Département Informatique et Mathématiques Appliquées

Projet Long 2008

ADMINISTRATION AUTONOME DE SERVEURS SUR LA GRILLE AVEC UNE MACHINE VIRTUELLE

Rapport de tests

Responsable : Daniel Hagimont - Professeur INPT/ENSEEIHT - Daniel.Hagimont@enseeiht.fr

Co-encadrant : Laurent Broto – Etudiant en thèse à l'UPS - Laurent.Broto@irit.fr

Superviseur industriel : Emmanuel Murzeau - emmanuel.murzeau@airbus.com

Chef de projet : Ezequiel Geremia - ezequiel.geremia@etu.enseeiht.fr

Etudiants :

- Julien Louisy
- Julien Clariond
- Hery Randriamanamihaga
- Ezequiel Geremia
- Mathieu Giorgino

Sommaire

1 Préliminaires

1.1 Notations

La campagne de tests est basée sur les scénarios présentés dans le plan de tests. Afin d'assurer la traçabilté des exigences validées, chaque test fait l'objet d'un rapport qui décrit son déroulement et les conclusions qui peuvent en être tirées. De plus, la technologie mise en œuvre lors du test profitera au rapport sous la forme d'une quantification des performances.

Les programmes de tests seront intégrés au rapport de manière à offrir une méthode standard de validation. Le résultat de ces programmes seront analysés automatiquement afin d'assurer l'objectivité de l'adéquation entre résultats obtenus et attendus.

Les notations suivantes seront utilisées pour les domaines Xen :

- *dom0* : système d'exploitation hôte associé à Xen s'exécutant sur une machine physique;
- *domU* : système d'exploitation porté sur Xen s'exécutant sur une machine virtuelle.

Ces deux notations seront suivies de *_<nom-machine-physique>* pour indiquer sur quelle machine physique les domaines s'exécutent.

Les commandes shell suivantes seront utilisées :

- `watch` : lance l'exécution synchrone de la commande donnée en argument;
- `ping` : sollicite une réponse d'un interlocuteur distant;
- `date` : renvoie la date.

Les commandes Xen accessibles à partir de tout *dom0* seront utilisées :

- `xm create` : crée un *domU* sur le *dom0* local à partir d'un fichier de configuration;
- `xm list` : renvoie la liste des domaines s'exécutant localement;
- `xm migrate` : lance la migration d'un *domU* vers un *dom0*.

Les fichiers de configuration de *domU* utilisés sont les suivants, chaque fichier de configuration permet de un nouveau *domU* :

- *etch-1-?* : correspond à un noyau Debian Etch;
- *dapper-1-1* : correspond à un noyau Debian Dapper Drake.

Les applications utilisées sont les suivantes :

- RMIServer : RMIServer.jar, RMICommon.jar, security.policy, server.sh;
- RMIClient : RMIClient.jar, RMICommon.jar, security.policy, client.sh;
- RMIPing : RMIPing.jar, RMICommon.jar, security.policy, RMIPing.sh;
- Ping : Ping.jar.

1.2 Configuration initiale

Seul le parc des machines *a-204-02* jusqu'à *a-204-06*, connectées au même réseau local, est utilisé

dans la campagne de tests. Le serveur NFS hébergé par la machines *a-204-06* offre aux *dom0_a-204-0[2345]* le moyen logiciel d'accéder les systèmes de fichiers des *domU* manipulés. De plus, *a-204-06* héberge un serveur d'horloge et un serveur DNS pour assurer la synchronisation entre *dom0_a-204-0[2345]* et permettre une désignation de haut niveau pour une meilleure intégration à TUNe. Enfin, un serveur DHCP hébergé par *a-204-06* est utilisé pour l'attribution des adresses IP et l'initialisation de l'annuaire de noms DNS.

