

Séminaire OLC : Tutoriel d'introduction au simulateur NS

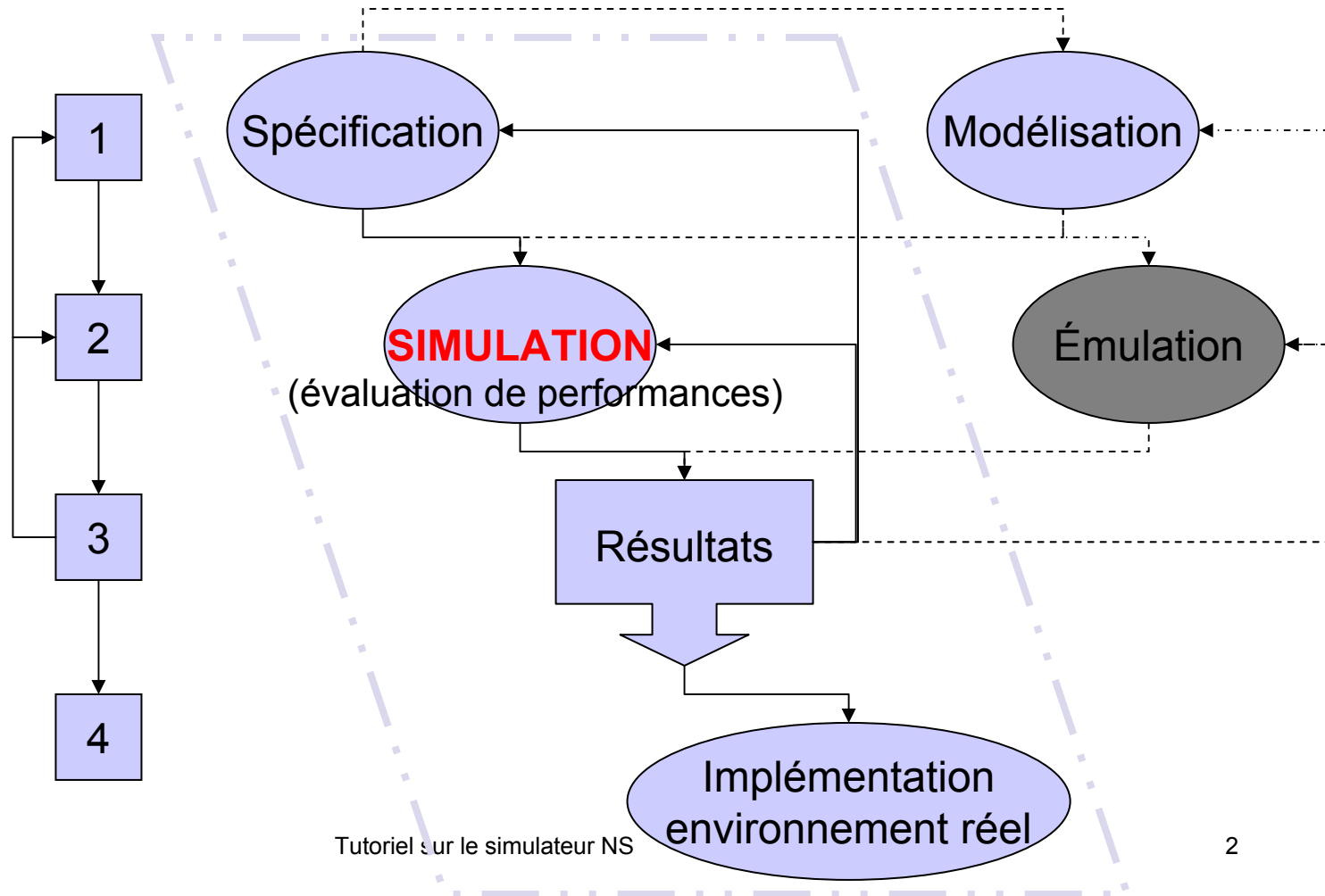
Guillaume AURIOL
Nicolas LARRIEU



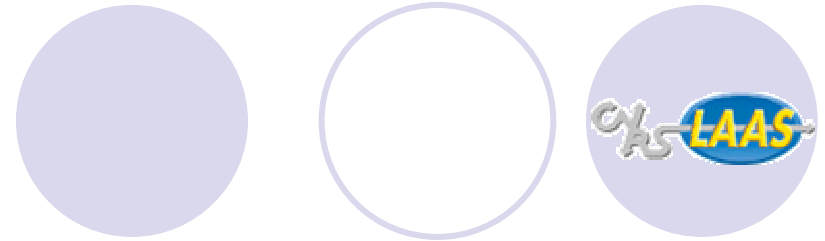
Introduction



- Processus de recherche en réseau



Introduction



- Les différents types de simulateurs

- formel :

- basé sur le langage RT-Lotos
- ...

- **non formel :**

- à temps discret : DTNS (Discrete Time Network Simulator)

- **à évènements discrets :**

- NS-2 (IETF),

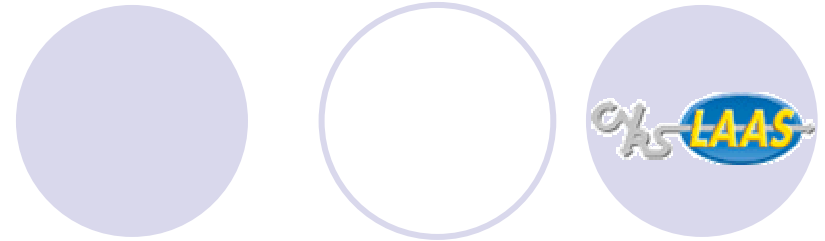
- OPNET (MIL3) :

-  intuitif : interface graphique, ajout nouveaux modules simples

-  peu ouvert : code non open-source, licence onéreuse

- PARSEC (PARallel Simulation Environment for Complex systems),
- SSF (Scalable Simulation Framework),
- JavaSim

Introduction



● Pourquoi choisir NS ?

- Support pour la recherche en réseau (et l'enseignement des réseaux)
 - Conception de protocoles, études de trafic, etc.
 - Évaluation / comparaison des protocoles
 - Évaluation de performances

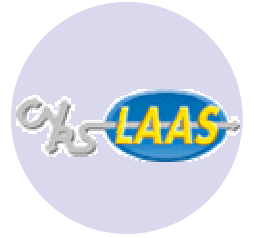
- Mise à disposition d'un environnement propice à la collaboration
 - Librement distribué, code open-source
 - Partage du code, des protocoles, des modèles, etc.
 - Permet une comparaison aisée de protocoles aux fonctionnalités similaires
 - ⇒ Augmente la confiance dans les résultats
 - Modèles testés dans de nombreuses situations par beaucoup de chercheurs
 - De nouveaux modèles sont développés par des experts

- Plusieurs niveaux de détails dans le simulateur
 - ⇒ Niveaux d'abstraction différents
 - ex : routage, propagation paquet, modèle de trafic, application, etc.

Introduction

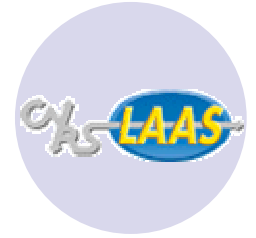
- 1995 : création de NS-2 avec le projet VINT (Virtual InterNetwork Testbed) (LBNL, Xerox PARC, UCB, USC/ISI : *University of Southern California / Information Sciences Institute*)
- Site officiel pour télécharger le code source
 - www.isi.edu/nsnam/ns
- Mises à jour périodiques (ns-2.1b9a, juillet 2002)
 - ~200K lignes de codes en C++ et OTcl,
 - ~100 suites de test et plus de 100 exemples
 - manuel de fonctionnement détaillé
- Validation de sa stabilité
 - www.isi.edu/nsnam/ns/ns-tests.html
- Plateformes supportées
 - FreeBSD, Linux, Solaris, Windows et Mac
- Utilisateurs
 - > 1k instituts (50 pays), > 100k utilisateurs
 - ~300 mails / mois sur ns-users@isi.edu

