

TP 2: Simulation à événements discrets & Introduction à OMNeT++

I. Simulation avec votre propre échéancier/gestionnaire

Dans le TP1, nous avons créé les départs indépendamment des arrivées et en particulier indépendamment du nombre de paquets dans la file d'attente (clients en attente). Seul le premier départ a été calculé à partir du temps de la première arrivée. Ensuite, les autres départs se basaient uniquement sur l'instant du départ précédant le départ en cours. Les résultats obtenus sont corrects seulement quand la file d'attente n'est jamais vide, i.e. quand le taux d'arrivée est plus grand que le taux de service. Or, c'est plutôt la condition inverse qu'on rencontre le plus souvent car les réseaux doivent être dimensionnés correctement afin que le taux d'arrivée soit plus petit que le taux de service (ou du moins la plupart du temps).

Dans une simulation à événements discrets, les événements devraient être planifiés au fur et à mesure dans l'échéancier. C'est la liste d'événements futurs (FES – Futur Event Set or FEL – Futur Event List). Les paramètres d'état du système doivent être convenablement définis. Lors de son exécution, chaque événement peut changer l'état du système et éventuellement générer un ou plusieurs événements futurs qui seront insérés dans la FEL.

Dans cette partie du TP, nous allons implémenter la FEL, un ordonnanceur et la mise à jour de la FEL. Le système à simuler est la file d'attente à un serveur comme présenté dans le slide 11. Ce modèle est adéquat pour la simulation d'un lien de transmission. Afin de focaliser sur le déroulement de la simulation, nous allons simuler le système avec seulement 10 arrivées. Le processus d'arrivée est un processus de Poisson dont les inter-arrivées suivent la loi exponentielle de paramètre $\lambda=5$ (donc 5 arrivées/seconde en moyenne). Pour faire simple, le temps de service suit également la loi exponentielle de paramètre $\mu=5$ (donc le temps moyen de service est $S=1/5 = 0.2s = 200ms$, et le taux moyen de service est égale à 5 paquets/seconde).

Créez le répertoire `~/sev/tp2` pour ce TP (~ désigne votre dossier personnel par défaut). Dedans, créez le répertoire **echeancier** pour le travail de cette partie.

- 1) Utilisez un éditeur pour écrire votre programme **echeancier.c** :

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(int argc, char** argv){

    // Declaration de variables
    struct event{
        double time;        // temps d'occurrence
        char event_type;    // 'a' = arrivee, 'd' = depart
    };
```

```

    struct event FEL[20]; // Futur Event List
    struct event current_event;
    struct event scheduled_event;
    double SimClock; // L'horloge du simulateur
    int nb, i, j, k;

    printf("Simulation finished !\n");

    return 0;
}

```

Pour compiler le programme:

```
$ gcc -lm -o echeancier echeancier.c
```

Pour lancer le programme:

```
$ ./echeancier
```

- Dans votre rapport, expliquez brièvement la structure de données struct event et la manière d'implémenter la FEL dans ce programme.

- Combien d'événements pouvons-nous avoir dans la FEL au maximum ?

2) Après les includes et avant le main(), insérez les deux macros suivants qui implémentent deux opérations de l'ordonnanceur :

```

#define GET_NEXT_EVENT() \
    current_event = FEL[0]; /*Extraction de l'événement*/ \
    printf("-----\n"); \
    printf("Extract current event : [%c %f]\n", \
        current_event.event_type, current_event.time); \
    \
    for(i=1; i<=nb; i++){ /*Mise à jour FEL*/ \
        FEL[i-1] = FEL[i]; \
    } \
    nb--; \
    \
    printf("FEL (%d events): ", nb); \
    for(i=1; i<=nb; i++){ \
        printf("[%c %f] ", FEL[i-1].event_type, FEL[i-1].time); \
    } \
    printf("\n"); \
\
#define SCHEDULE_EVENT(e, t) \
    scheduled_event.time = t; \
    scheduled_event.event_type = e; \
    printf("event scheduled at %f\n", scheduled_event.time); \
    /* Mettre à jour la FEL en ajoutant le départ planifié */ \
    j = 0; \
    while ((scheduled_event.time >= FEL[j].time) && \
        (j <= (nb-1))){ \
        j++; \
    } \
    /* si le temps planifié est le plus grand, ajouter le */ \
    /* à la fin de la FEL */ \
    if (j == nb) { \
        FEL[nb] = scheduled_event; \
    } else { /* si non, insérer le nouvel événement entre */ \
        /* FEL[j-1] et FEL[j] */ \
        for (k=nb-1; k>=j; k--){ \
            FEL[k+1] = FEL[k]; \
        } \
        FEL[j] = scheduled_event; \
    } \
}

```