La séquence des commandes suivante résume les tâches de configuration effectuées automatiquement au démarrage des*dom0_a-204-0[2345]* :

- récupération de l'adresse IP : `# dhclient eth0`
- synchronisation de l'horloge : `# ntpdate -u server`
- monter le répertoire distant : `# mount server:/xen/ /xen/`
- démarrage du daemon xen : `# xend start`

2 Rapport de test

2.1 Test des propriétés du mécanisme de migration

R-T1	*Énoncé du test*	Migration d'un *domU* entre deux *dom0*s.
	Programmes utilisés	`watch`, `xm create`, `xm list`, `xm migrate`.
	Noyaux utilisés	*etch-1-1.*

	Déroulement du test	*Préliminaire :* les *dom0_a-204-02 et dom0_a-204-03* exécutent le script suivant tout au long de l'expérience pour lister les domaines sur chaque *dom0* :
		`$ watch -n1 xm list`
		l'option `-n` indique que la fréquence de la mise à jour, ici 1 seconde.
		État initial (cf. Figure 2.1.1) : sur le *dom0* qui s'exécute sur *a-204-02*, la commande suivante est lancée pour créer le *domU etch-1-1* :
		`$ xm create /xen/group1/conf/etch-1-1.cfg`
		`/xen/group1/conf/etch-1-1.cfg` est le chemin vers le fichier de configuration du *domU etch-1-1*.
		État transitoire (cf. Figure 2.1.1) : l'utilisateur se trouve sur le *dom0* qui s'exécute sur *a-204-02*. La migration du *domU etch-1-1* du *dom0_a-204-02* vers *dom0_a-204-03* est effectuée grâce à la commande :
		`$ xm migrate --live etch-1-1 a-204-03`
		le drapeau `--live` indique au daemon *xend* de continuer l'exécution du *domU* qui subit la migration.
		État final (cf. Figure 2.1.1) : le résultat obtenu correspond au résultat attendu.

	Succès	Oui

Figure 2.1.1. État initial. *Le dom0 sur la machine physique a-204-02 contient un domU appelé etch-1-1 et le dom0 sur la machine a-204-03 ne contient aucun domU.*