Plan de la présentation



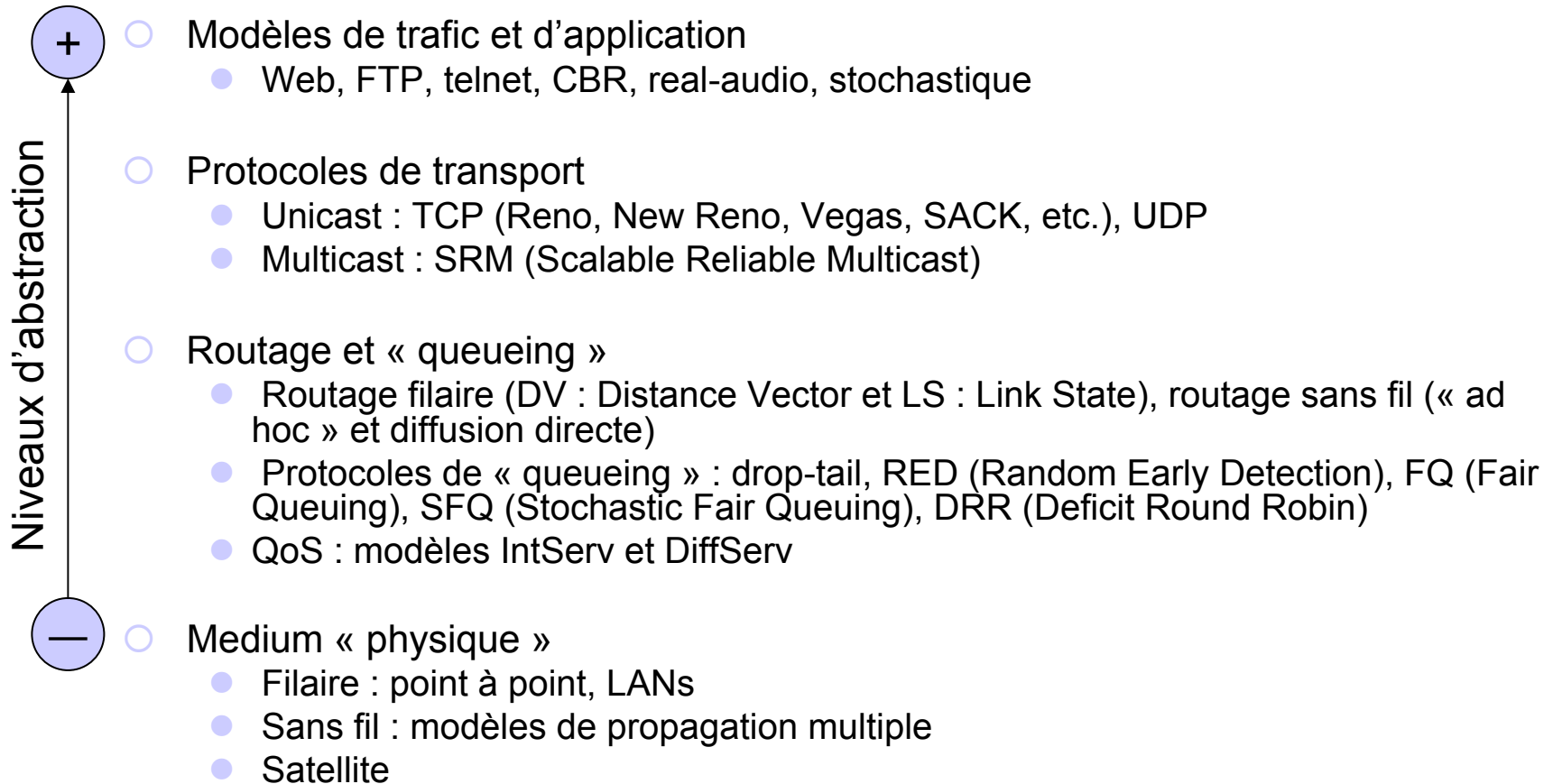
- Les composants de NS
 - Ses grandes fonctionnalités
 - Les modèles disponibles
- Le fonctionnement du simulateur
 - Architecture
 - Le cœur du simulateur
 - Son interface : Utilisateur/Cœur du simulateur
- Les utilisations de NS
 - Les différentes étapes de la simulation
 - Les résultats de simulation
 - La modularité de NS : ajouts / évolutions possibles
 - Les outils liés à NS
- Conclusion

Les composants de NS

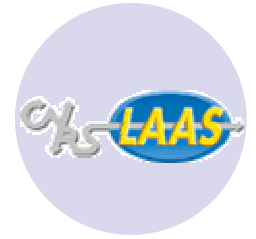


- NS : le simulateur
- NAM (Network AniMator) : l'animateur de réseau
 - Edition : interface GUI permettant la génération de scripts NS simples
 - Visualisation graphique des résultats NS
- Utilitaires de « prétraitement » :
 - Générer des topologies (BRITE, NEM...)
 - Générer du trafic (cf. modifications du simulateur)
- Utilitaires de « post traitement » (cf. évaluation de performances) :
 - Dépouillage des simulations (Awk, routines en Perl, Tcl, C, etc.)
 - Visualisation (statistique) des résultats de simulation

Modèles disponibles



Fonctionnement du simulateur NS



- Architecture du simulateur

**Interface utilisateur
(contrôle)**
Langage OTcl

**Cœur du simulateur
(données)**
Langage C++

- Orientée objet

- Approche modulaire

 - Composition d'objets très fine

● Réutilisation

● + Maintenance

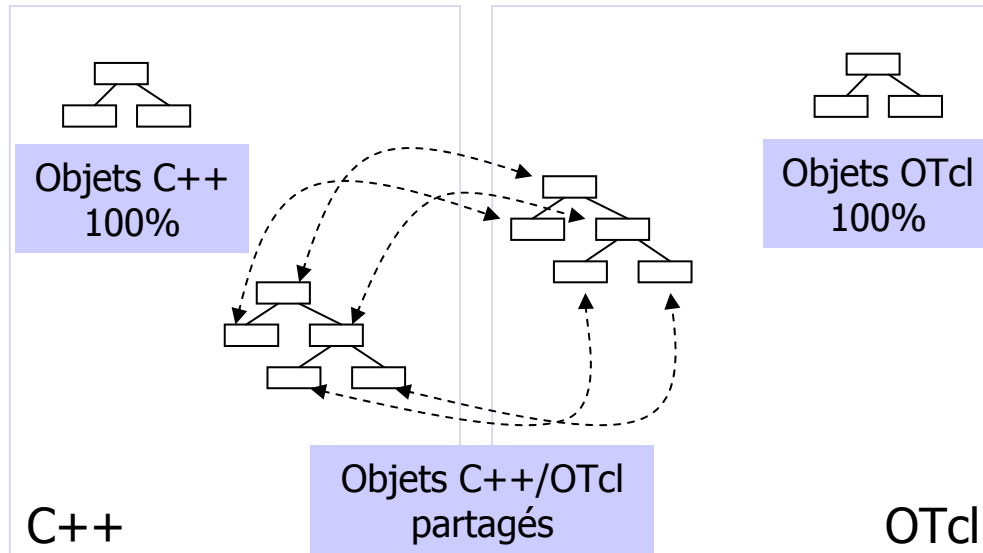
Modifications / ajouts possibles

● Performances (vitesse / mémoire)

● - Complexité / taille des simulations

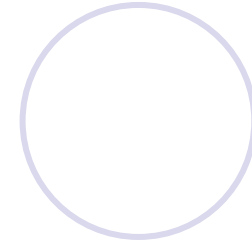
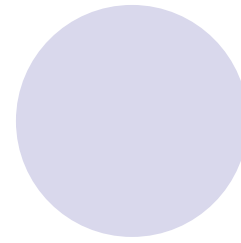
Développement intelligent ⇒ modularité

Dualité OTcl / C++



- OTcl et C++ partage une même hiérarchie des composants
- TclClass : librairie qui assure le lien entre les classes OTcl et C++ (partage de fonctions, de variables, etc.)

Cœur du simulateur

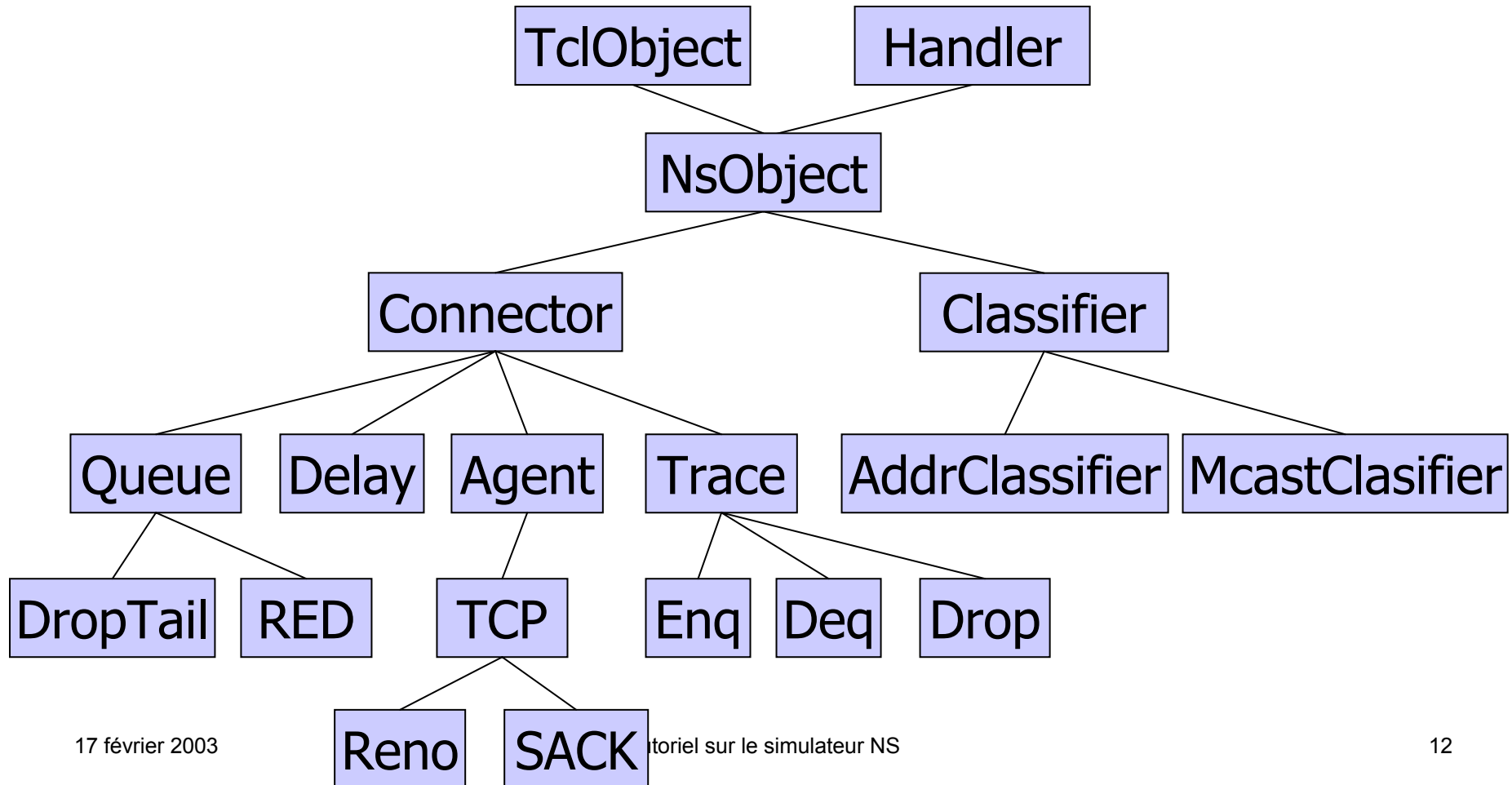


- Rôle : gestion des données / définition des comportements
- Gestion par paquet
- Langage C++ (rapidité d'exécution)
- Fonctionnement détaillé
- Contrôle total de son fonctionnement
 - ⇒ débogueur évolué
- Architecture des classes (composants NS) hiérarchisée

