourd hui: 22 Décembre
```

- **accolade** inhibe toutes les substitutions et tous les caractères spéciaux. Les caractères compris entre les crochet sont considérés comme un mot.

```
>set msg {Aujourd hui: $date}
>set msg
<Aujourd hui: $date
```

3.1.6. Conclusion

Dans cette section nous venons de donner les bases pour la lecture des scripts écrits en Tcl. Les différentes commandes de Tcl sont détaillées dans l'ouvrage [2] et dans la section 5 du manuel en ligne (man) d'UNIX.

3.2. OTcl

La documentation est accessible à l'URL `ftp://ftp.tns.lcs.mit.edu/pub/otcl/`. OTcl est une extension orientée objet de Tcl. Les commandes Tcl sont appelées pour un objet. En OTcl, les classes sont également des objets avec des possibilités d'héritage. Les analogies et les différences avec C++ sont:

- C++ a une unique déclaration de classe. En OTcl, les méthodes sont attachées à un objet ou à une classe.
- Les méthodes OTcl sont toujours appelées avec l'objet en préfixe
- L'équivalent du constructeur et destructeur C++ en OTcl sont les méthodes `init{}`/`destroy{}`
- L'identification de l'objet lui-même: `this` (C++), `$self` (OTcl). `$self` s'utilise à l'intérieur d'une méthode pour référencer l'objet lui-même. A la différence de C++, il faut toujours utiliser `$self` pour appeler une autre méthode sur le même objet. C'est à dire "`$self xyz 5`" serait "`this->xyz(5)`" ou juste "`xyz(5)`" en C++.
- Les méthodes OTcl sont tout le temps "virtual". A savoir, la détermination de la méthode à appeler est effectuée à l'exécution.
- En C++ les méthodes non définies dans la classe sont appelées explicitement avec l'opérateur de portée "::". En OTcl, les méthodes sont implicitement appelées dans l'arbre d'héritage par "`$self next`". Cette commande appelle la méthode de même nom de la classe parent.
- L'héritage multiple est possible dans les deux langages.

La définition d'une classe commence par la directive `Class`. Les fonctions et les attributs d'une classe s'installent par la suite par les commandes `instvar` et `insproc`. L'utilisation `instproc` définit les méthodes de la classe de manière assez semblable à C++. Par exemple écrire en C++ "`void XTP::send(char* data)`" s'écrit en OTcl "`XTP instproc send {data}`". Lors d'une instanciation

en OTcl, le constructeur de la classe appelle `init{}` pour les initialisations et les traitements propres à l'objet. La méthode `init{}` est optionnelle, c'est à dire que si l'instanciation ne nécessite aucune initialisation il n'est pas nécessaire de la définir.
Un attribut est une variable qui sera instanciée autant de fois que la classe comportera d'instances. On peut l'appeler également variable d'instance. La valeur de l'attribut appartient à l'objet. La commande `instvar` est utilisée dans une méthode pour mettre en correspondance une variable d'instance avec une variable locale à la méthode. Il n'est pas nécessaire de faire une déclaration préalable au niveau de la classe. Une déclaration d'une variable d'instance se fait à l'intérieur d'une méthode membre. Par exemple, pour déclarer la variable d'instance, il suffit d'écrire "`$self instvar x`". Le schéma général de déclaration d'une classe est le suivant:

```
Class <Nom de classe>
<Nom de classe> instproc init {args} {
# constructeur
...
}

<Nom de classe> instproc methode1 {args} {
$self instvar variable1                 # variable membre de la classe
...
}
etc.
```

En NS, les objets sont créés par la commande `new{}` qui retourne une référence sur l'objet créé. Par exemple, la création d'une instance de Simulator est effectuée par la commande suivante:

```
set ns [new Simulator]
```

Les objets topologiques (noeud et lien) ont une syntaxe de création quelque peu différente. On n'utilise pas la commande `new{}` mais une procédure de la classe Simulator. Dans les deux cas, la classe est uniquement OTcl et la référence des instances de ces classes est mémorisée dans un tableau de la classe Simulator. Par exemple la création d'un noeud est effectuée avec la syntaxe suivante:

```
$ns node
```

Une fois créés, les objets sont manipulables avec les méthodes dont ils disposent. Ces méthodes proviennent de leur classe et de leurs super-classes. L'appel d'une méthode est effectué via l'objet. Le nom de la méthode est le premier argument qui suit la référence de l'objet comme c'est le cas pour créer un noeud (cf exemple précédent). La méthode appelée est la première trouvée dans l'arbre d'héritage de l'objet. Les affectations sur les variables membres se font par la commande `set{}` et suivent le format général des commandes OTcl:

```
<référence instance> set <instance variable> <valeur>
```

L'héritage est indiqué par l'option `-superclass <nom de la super-classe>` après la directive Class. Par exemple la class1 hérite de la class2 et class3 en utilisant la syntaxe suivante:

```
Class class1 -superclass {class2 class3}
```

Lors des créations d'objets en NS, la première ligne est souvent `eval $self next $args`. La méthode `next` est utilisée pour appeler la méthode de même nom dans la hiérarchie de classe. Elle sert à effectuer une agrégation de méthode. Si aucune autre méthode de même nom n'est trouvée, une chaîne nulle est retournée sinon les arguments passés avec `next` sont passés à la méthode. Par la suite, nous reviendrons sur la création des objets.

Les détails les plus "croustillants" à propos de OTcl sont dans sa documentation. La maîtrise d'OTcl comme celle de C++ demande d'abord une bonne connaissance des concepts objets plutôt que celle de la syntaxe elle-même qui reste d'ailleurs très simple.

3.3. Commandes de Tcl

Selon les règles du langage Tcl, une commande Tcl consiste en un à plusieurs mots. Le premier mot est le nom de la commande, les mots suivants les arguments. Les mots sont séparés par des espaces, les commandes sont séparées par retours à la ligne ou par le caractère ";". Un script Tcl se présente sous la forme d'une suite de *commandes*.
Le langage Tcl est sensible à la case des caractères: une même lettre en majuscule et en minuscule est traitée comme deux caractères différents.
Le commentaire commence par le caractère # et se termine en fin de ligne.

3.3.1 Les variables

Les variables ne sont pas déclarées, elles sont définies au moment de leur utilisation par la commande `set`. La commande `set` est la commande d'affectation et de consultation d'une variable. La commande `unset` rend la variable indéfinie. Le seul type de valeur manipulé par Tcl est le type chaîne de caractères. Il existe de types de variables: les variables simples et les variables tableaux. L'indice d'un élément de tableau peut être n'importe quelle chaîne.

Éléments de syntaxe	**Signification**
$A	valeur de la variable A
$tab(5a)	valeur du tableau indicé par la chaîne de caractères 5a
set B $A	B prend la valeur de la variable A
set i [expr 2 + 3]	La commande [expr 2 + 3] est évaluée et le résultat est affecté à la variable i
set tab($i) 10	L'élément du tableau tab d'indice indiqué par la valeur de la variable i prend la valeur 10
set C $tab($i)	La variable C prend la valeur de l'élément du tableau indicé par la valeur de la variable i
set A "une chaine de caractères"	Affectation d'une chaîne de caractères à la variable A

3.3.2. Les listes

Une liste est un ensemble ordonnée de chaîne de caractères. Les listes peuvent s'imbriquée dans des listes. Les listes sont notées entre accolades.

Éléments de syntaxe	**Signification**
{A B C}.	Une liste de 3 éléments
{a {b c} d} e	La liste {b c} est imbriquée dans une liste de 3 éléments elle-même formée par 2 éléments
{}	liste vide
list ?*arg arg ...*?	retourne une liste constituée par les arguments
lappend *varName* ?*value value value ...*?	Traite *varName* comme une liste et ajoute les valeurs dans la liste *varName*
lindex *list ?index...?*	Retourne l'élément de rang index de la liste donnée en argument. Le premier élément a l'index 0.
split *string* ?*splitChars*?	Convertit une chaîne en une liste, le séparateur de mot est *splitChars*
join *list* ?*joinString*?	Transforme une liste en un chaîne
concat ?*arg arg ...*?	Retourne une liste produit de la concaténation de plusieurs liste
llength *list*	Retourne le nombre d'éléments de la liste
lrange *list first last*	Retourne une sous liste commençant à l'élément de rang *first* jusqu'à l'element de rang *last* (compris)
lreplace *list first last* ?*element element ...*?	Retourne une nouvelle liste en remplaçant des éléments

3.3.3. Expressions

Une expression combine les valeurs et les opérandes pour produire de nouvelles valeurs. Un expression en Tcl suit la syntaxe et utilise les opérateurs du langage C. La commande `expr` évalue une expression et retourne une valeur. Il existe des fonctions prédéfinies qui peuvent être utilisées dans une expression. Un expression booléenne retourne 0 quand le résultat est faux.

3.3.4. Structures de contrôle

Les structures de contrôle Tcl sont les mêmes que l'on traouve dans les autres langages. Les structures de contrôle sont traitées en interme comme une commande. Les différents éléments de la structure de contrôle sont des listes. Dans ces conditions, il est important de mettre une accolade ouvrante à la fin de la ligne (et non pas en début de ligne suivante) si un élément doit être décrit sur plusieurs lignes.

Lorsqu'une commande comporte un élément qui est une expression, celle-ci doit être mise entre accolades. Par exemple, pour un while, l'expression doit être evaluée à chaque itération. L'expression est évaluée puis la commande while est exécutée. Sans l'accolade, l'expression ne contient plus que des constantes. Avec des accolades, l'expression est passée sans évaluation à la commande while qui en fera l'évaluation à chaque itération. Dans le cas sans accolade, on a une boucle sans fin.

if *expr1* ?**then**? *body1* **elseif** *expr2* ?**then**? *body2* **elseif** ... ?**else**? ?*bodyN*?