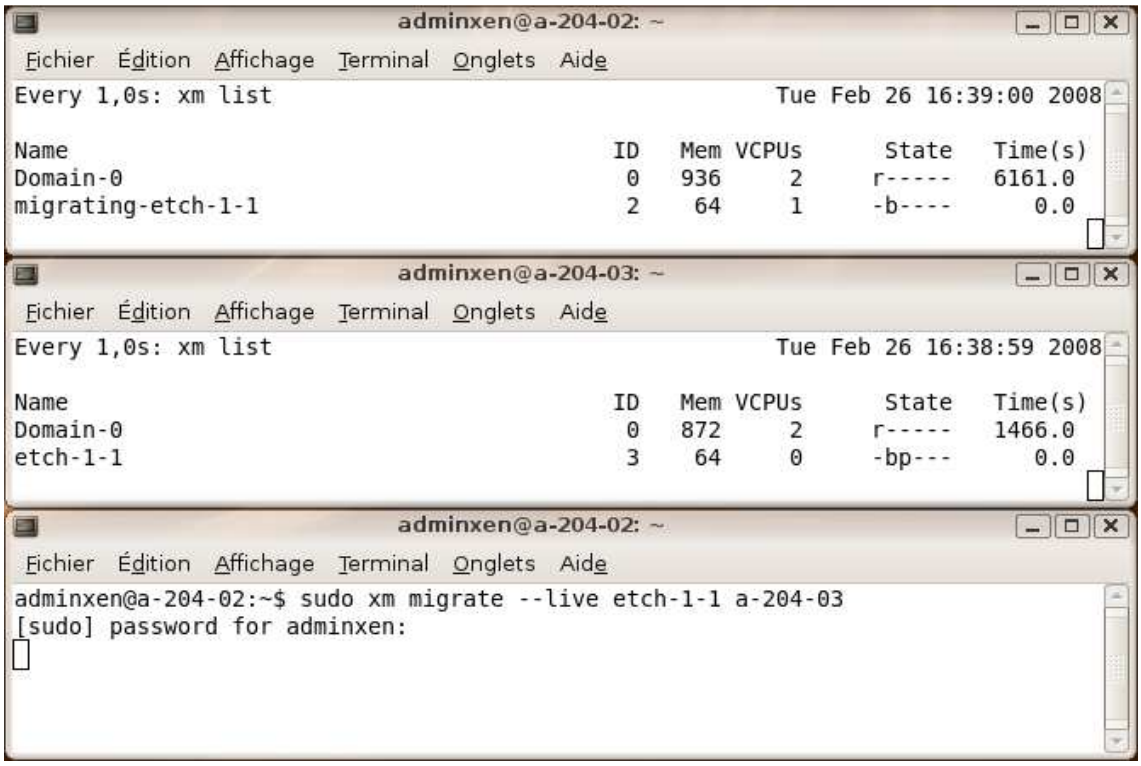


Figure 2.1.2. État transitoire. *Le domU etch-1-1 est en cours de migration du dom0 hébergé par a-204-02 vers le dom0 hébergé par a-204-03.*