Cœur du simulateur (suite)



- Architecture hiérarchisée par composant



Interface utilisateur

- (O)Tcl

ooo

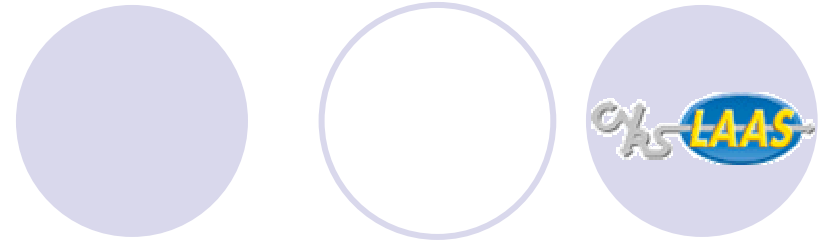
```
proc creer-noeud {nombre-noeuds nom} {  
    global ns  
    for {set i 1} {$i < $nombre-noeuds} {incr i} {  
        set $nom($i) [$ns node] ; # Declaration des noeuds  
        puts "Creation des noeuds : $i "  
    }  
}
```

ooo

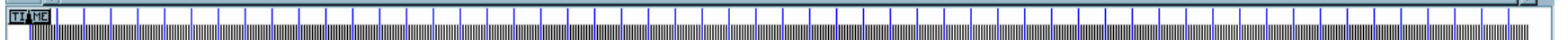
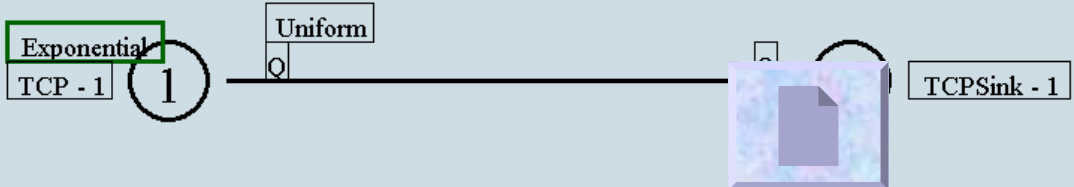
```
# Creation des emetteurs  
creer-noeud 4 $emetteurs  
  
# Creation des recepteurs  
creer-noeud 4 $recepteurs
```

ooo

Utilisation de NS



- Les différentes phases pour réaliser une simulation :
 - définition de la topologie (utilisation de NAM)
 - définition des acteurs de la simulation
 - définition de la dynamique de la simulation



Exécution de la simulation

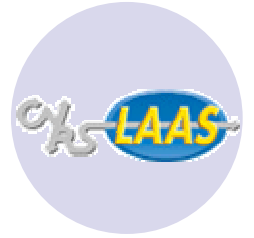


- Importance du dimensionnement (topologie, type et nombre d'agents, trafic applicatif, etc.)

⇒ Pb : simplification du réel vs. réalisme des simulations

- Capacité de calcul nécessaire (mémoire, processeur)
- Temps de calcul pour l'exécution de la simulation très variable (fonction de la complexité et de la taille)

Résultats de simulation



- Présentation des différents formats de sortie :

- fichier *.tr :

fichiers standards de sortie des simulations

- fichier *.nam :

fichiers standards pour exploitation sous NAM

=

Informations des fichiers .tr

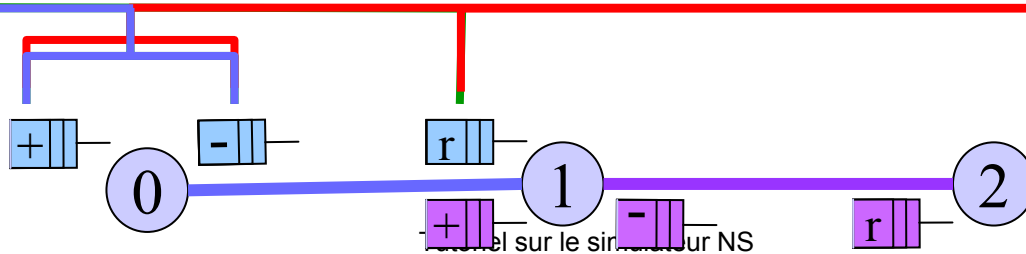
+

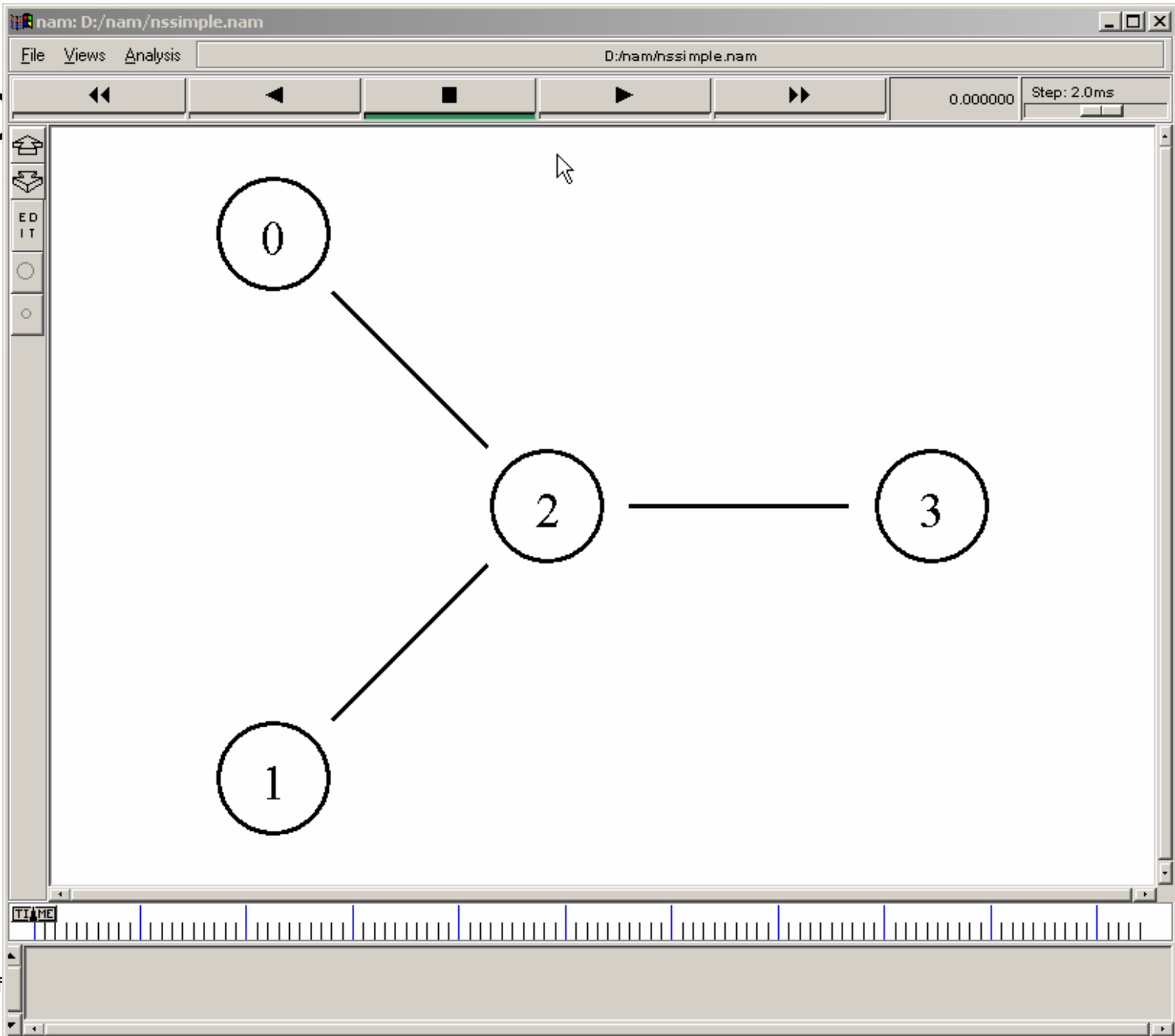
mémorisation de la topologie

Exemple de fichier de trace .tr

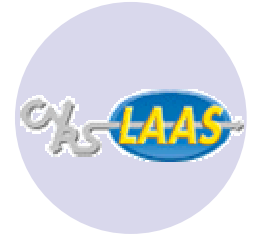


	date	nd depart	nd arrivee	proto	taille	flag	f_id	src	dst	Ø	p_id
+	0,1	0	1	cbr	980	-----	1	0	2	0	0
-	0,1	0	1	cbr	980	-----	1	0	2	0	0
+	0,1078	0	1	cbr	980	-----	1	0	2	1	1
-	0,1078	0	1	cbr	980	-----	1	0	2	1	1
r	0,1139	0	1	cbr	980	-----	1	0	2	0	0
+	0,1139	1	2	cbr	980	-----	1	0	2	0	0
-	0,1139	1	2	cbr	980	-----	1	0	2	0	0
+	0,1157	0	1	cbr	980	-----	1	0	2	2	2
-	0,1157	0	1	cbr	980	-----	1	0	2	2	2
r	0,1218	0	1	cbr	980	-----	1	0	2	1	1
+	0,1218	1	2	cbr	980	-----	1	0	2	1	1
-	0,1218	1	2	cbr	980	-----	1	0	2	1	1
+	0,1235	0	1	cbr	980	-----	1	0	2	3	3





Dépouillage des résultats (suite)



- Analyse quantitative : routines à écrire (scripts Awk, programmes en C, Perl, etc.)