```

        nb++;
        printf("FEL (%d events): ", nb);
        for(i=1; i<=nb; i++){
            printf("[%c %f] ", FEL[i-1].event_type,
                FEL[i-1].time);
        }
        printf("\n");

```

- Recompilez.

- Quel est le rôle de la première opération ? Quel est le rôle de la deuxième opération ? Que représente la variable nb ? Même question pour la variable t ?

- 3) Maintenant que notre échéancier/ordonnanceur est programmé. Nous pouvons l'utiliser pour simuler le système mentionné ci-dessus :

Insérez le code suivant juste avant la ligne de fin (`printf("Simulation finished !\n");`). Ce code correspond aux lignes 1-7 du slide 15.

```

// suite des déclarations pour la simulation
int num_queue, num_sys;
double u, x, t, lambda, mu;
char channel_free; // 'y' = yes, 'n' = no

// Initialisation des variables
SimClock = 0; t = 0 ;
nb = 0;
lambda = 5;
channel_free = 'y';
mu = 5;
num_queue = 0; // nombre de paquets dans la file d'attente
num_sys = 0; // nombre de paquets dans le système

srandom(Votre numéro d'étudiant);

// Générer les 10 arrivées et les planifier dans la FEL
// Ici les arrivées ne dépendent pas d'autres événements
// On peut insérer toutes les arrivées directement dans la
// FEL avant la boucle principale de la simulation.
for (i=1; i<=10; i++){
    u = (double) random() / (RAND_MAX + 1.0);
    x = -log(1-u)/lambda;
    t = t + x;
    nb++;
    FEL[i-1].time = t;
    FEL[i-1].event_type = 'a';
}

// Afficher l'état initial du système
printf("Simulation starts !\n");
printf("SimClock = %f\n", SimClock);
printf("%d packets in the queue, %d packets in the system\n",
num_queue, num_sys);
printf("FEL (%d events): ", nb);
for(i=1; i<=nb; i++){
    printf("[%c %f] ", FEL[i-1].event_type, FEL[i-1].time);
}
printf("\n");

```

Compilez et lancez le programme. Copiez le résultat dans votre rapport.

- 4) Une implémentation simple de la boucle de l'ordonnancement des événements (simulation) correspond au code suivant. Ce code correspond aux lignes 8-13 du

slide 15. Insérez ce code juste avant la ligne de fin (`printf("Simulation finished !\n");`).

```
// Ordonnanceur - gestionnaire - main loop
while (nb > 0) {

    // Récupérer le premier événement et l'enlever de la FEL
    GET_NEXT_EVENT();

    // Avancer l'horloge du simulateur
    SimClock = current_event.time;
    printf("SimClock = %f\n", SimClock);

    // Exécuter l'événement courant
    if (current_event.event_type == 'a') { // si c'est une arrivée
        printf("This is an arrival.\n");
    } else {                               // si c'est un départ
        printf("This is a departure.\n");
    }
}
```

- Compilez et lancez le programme. Copiez le résultat dans votre rapport.

- Quel est le rôle de la variable `SimClock` ?

- 5) L'exécution de l'événement courant (ligne 12, slide 15) correspond à appeler la fonction `pkt_arrival()` (slide 16) en cas d'arrivée d'un paquet et à appeler la fonction `pkt_complete()` (slide 17) en cas de départ d'un paquet. Dans cette étape, nous allons implémenter la fonction `pkt_arrival()` pour traiter l'arrivée d'un paquet. Insérez directement le code suivant juste après la ligne `« printf("This is an arrival.\n"); »`. Ce code correspond au slide 16.