```
if {$vbl == 1} {
   puts "vbl is one"
} else {
   puts "vbl is not one"
}
```

La partie else est optionnelle. Plusieurs tests peuvent s'enchaîner avec le mot clef elsif.

while *test body*

Une boucle dont le test est évalué avant l'exécution du body. Le corps peut contenir des break et continue, comme en C.

foreach *varname list body*

énumération d'une liste, varname prends la valeur des éléments successifs de la liste. Les instructions break et continue peuvent être employées.

for *start test next body*

La commande de boucle. Quand le test retourne une valeur 0, l'itération s'arrête.

switch ?*options*? *string* {*pattern body* ?*pattern body* ...?}

La commande de sélection par cas, à chaque cas est associé un script. Si plusieurs cas partagent le même script, on utilise un "-".

```
switch -exact $car {
   a { puts a }
   b -
   c { puts "b ou c" }
   default { puts ?}
}
```

3.3.5. Procédure

La commande proc :

proc *name args body*

Elle sert à créer des nouvelles commandes qui sont utilisées ensuite comme des commandes de Tcl. Toutes variables utilisées dans une procédure restent locales à cette procédure. Il faut utiliser la directive global pour accéder aux variables globales.

```
proc fonc { par1 par2 } {
  global var1 var2
  # code
  return $quelquechose
}
```

4. Commandes usuelles de NS

Les commandes associées à la classe Simulator. La classe Simulator fournit un ensemble de méthodes pour configurer une simulation. Un script de simulation commence habituellement par la création d'une instance de cette classe. Ensuite les noeuds, les liens et les autres aspects de la simulation sont crées.

La convention de nommage des variables de ns indique qu'une variable d'instance à un nom qui est suffixé par le caractère "_".

```
Simulator instproc now ;                # return scheduler's notion of current
                                        time
Simulator instproc at cmd ;             # schedule execution of code at
                                        specified time
Simulator instproc cancel args ;        # cancel event
Simulator instproc run ;                # start scheduler
Simulator instproc halt ;               # stop (pause) the scheduler
Simulator instproc flush-trace ;        # flush all trace object write buffers
Simulator instproc trace-all filefd ;   # ask to trace all the packet events.
                                        The file descriptor must be open with
                                        the appropriate permission.
Simulator instproc node                 Command to create and return a node
                                        instance.
Simulator instproc duplex-link { n1 n2  constructs a bi-directional link from
bw delay type }                         two simplex links.
Simulator instproc create-connection    This sets up a complete connection
{s_type node1 d_type node2 fid}         between two agents.
Simulator instproc attach-agent { node  attaches the <agent> to the <node>.
agent }
Simulator instproc queue-limit { n1 n2  arrange for queue length of link
limit }                                 between nodes n1 and n2
Agent instproc attach-app {s_type}      Add application of type s_type to
                                        agent and return the app
```

Le format de trace est le suivant:

```
<type evt> <tps> <orig> <dest> <type pkt> <taille> --- <flow id> <id src> <id
dest> <seq> <attr>
```

Où le `<type evt>` est:

- `+` pour un paquet mis en file d'attente,
- `-` pour un paquet sortie de la file d'attente
- `d` pour une perte de paquet (drop),
- r pour la réception d'un paquet à un noeud

5. Références

[1] **Paxson, V. and Floyd, S.** (1997). Proceedings of the Winter Simulation Conference, Dec. 1997.
Why we don't know how to simulate the Internet.

[2] **Ousterhout, J.K.** (1994). Ed.: Addison-Wesley Publishing Company, 458 p.
Tcl and the TK Toolkit.