Calculs évolués

+ Analyse plurifactorielle

Relation entre les paramètres étudiés

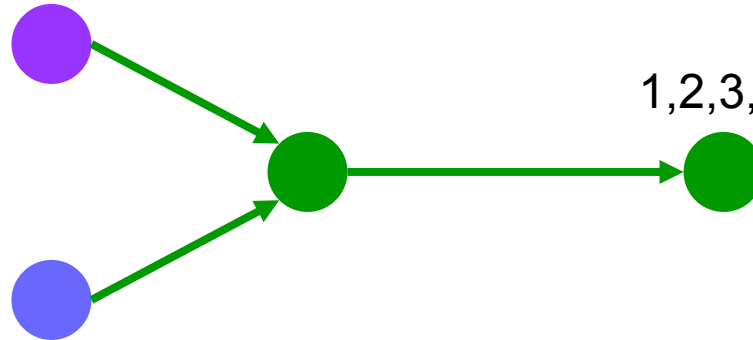
- Complexité des programmes
Maîtrise du format des traces

Exemple de dépouillage

QoS / flux

- Objectif : Récupérer individuellement pour chaque paquet d'un flux la date d'émission et la date de réception (ou détecter la perte)
- Problème : Numérotation des paquets continu \forall flux

...,18,14,11,8,4,1



...,17,16,15,13,12,11,10,9,7,6,5,3,2

QdS / flux (suite)

• Solution : Programme C !!!



• paramètres d'entré :

Nœud_source,
Identifiant_de_Flux,
Nœud_destination,
.....

• paramètres de sortie : Tableau de résultats

p_id	Délai de bout en bout (s)
1	0.01856
2	0.0197
3	d
4	0.0246
...	...

• performances : (~ 10 ko , ~ 30 s) / ~100 Mo de traces tr
(~250 Mo de traces .nam)

QdS / flux (fin)

0 / 9125 paquets perdus / total paquets flux

n0

Exemple

- 2 scénarios
- scénario 1
- scénario 2

```
000
```

```
# Recuperation de l'argument
```

```
set scenario [lindex $argv 0]
```

```
000
```

```
# Ouverture du fichier tr
```

```
set f [open $scenario.tr w]
```

```
# Traitement en fonction du scenario
```

```
if {$scenario=="scenario1" } {
```

```
000
```

```
}
```

```
#!/bin/sh
```

```
# Lancement de la simulation
```

```
for scenario in scenario1 scenario2
```

```
do
```

```
    ns essai.tcl $scenario
```

```
done
```

```
#!/bin/sh
```

```
# Lancement du depouillage pour les 2 scenarios
```

```
for scenario in scenario1.tr scenario2.tr
```

```
do
```

```
    echo "debut depouillage de $scenario"
```

```
    ../../softdepouillage/miseEnForme $scenario $scenario.txt 0 0 3
```

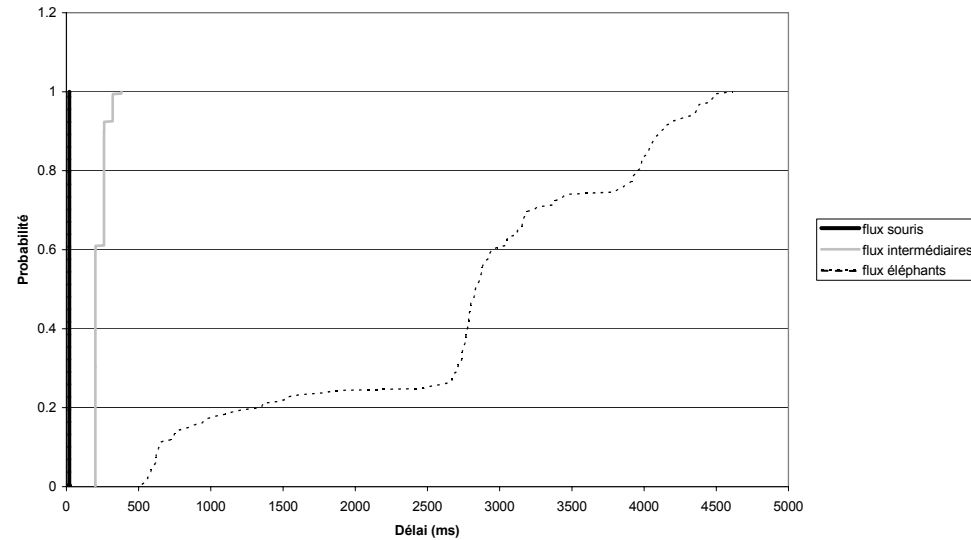
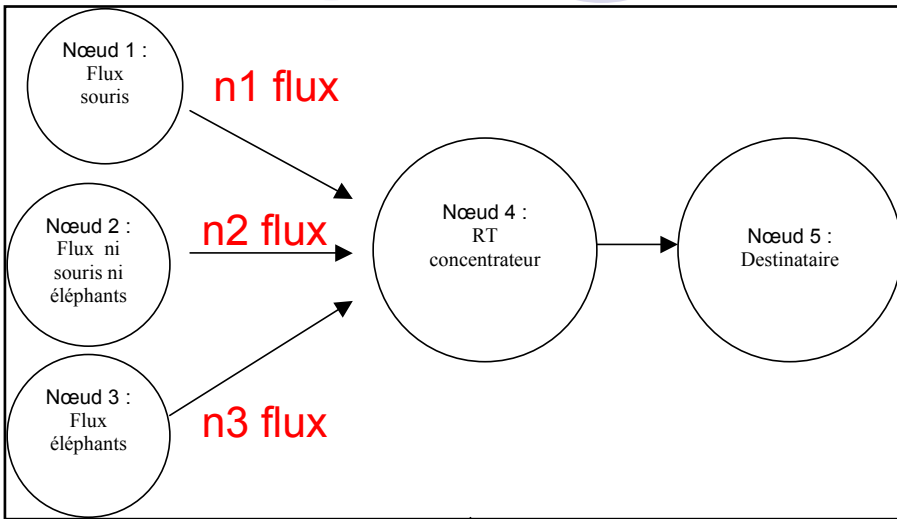
```
    echo "fin depouillage de $scenario"
```

```
done
```

Exemple d'analyse des résultats (langage C) orientée évaluation de performances



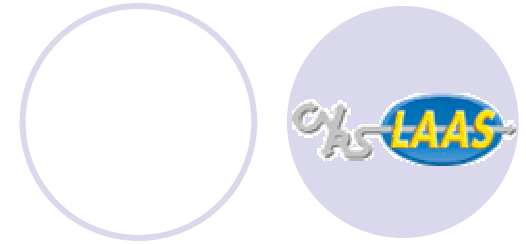
Délai d'émission



----- 1 -----
Calcul du délai à l'émission / flux pour :
- les n1 flux souris
- les n2 flux intermédiaires
- les n3 flux éléphants

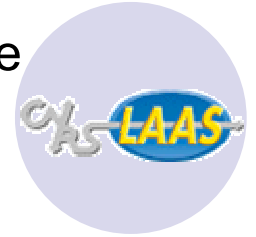
----- 2 -----
Calcul des CDF pour les délais à l'émission / type de flux :
- des flux souris
- des flux intermédiaires
- des flux éléphants

Modularité du simulateur



- Les possibilités d'évolution
 - approche théorique : architecture permettant une évolution des composants (*modifications, ajouts*)
 - approche pratique :
 - Ajout d'un ordonnanceur type WFQ (cf. routeur AIRS)
 - Le rejeu de trace
 - Simulation d'une nouvelle version de TCP

Approche théorique : Eléments pour ajouter un module



- Création de l'objet via l'interpréteur OTcl
- Clonage de cet objet en un objet compilé (C++) et codage du comportement

- Cas particuliers :
 - *Les classes contenues uniquement dans l'interpréteur pour aider à manipuler d'autres classes*
ex. : agrégation de classe : duplex-link, simplex-link,...