```
// si le canal est libre, le paquet est servi directement,
// pas dans la file d'attente
if (channel_free == 'y'){
    printf("Channel is free. Packet is served.\n");
    channel_free = 'n';
    num_sys++;
    u = (double) random() / (RAND_MAX + 1.0);
    x = -log(1-u)/mu;
    printf("Generate a service time = %f\n", x);
    printf("Departure ");

    SCHEDULE_EVENT('d', SimClock + x)

} else { // si le canal est occupé, le paquet est mis dans
    // la file d'attente
    printf("Channel is busy. Packet is queued.\n");
    num_queue++;
    num_sys = num_queue + 1;
}
printf("%d packets in the queue, %d packets in the
system\n", num_queue, num_sys);
```

- Compilez et lancez le programme. Donnez le résultat dans votre rapport.

- Expliquez brièvement l'affichage du programme et en particulier la dernière ligne affichée avant la ligne affichant `Simulation finished !`.

- 6) Dans cette étape, nous allons compléter notre programme par l'implémentation de la fonction **pkt_complete()** qui traite le départ d'un paquet. Insérez le code suivant juste après la ligne « `printf("This is a departure.\n");` ». Ce code correspond au slide 17.

```
num_sys--;
if (num_queue > 0){
    printf("Serve the next packet in the queue.\n");
    num_queue--;
    //Générer le temps de service x et planifier un départ
    u = (double) random() / (RAND_MAX + 1.0);
    x = -log(1-u)/mu;
    printf("Service time = %f\n", x);
    printf("Departure ");

    SCHEDULE_EVENT('d', SimClock + x)

    printf("%d packets in the queue, %d packets in the system\n",
num_queue, num_sys);
} else { // Si la file d'attente est vide
    printf("Queue is empty. Channel becomes free. \n");
    channel_free = 'y';
    num_sys = 0;
    num_queue = 0;
    printf("%d packets in the queue, %d packets in the system\n",
num_queue, num_sys);
}
```

- Compilez et lancez le programme. Donnez le résultat dans votre rapport.
- Expliquez en détail l'affichage du programme et décrivez en même temps le déroulement de la simulation.

II. Introduction à OMNeT++

OMNeT++ est un environnement de simulation à événements discrets orienté objet. Son architecture générique peut être utilisée dans de nombreux domaines :

- Réseaux filaires ou sans fil
- Développement de protocoles/applications réseaux
- Etude des performances de files d'attentes
- Réseaux mobiles
- Etc.

Dans OMNeT++, un réseau est modélisé par les **modules** qui se communiquent en envoyant les **messages** sur les **connexions** qui interconnectent les modules. L'envoi d'un message sur une connexion se fait via une **interface** définie dans le module. Les **modules simples** sont les modules de base. Les **modules composés** sont des modules plus complexes qui sont construits à partir des modules simples. La figure 2.1 présente le modèle réseau dans OMNeT++.

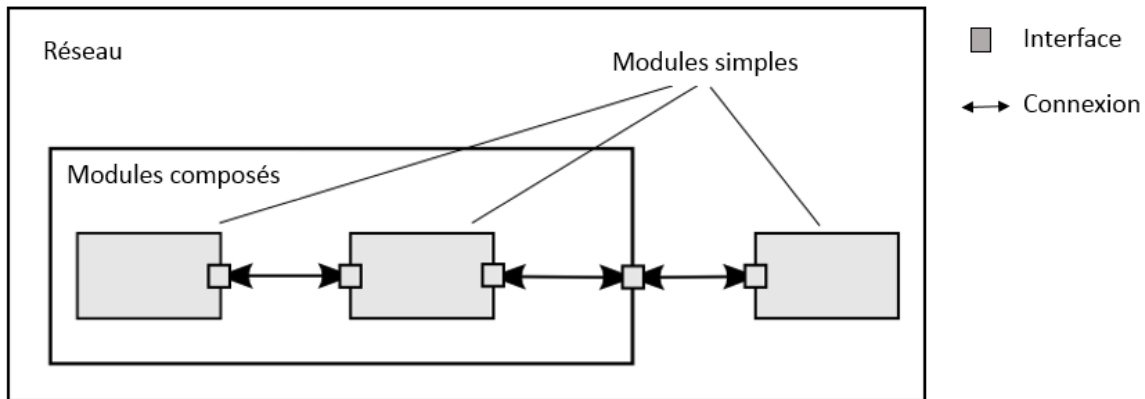


Figure 2.1 – Modèle général d'un réseau dans OMNeT++

Pour écrire et lancer une simulation, vous pouvez utiliser l'environnement de développement intégré (IDE – Integrated Development Environment) d'OMNeT++. Vous pouvez également utiliser un éditeur de texte pour écrire le programme de simulation et lancer la simulation par ligne de commande.

Ouvrez un Terminal. Lancez l'IDE avec la commande