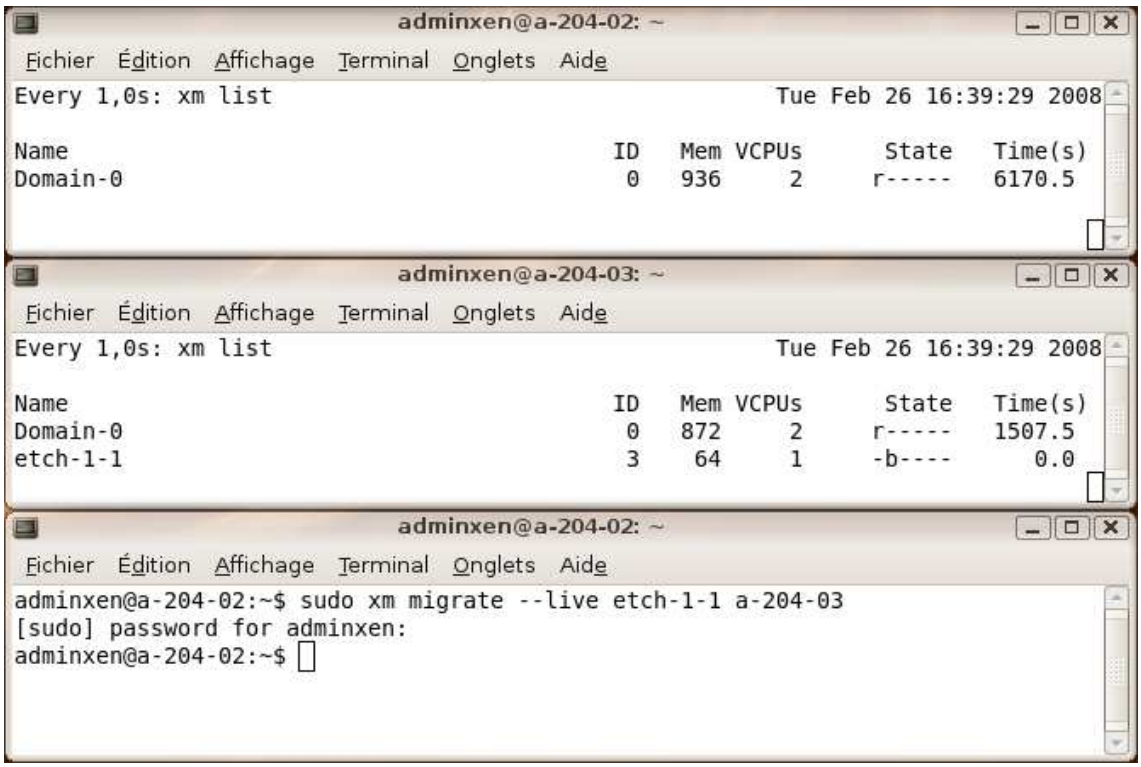
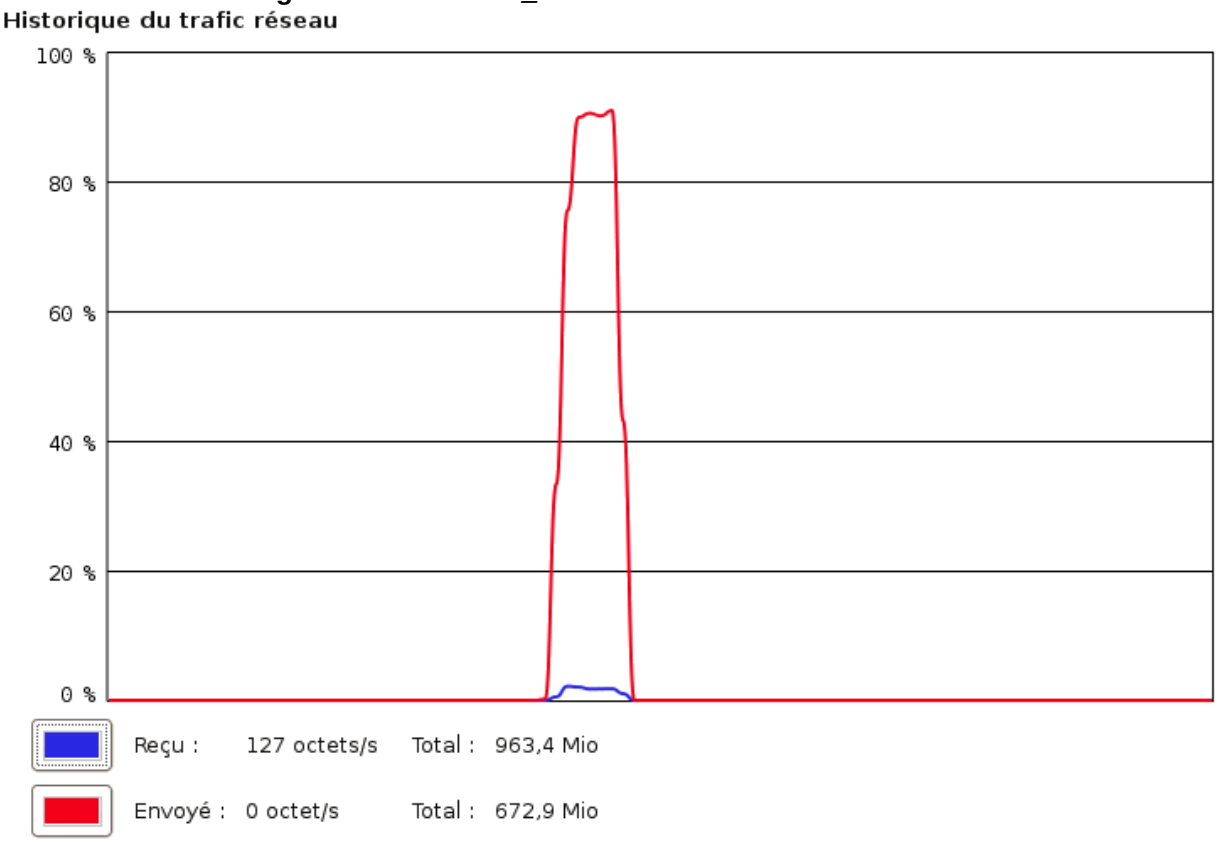


Figure 2.1.3. État final. *Le dom0 sur la machine physique a-204-03 contient un domU appelé etch-1-1 et le dom0 sur la machine a-204-02 ne contient aucun domU.*

Mesure de la charge réseau sur *dom0_a-204-02* lors de l'envoi du *domU* :