 - *Les classes contenues uniquement dans le simulateur pour servir aux fonctions internes, elles ne seront donc pas accessibles aux utilisateurs*

- La classe mère des 2 types d'objet (Tcl/C++) est TclObject et tout nouvel objet possède une classe héritant de la classe TclClass

- Au lancement, toutes les classes de TclClass sont créées, le constructeur de TclClass enregistre les classes en OTcl

Approche pratique : WFQ

- Les fichiers nécessaires :



Pour l'interface

Interpréteur
ns-wfq.tcl

Simulateur
wfq.h
wfq.cc

Pour le comportement

C++

```
000  
else if (argc == 5) {  
    if (strcmp(argv[1], "install") == 0) {  
        int codepoint = atoi(argv[2]);  
        if (hash_.lookup (codepoint) != NULL) {  
            tcl.resultf("WFQ: PHB %s already defined",argv[2]);  
            return (TCL_ERROR);  
        }  
        Queue* q = (Queue*) TclObject::lookup(argv[3]);  
        double weight = atof(argv[4]);  
        PHB* newphb = new PHB(this, q);  
        newphb->weight() = weight;  
        hash_.insert(newphb, codepoint);  
        num_classes_++;  
        return (TCL_OK);  
    }  
}
```

ϕ codepoint ϕ install ϕ codepoint ϕ queue ϕ (ϕ codepoint) ϕ weight

Approche pratique : WFQ (suite)

- L'architecture des fichiers ns

ns-2/Makefile.in :

OBJ_CC = ...\

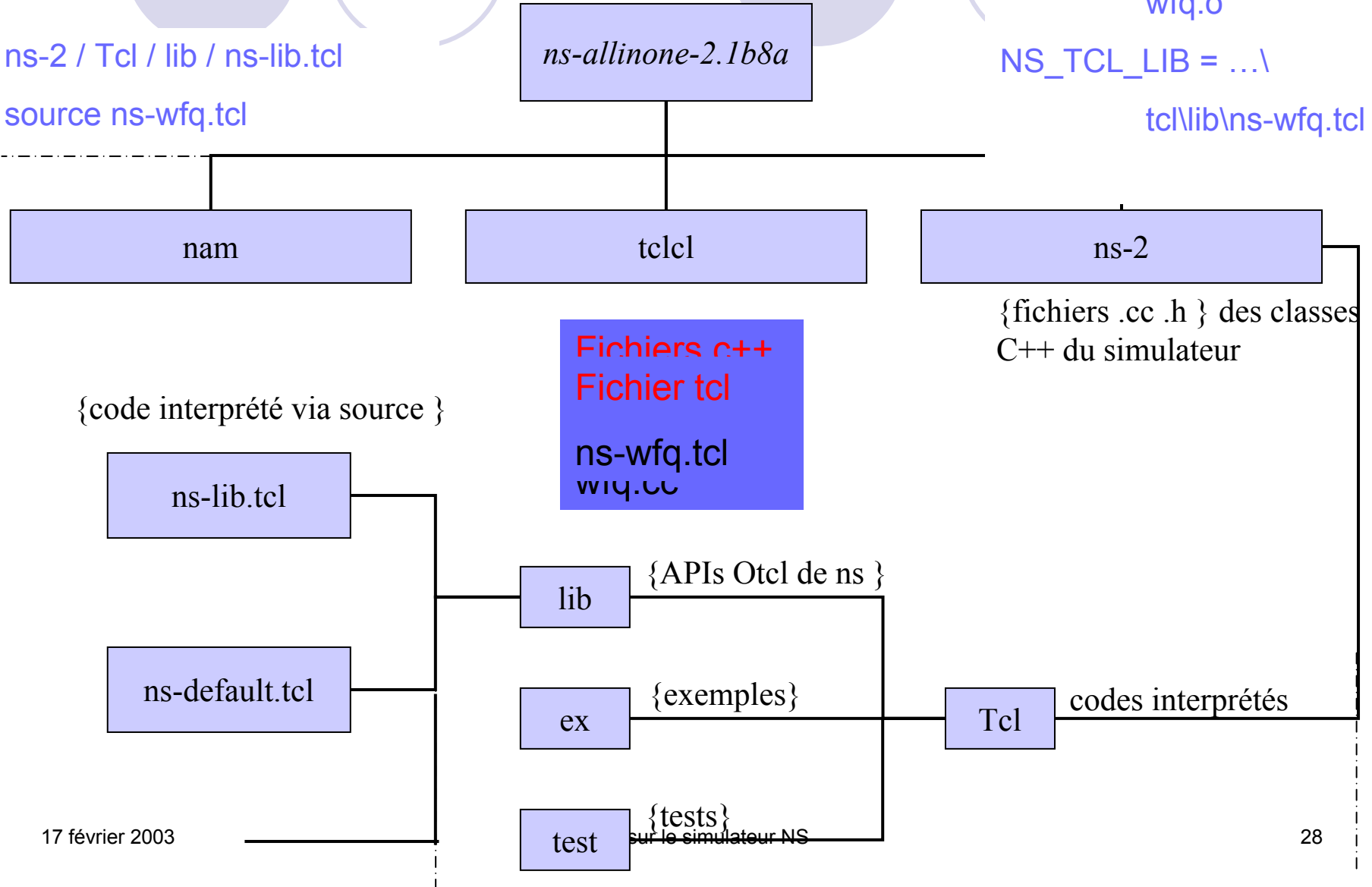
wfq.o

NS_TCL_LIB = ...\

tc\lib\ns-wfq.tcl

ns-2 / Tcl / lib / ns-lib.tcl

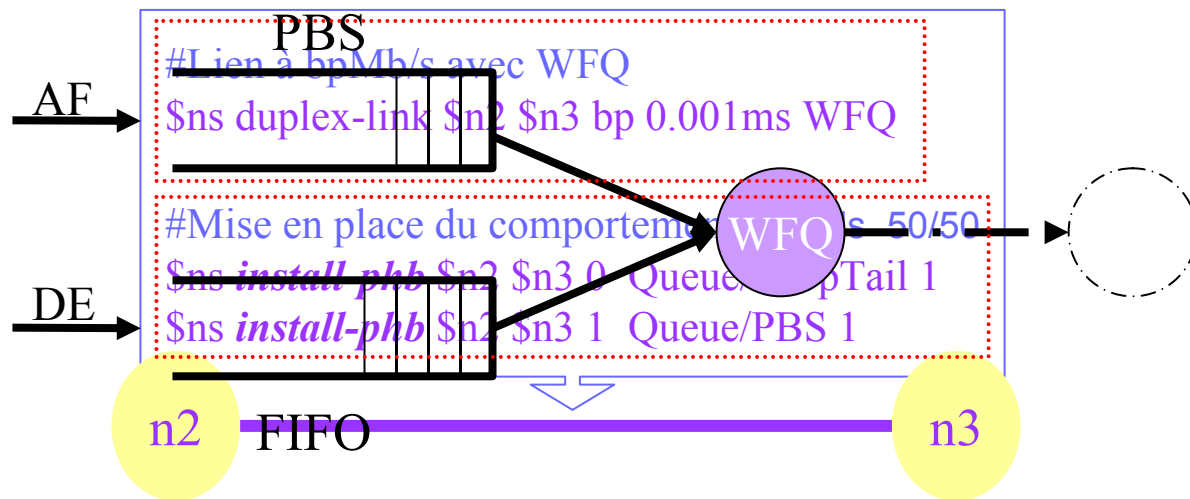
source ns-wfq.tcl



Modélisation d'un routeur AIRS



- Simplification par découplage des mécanismes



- Pourquoi rejouer des traces dans NS ?
 - Besoin de disposer de sources de trafic réalistes
 - Amélioration du réalisme des simulations
 - Traces métrologiques en entrée :
 - réseau de cœur : OC-3 à 155 Mbps (Sprint)
 - réseau de bordure : Ethernet à 10 Mbps (lien d'accès à vBNS NZ)

Fonctionnement du rejeu de trace

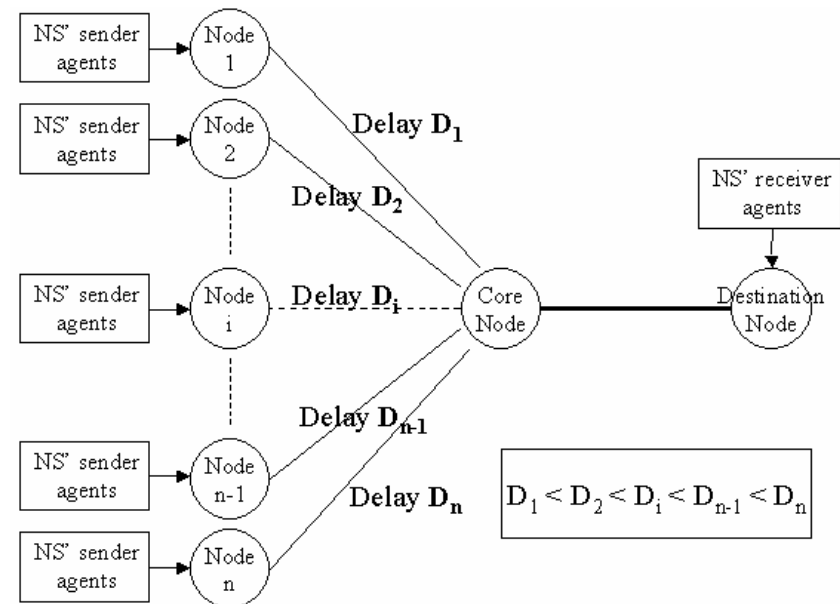


- **Prétraitement / extraction :**
 - des instants de début de chaque flux dans le profil général
 - de la taille de chacun des paquets émis à l'intérieur d'un flux
 - du RTT de chaque flux de la trace
- **Simulation :**
 - un fichier TCL : topologie, dynamique des flux
 - un fichier binaire / flux de la trace

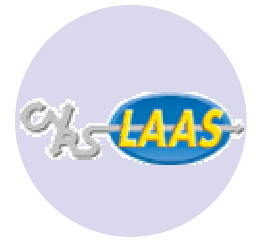
Fonctionnement du rejeu de trace (suite)