```
$ omnetpp
```

Lorsque l'application demande de choisir un espace de travail (a workspace), choisissez le répertoire de ce TP (~/*sev/tp2*).

Lorsque l'application demande si vous voulez installer ou importer INET et les exemples, cliquez sur « Annuler ».

Depuis l'écran d'accueil d'OMNeT++, cliquez sur Workbench. Vous êtes dans l'IDE d'OMNeT++.

Si vous êtes trompé dans le choix de l'espace de travail, vous pouvez toujours choisir l'espace de travail depuis l'IDE. Allez dans **File** → **Switch workspace** → choisissez le répertoire ~/*sev/tp2*

III. Simulation par l'IDE d'OMNeT++

Dans cette partie, nous allons simuler un réseau de 2 nœuds interconnectés. Lors du démarrage, chaque nœud envoie à l'autre nœud un message « *data* ». Lors de la réception du message, chaque nœud écrit dans la fenêtre de log « *<nom_du_noeud> a reçu des données* ».

- 1) Créez votre nouveau projet nommé **reseauTP2**. Allez à **File** → **New** → **OMNeT++ project** : **Project name** = **reseauTP2** ensuite **Next** et choisissez **Empty Project** ensuite **Finish**
- 2) Créez un module simple qui va modéliser un nœud. Allez à **File** → **New** → **Simple Module** : **Enter or select the parent folder** = **reseauTP2** ; **File name** = **noeud.ned** ensuite **Next** → **Select template** = **A simple module** ensuite **Finish**.

- Quels sont les 3 nouveaux fichiers créés par OMNeT++ pour ce module ?

- 3) Personnalisez l'image de votre nœud en **faisant un clic droit** sur l'icône actuelle puis sur **Properties** ensuite dans l'onglet **Appearance**, choisissez l'image suivante :



Ensuite appuyez sur OK.

- 4) Allez dans le code source de votre module (fichier **noeud.ned** en cliquant sur l'onglet **Source** de la fenêtre du bas). Nous allons déclarer deux interfaces unidirectionnelles : une pour la transmission (de type **output**) et une pour la réception (de type **input**). Ajouter les lignes suivantes dans le bloc de codes (à l'intérieur des accolades) du module simple **Nœud**.