Mesure de la charge réseau sur *dom0_a-204-03* lors de la réception du

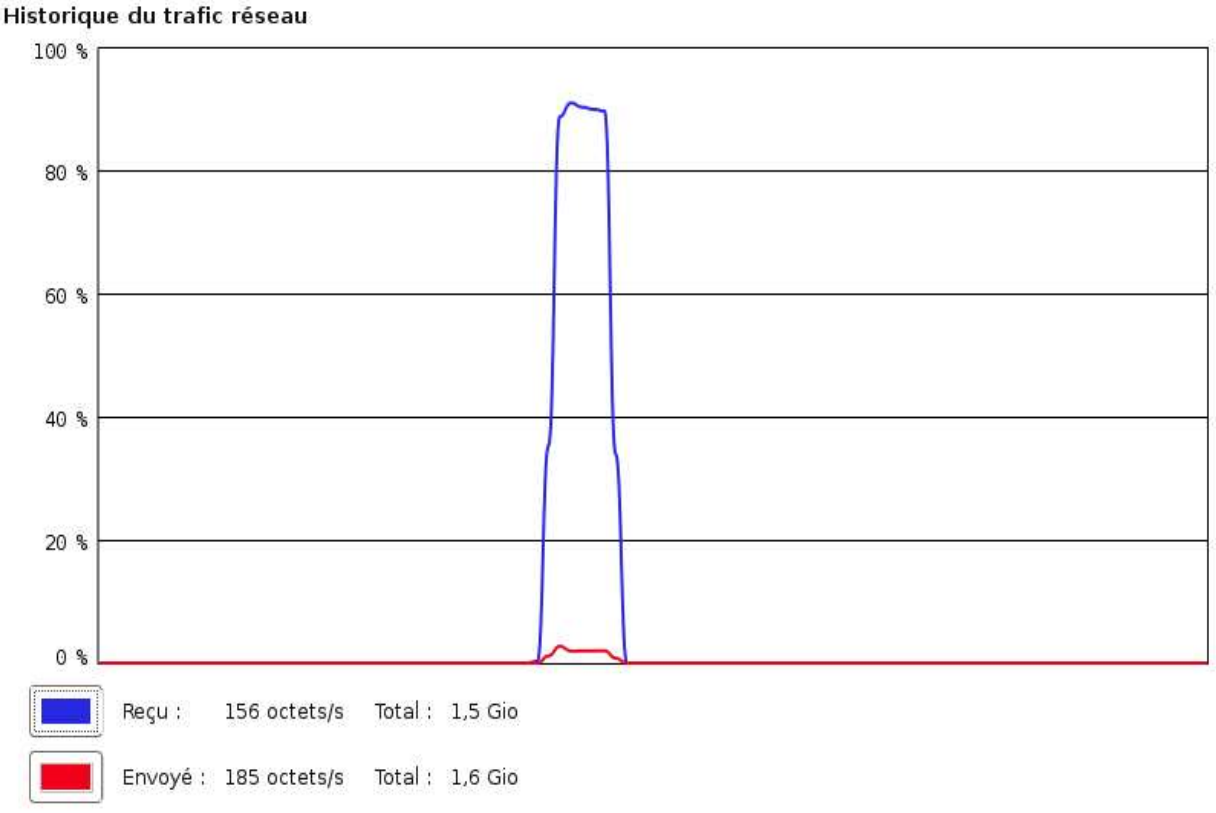


Figure 2.1.4. Mesure de la charge réseau due à la migration d'un domU entre deux dom0s. *Le*

graphique du haut représente la charge réseau en fonction du temps mesurée sur la machine a-204-02 qui envoie le domU. Celui du bas correspond à la machine a-204-03 qui reçoit le domU. La charge symétrique observée de part et d'autre de la migration correspond à la somme de deux phénomènes. Le premier phénomène est prépondérant et correspond au domaine du domU qui est envoyé par le réseau entre les deux machines a-204-02 et a-204-03. Le deuxième correspond à l'envoi par a-204-03 des accusés de réception des données codant le domU; ces derniers sont reçus par a-204-02.