- Topologie d'un rejeu de traces :
 - en émission : un couple agent TCP / TrafficTrace par flux de la trace
 - en réception : un agent TCPSink
 - prise en compte du RTT / flux dans la topologie

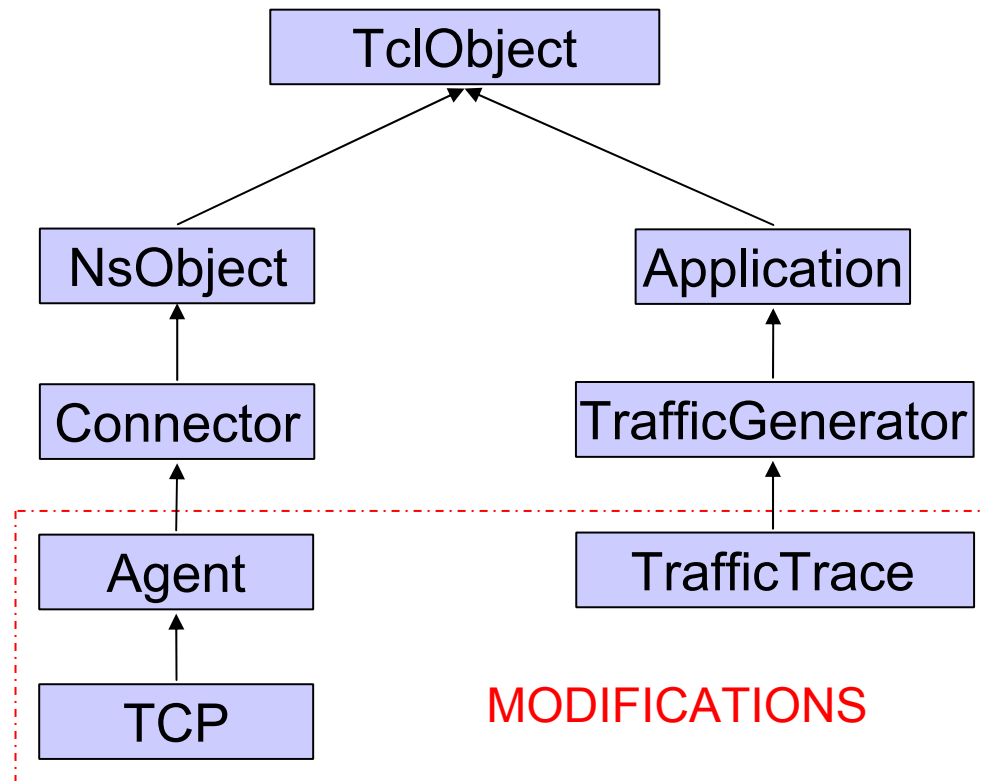


Mise en place du rejeu de trace dans NS



- Modifications apportées au simulateur

- uniquement dans la partie données (C++)
- classes Agent, TCP et TrafficTrace modifiées
- gestion des paquets de tailles différentes
- suppression des durées inter-paquets (VBR) ⇒ contrôle de congestion de TCP



Utilisation du module de rejeu de trace dans NS



- Extrait du code TCL nécessaire pour réaliser une simulation à base de rejeu

1. Connexion d'un générateur de trafic à son fichier de données (1 fichier / flux TCP)

Définition du fichier de trace

set fichierTrace [new Tracefile]

\$fichierTrace filename \$nom_fichierTrace

Création et connexion du générateur de trafic

set generateur_trafic [new Application/Traffic/Trace]

\$generateur_trafic attach-tracefile \$fichierTrace

2. Connexion d'un agent TCP avec le générateur de trafic

Création de l'agent TCP émetteur

set agentEmetteur [new Agent/TCP/Newreno]

\$ns attach-agent \$noeud_emission \$agentEmetteur

Connexion de l'agent TCP avec le générateur de trafic

\$generateur_trafic attach-agent \$agentEmetteur

Utilisation du module de rejeu de trace dans NS (suite)



3. Connexion d'un agent récepteur à l'agent émetteur

Création de l'agent récepteur

```
set agentRecepteur [new Agent/TCPSink]
```

```
$ns attach-agent $noeud_reception $agentRecepteur
```

Connexion du récepteur à l'émetteur

```
$ns connect $agentEmetteur $agentRecepteur
```

4. Dynamique de l'agent TCP et de la simulation

```
$ns at $debutFlux "$generateur_trafic start"
```

```
$ns at $duree_simulation "finish"
```

Élaboration d'un nouveau protocole avec NS

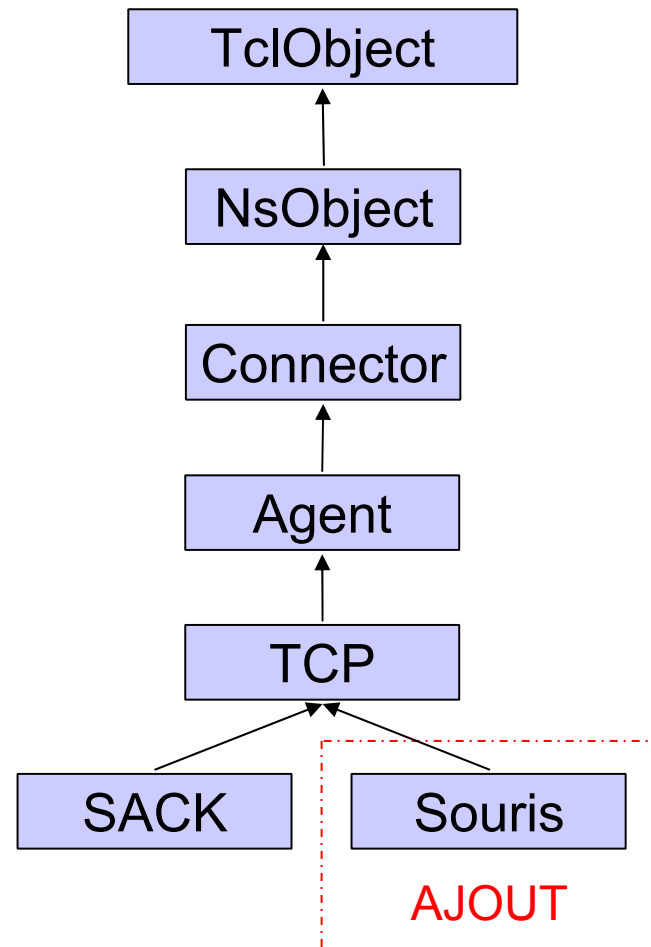


- Simulation du protocole TCP sans Slow-start pour les flux souris
 - Émission de tous les paquets du flux dans une seule fenêtre de données
 - ⇒ amélioration de l'agressivité de TCP
 - ⇒ augmentation de la QoS BE pour ce type de flux
 - NS va permettre
 - la vérification de ce protocole
 - l'évaluation des performances de ce mode de fonctionnement

TCP sans Slow-Start pour les souris

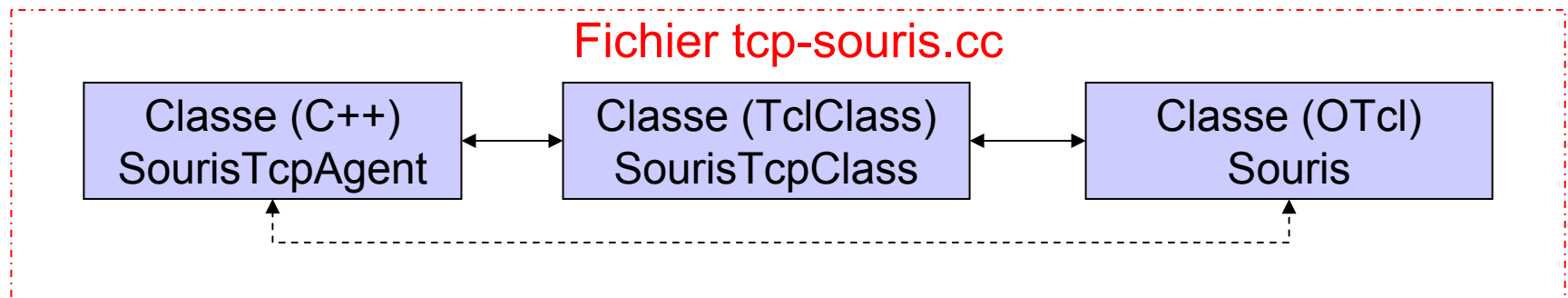


- Modifications apportées au simulateur
 - uniquement la partie données (C++)
 - classe « tcp » modifiée (tcp.h)
 - création de la classe « tcp-souris » (tcp-souris.cc)

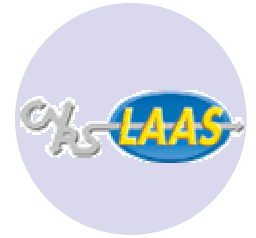


TCP sans Slow-Start pour les souris (suite)