```
gates:  
    input in;  
    output out;
```

Dans votre rapport, donnez la capture d'écran de l'onglet « Design » et le code source (l'onglet « Source ») de votre module simple **Nœud**.

- 5) Nous avons défini notre nœud. Nous allons maintenant définir notre réseau ayant 2 nœuds. Allez à **File** → **New** → **Network** ensuite sélectionnez le projet « **reseauTP2** », **File name** = **reseau.ned** → **Next** ; **Select template** = **An empty network** → **Finish**.

Quel est le nouveau fichier créé par l'OMNeT++ ?

- 6) Personnalisez l'image de votre réseau en choisissant l'image suivante :
- 7) Nous allons utiliser l'interface graphique pour ajouter 2 nœuds dans notre réseau. Allez dans la « **Palette** » et choisissez le sous-module **Nœud**. Cliquez sur notre réseau vide pour ajouter un nœud. Nommez le « **noeud1** ». Ajoutez le deuxième nœud de la même manière et le nommez « **noeud2** ».
- 8) Nous allons maintenant créer un lien bi-directionnel entre deux nœuds. Dans la palette, choisissez « **Connection** » → cliquez sur le **noeud1** → cliquez sur le **noeud2** → choisissez le code proposé « **noeud1.out --> noeud2.in** ». De la même manière, choisissez « **Connection** » dans la Palette → cliquez sur le **noeud2** → cliquez sur le **noeud1** → choisissez le code proposé « **noeud2.out--> noeud1.in** »

Dans votre rapport, donnez la capture d'écran de l'onglet « Design » et le code source (l'onglet « Source ») de votre réseau.

Remarque : Bien sûr tout cela peut être fait en créant directement les fichiers et en les éditant manuellement sans passer par l'IDE.

- 9) Jusqu'à présent, nous avons créé un réseau de deux nœuds mais les nœuds ne font rien. Pour implémenter le comportement du nœud lors du démarrage, allez dans le code source du fichier *nœud.cc* et repérez la méthode *initialize()* qui est invoquée lorsque OMNeT++ crée le réseau. Pour que notre nœud envoie un message « data » lors de son démarrage, nous allons créer un message de type *cMessage* et l'envoyer sur l'interface de sortie du nœud en utilisant la fonction *send()*. Ajouter les lignes suivantes dans la méthode *initialize()* :


```
cMessage *msg = new cMessage("data");  
send(msg, "out");
```

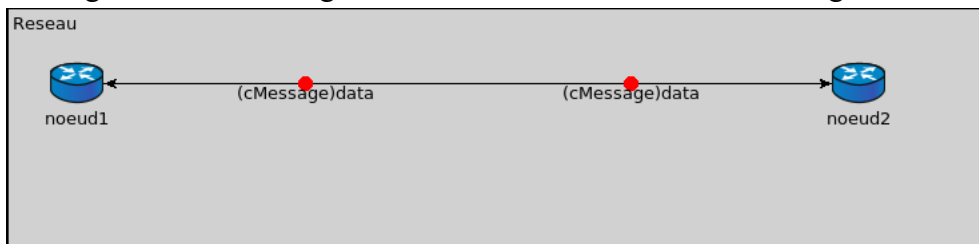
- 10) Pour implémenter le traitement d'un message reçu, allez dans le code source du fichier *nœud.cc* et repérez la méthode *handleMessage()* qui est invoquée lorsque le nœud reçoit un message. Nous allons utiliser le flux *EV* pour afficher un texte dans la fenêtre de logs et la fonction *getName()* pour obtenir le nom du nœud. Insérez le code suivant dans la méthode *handleMessage()* :

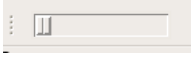
```
EV << getName() << " a reçu des données." << std::endl;
```

Dans votre rapport, donnez le code source du programme *nœud.cc*

- 11) Vous pouvez maintenant compiler (**Project → Build Project**) votre projet et lancer (**Run → Run → Run as OMNeT++ Simulation → OK**) votre simulation. Lorsque l'IDE demande si vous voulez créer le fichier de configuration .ini, acceptez la proposition.

- 12) Dans l'interface de simulation, cliquez sur le bouton **Run** . Avez-vous vu les messages « data » échangés entre les nœuds lors de son démarrage ?



Vous pouvez ralentir ou accélérer la simulation avec  et rejouer la simulation avec [Rebuild network].

Avez-vous vu les textes « <nom_du_noeud> a reçu des données. » dans la fenêtre de logs ?