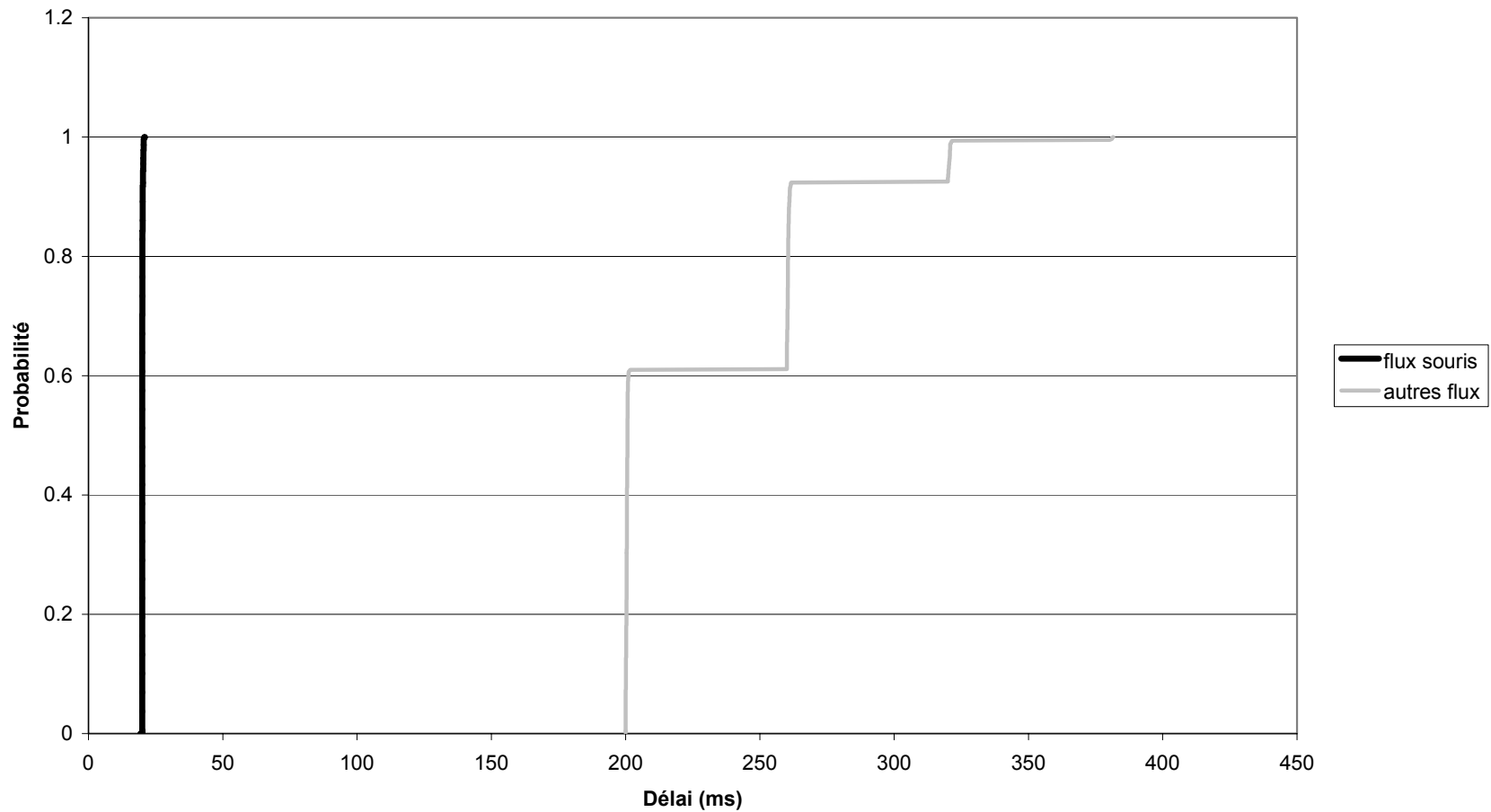
- Modifications apportées au simulateur (tcp-souris.cc)
 - Définition d'un constructeur pour la classe SourisTcpAgent
 - Redéfinition (surcharge) de la méthode sendmsg() de la classe SourisTcpAgent
 - Définition d'une classe fille de TclClass : lien entre la classe SourisTcpAgent (C++) et Agent/TCP/Souris (OTcl)



Évaluation des performances obtenues



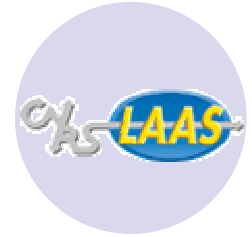
Délai d'émission



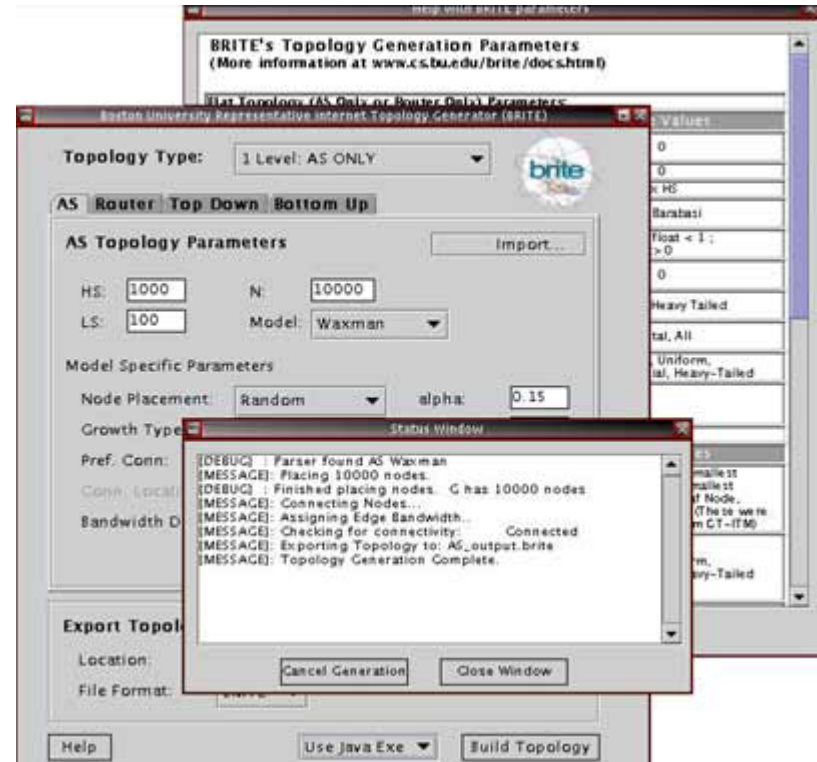
Les outils gravitant autour de NS

- Notoriété du simulateur assise
- Nombreux outils connaissent son format :
 - Outils de génération de topologies : BRITE, NEM...
 - Outils de dépouillage des traces : TraceGraph

Générateurs de topologies



- BRITE (Boston university Representative Internet Topology Generator) :
 - Génération à partir de modèles topologiques
 - Génération à partir de données réelles (niveau AS ou niveau IP)
 - GUI interface



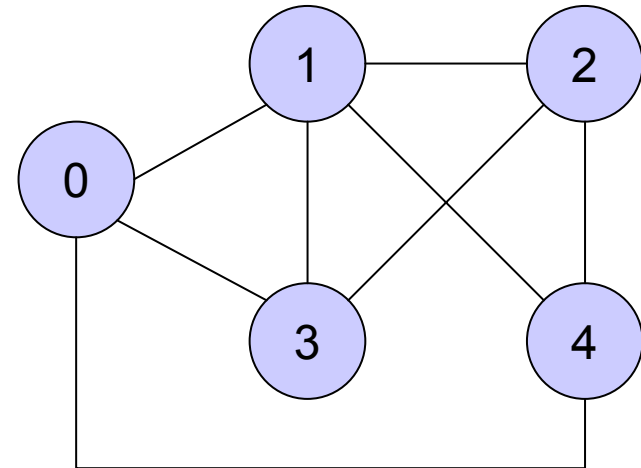
Topologie TCL générée par BRITE



```
# Export from BRITE topology
# Generator Model Used: Model (1 – Waxman)
proc create_topology{} {
  global ns
```

```
#nodes:
set num_node 5
for {set i 0} {$i < $num_node} {incr i} {
  set n($i) [$ns node]
}
```

```
#links:
set qtype DropTail
```



```
$ns duplex-link $n(0) $n(1) 0.016816047038032907Mb 0.4105142841484094ms $qtype
$ns duplex-link $n(0) $n(3) 0.309201899489568Mb 0.27717952011377456ms $qtype
$ns duplex-link $n(1) $n(2) 0.04414118488958577Mb 0.1804023673336033ms $qtype
$ns duplex-link $n(1) $n(3) 0.1700970597283317Mb 0.1390206152829549ms $qtype
$ns duplex-link $n(2) $n(4) 0.04837306996296799Mb 0.016678204759907604ms $qtype
$ns duplex-link $n(2) $n(3) 0.11482918413244239Mb 0.051025494913478964ms $qtype
$ns duplex-link $n(4) $n(1) 0.1046678115336829Mb 0.19620828713246338ms $qtype
$ns duplex-link $n(4) $n(0) 0.15615380179758687Mb 0.24912571149255316ms $qtype
```

```
} #end function create_topology
```

Générateurs de topologies (suite)



- NEM (NETwork Manipulator) (Magoni / U-Strasbourg) :
 - Génération à partir de modèles topologiques
 - Génération à partir de données réelles (niveau AS ou niveau IP)
 - Interface en ligne de commande
- **— Moins puissant que BRITE**
- **+ Plus facilement modifiable (code moins complexe)**

Topologie TCL générée par NEM



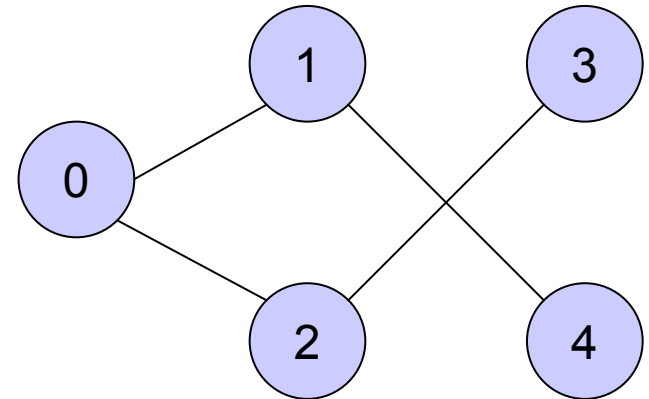
```
#-----  
# network topology generated by nem for ns-2 (Waxman model)  
# beginning of ns script must have: set ns [new Simulator]  
# loading of this <file> in ns script by: source <file>.tcl
```

```
# create nodes  
for {set i 0} {$i < 5} {incr i 1} {  
    set n($i) [$ns node]  
}
```

```
# create links  
set qtype DropTail
```

```
$ns duplex-link $n(0) $n(2) 171692kb 8ms $qtype  
$ns duplex-link $n(0) $n(1) 17995kb 20ms $qtype  
$ns duplex-link $n(1) $n(4) 31626kb 17ms $qtype  
$ns duplex-link $n(2) $n(3) 31626kb 17ms $qtype
```

```
#-----
```



Dépouillage automatisé de simulations



- Trace Graph :
 - Programme open-source basé sur Matlab 6.0
 - Nombreuses analyses possibles : délai, gigue, RTT, débit, statistiques...
 - GUI interface (graphes 2D & 3D)
 - Sauvegarde des résultats sous forme de données brutes ou d'images JPEG

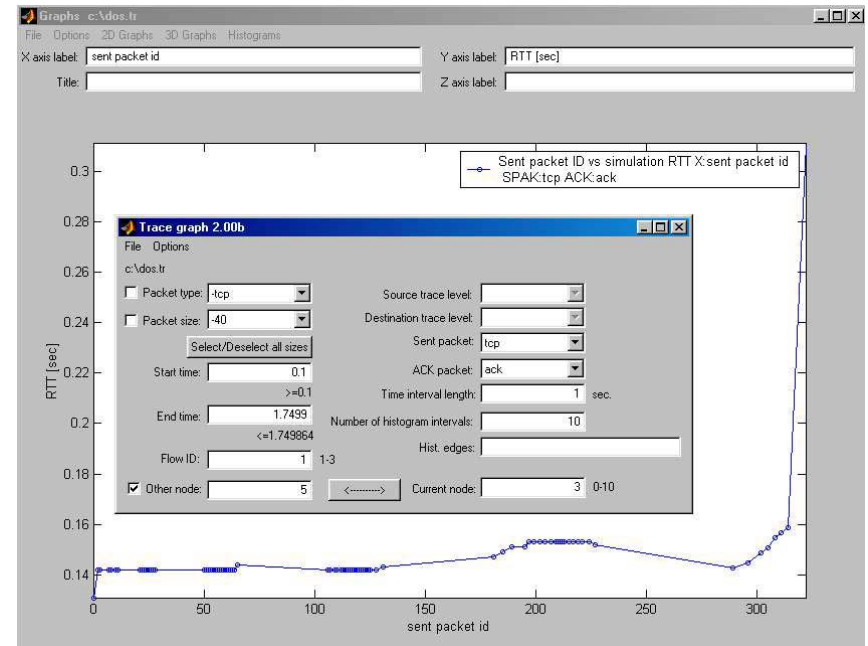


Temps de calcul important
(fonction de la taille / complexité
des simulation)

Analyse mono factorielle



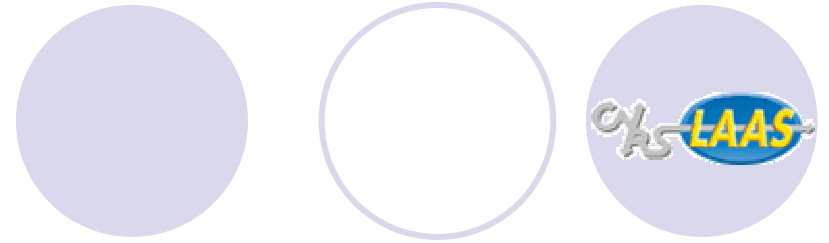
Utilisation intuitive
Résultats obtenus rapidement



Conclusion

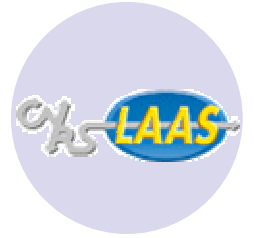
- Outil *facile* d'approche aux possibilités très évoluées
- Code open-source : garanti des possibilités d'évolution fortes et rapides
 - MAIS difficulté relativement importante pour faire évoluer les modules existants et a fortiori en développer
- Outil très répandu dans la communauté de recherche en réseau :
 - nombreux outils connexes compatibles avec le format NS
 - nombreuses architectures et protocoles supportés
 - réactivité (protocoles, modèles, sources) importante

Voir plus loin...



- Conception d'outils d'analyse homogénéisés
 - ⇒ Ex : logiciel graphique prenant en compte les spécificités de nos analyses
- Mise à jour de NAM
 - ⇒ prise en compte des nouveaux modules
 - nouveaux ordonnanceurs,
 - nouvelles versions de TCP
 - Etc.
- Fonctionnalités d'émulation de NS
 - ⇒ Cf. plateforme d'émulation du groupe

Émulation de réseau avec NS



- Fonctionnement temps réel de l'ordonnanceur NS

⊕ Utilisation directe (sans réécriture) des protocoles implémentés en C++ dans NS

Uniquement testé sous FreeBSD 2.2.5

⊖ Peu de visibilité dans les papiers concernant ce mode de fonctionnement (au profit de Dummynet)

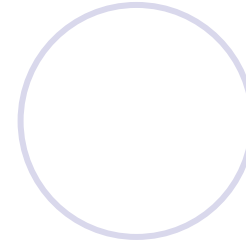
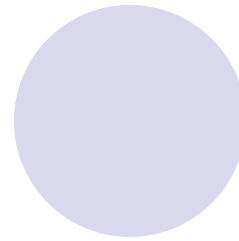
Références

- Les simulateurs réseaux :

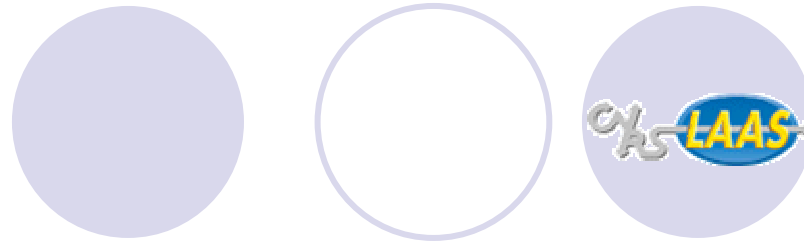
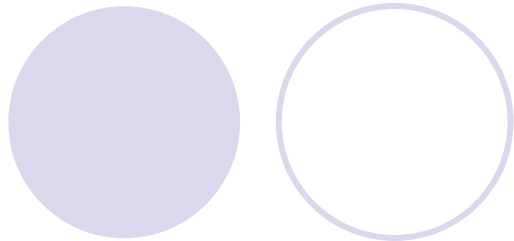
- Site web de NS-2 : www.isi.org/nsnam
- Site web de OPNET : www.opnet.com
- Site web de PARSEC : pcl.cs.ucla.edu/projects/parsec/
- Site web de SSF : www.ssfnet.org/homePage.html
- Site web de JavaSim : www.javasim.org/

- DTNS : K. Kuusilinna, J. Riihimäki, T. Hämäläinen, and J. Saarinen, *DTNS: a Discrete Time Network Simulator for C/C++ Language Based Digital Hardware Simulations*, 4th World Multiconference on Circuits, Systems, Communications and Computers, CSCC2000, Athens, Greece, Jul 10-15, 2000.

Références (suite)



- Manuels NS :
 - NS Notes and Documentation : www.isi.edu/nsnam/ns/doc/ns_doc.pdf
 - NS tutorial : John Heidemann and Polly Huang, UCLA/Institute for Pure and Applied Mathematic, March 2002, IPAM'02.
- Les articles (importants) sur NS :
 - La problématique de la simulation réseau : S. Floyd and V. Paxson, *Difficulties in Simulating the Internet*, IEEE/ACM Transactions on Networking, Vol.9, No.4, pp. 392-403, August 2001.
 - Mise en garde sur l'utilisation de NS : www.isi.edu/nsnam/ns/ns-advice.html
 - L'émulation de réseau avec NS : Kevin Fall, *Network Emulation in the Vint/NS Simulator*, ISCC July 1999
 - Page Web dédiée à l'émulation : www.isi.edu/nsnam/ns/ns-emulation.html
- Les outils en rapport avec NS :
 - Prétraitement :
 - BRITE : www.cs.bu.edu/brite/
 - NEM : ww-r2.u-strasbg.fr/nem/
 - Post traitement :
 - AWK : www.gnu.org/manual/gawk-3.1.1/gawk.html
 - Trace Graph : www.geocities.com/tracegraph/
 - L'émulation de réseau avec Dummynet : info.iet.unipi.it/~luigi/ip_dummynet/



Émulation de réseau avec NS



- Capture du trafic qui passe sur le réseau (BPF = librairie niveau liaison)
- Deux modes de fonctionnements :
 - Mode « opaque » : les champs des paquets capturés ne sont pas interprétés
 - Test de bout en bout (délai, réordonnancement, pertes)
 - Mode « protocole » : les paquets sont interprétés, modifiés et réémis dans le réseau
 - Test de nouveaux protocoles