```

** Initializing network
Initializing module Reseau, stage 0
Reseau.noeud1: Initializing module Reseau.noeud1, stage 0
Reseau.noeud2: Initializing module Reseau.noeud2, stage 0
** Event #1 t=0 Reseau.noeud2 (Noeud, id=3), on `data' (cMessage, id=12)
noeud2 a reçu des données.
** Event #2 t=0 Reseau.noeud1 (Noeud, id=2), on `data' (cMessage, id=14)
noeud1 a reçu des données.
<!-- No more events -- simulation ended at event #3, t=0.
** Calling finish() methods of modules

```

Dans votre rapport, donnez une capture d'écran de votre simulation et une capture d'écran de la fenêtre de logs à la fin de votre simulation.

Question : Pourquoi `t` est toujours égale à 0 dans l'affichage précédent ?

IV. Simulation par lignes de commande

Dans cette partie, nous allons réaliser le réseau Tic-Toc (slide 28 du cours). Cette simulation est faite uniquement par un éditeur de texte et des lignes de commande.

- 1) Dans le répertoire `~/sev/tp2`, créez un dossier **tictoc** pour notre simulation (slide 29 du cours).

```

$ cd ~/sev/tp2
$ mkdir tictoc
$ cd tictoc

```
- 2) Utilisez un éditeur de texte (e.g. **gedit**) pour écrire le fichier **tictoc1.ned** (slide 30). Dans ce fichier, nous utilisons le langage NED pour décrire notre nœud, **Txc1**, qui est un module simple, et notre réseau, **Tictoc1**, qui a deux nœuds – **tic** et **toc** – de type **Txc1**. Ces deux nœuds sont interconnectés par une liaison bidirectionnelle ayant un délai de 100 ms.
- 3) Nous avons créé un réseau de deux nœuds mais les nœuds ne font rien. Pour implémenter le comportement du nœud, utilisez un éditeur de texte et écrivez le fichier **txc1.cc** (slide 32). Vous allez retrouver ici les méthodes **initialize()** et **handleMessage()** expliquées dans la section précédente. Lors du démarrage, Tic envoie un message à Toc. Lors de la réception du message, chaque nœud renvoie simplement le message à l'autre nœud. Vous retrouvez également les fonctions **getName()**, **send()**, et la manière de créer un message comme nous avons fait dans l'exemple précédent.
- 4) Les paramètres de simulation (e.g. le réseau à simuler, la durée de la simulation) sont configurés dans un fichier **.ini**. Utilisez l'éditeur de texte pour créer le fichier **omnetpp.ini** avec deux lignes de codes comme indiquées dans le slide 34.
- 5) Compiler et lancer la simulation avec les lignes de commandes indiquées dans le slide 35.

A mettre dans votre rapport les codes source des fichiers **tictoc1.ned**, **txc1.cc**, **omnetpp.ini** et une capture d'écran de votre simulation. ■

Procédure d'envoi du compte rendu du TP

Le rapport devra être remis sous format électronique en pdf seulement au plus tard 15mn après le TP.

- *Le fichier est nommé suivant le format SEV19_TP2_NOM1_NOM2.PDF*
- *Sur la page de garde doivent figurer vos prénoms, noms, numéros d'étudiant et le numéro ainsi que l'intitulé du TP.*
- *Reportez dans votre rapport le numéro de la question où du test à effectuer. Toutes les questions de ce TP nécessitent une réponse (généralement une ou quelques phrases suffisent) avec une capture d'écran uniquement si nécessaire.*
- *Les codes sources et/ou commandes Shell doivent être commentés correctement. Ils doivent être écrits en police à pas fixe (par exemple Courier)*
- ***Il est à rendre à la fin du TP par email (david.cordova@lip6.fr)***
- ***Les rapports qui ne vérifient pas ces consignes seront pénalisés. En cas de plagiat, les étudiants concernés auront la note globale de 0/20.***