Problème	Après la migration du *domU* du *dom0_source* vers le *dom0_destination*, le *domU* est bloquée. Plus précisément, l'horloge du *domU* en question est bloquée. Le même problème peut aussi se manifester par une désynchronisation de l'horloge du *domU* lors de la migration.
Cause	Les horloges des *dom0*s entre lesquels s'opère la migration ne sont pas synchronisées.
Solution	Synchroniser les horloges sur chaque *dom0*s. Un serveur NTP (Network Time Protocol) connecté au réseau peut être utilisé pour fournir une référence temporelle à tous les *dom0*s et *domU*s connectés.

R-T2	*Énoncé du test*	Migration d'un serveur de streaming vidéo s'exécutant sur un *domU* entre deux *dom0*s.
	Programmes utilisés	`watch`, `xm create`, `xm list`, `xm migrate`. L'application *vlc* associée au script `streaming.sh` seront utilisés comme pour lancer le serveur de streaming vidéo sur le protocole HTTP. Les appels RMI seront réalisés grâce aux tarballs *RMIServer*, *RMIClient*, *RMIPing*.
	Noyaux utilisés	*dapper-1-1*.
	Déroulement du test	*Préliminaires :* installer *java-1.6-jre*, *vlc (Video Lan Controller)* et créer le *domU dapper-1-1* sur *dom0_a-204-03 par la commande suivante.* *État initial :* sur le *domU dapper-1-1 hébergé* par *dom0_a-204-03 ,* lancer le serveur de streaming vidéo à l'aide la commande suivante : `# . /home/video/streaming.sh` les paramètres du streaming sont précisésdans le script `streaming.sh`. Le serveur de streaming est accédé par un client situé sur dom0_a-204-05 (cf. Figure 2.4.5). *État transitoire :* une première migration du *domU dapper-1-1* est opérée de *dom0_a-204-03* vers *dom0_a-204-02*, le client n'est pas déconnecté. Le client n'est pas déconnecté et la vidéo suit son déroulement normal. Après 10 secondes d'attente, une seconde migration du *domU dapper-1-1* est opérée de *dom0_a-204-02* vers *dom0_a-204-03*. Le client n'est pas déconnecté et la vidéo suit son déroulement normal. *État final :* le système est dans la même configuration qu'à l'état initial à ceci près que la vidéo est plus avancée dans le temps. Étant donné que le protocole de migration est HTTP, lui même basé sur TCP, le test permet de conclure que les connections TCP sont conservées lors de la migration.
	Succès	Oui

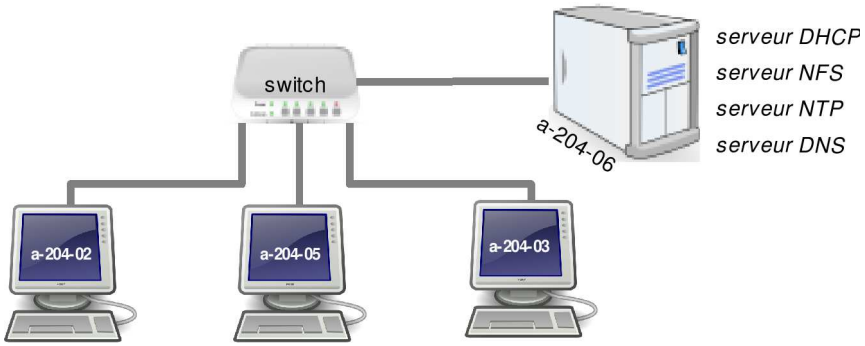


Figure 2.4.5. Architecture matérielle associée au test T2. *La machine physique a-204-06 fait office de serveur qui organise le fonctionnement du réseau.*

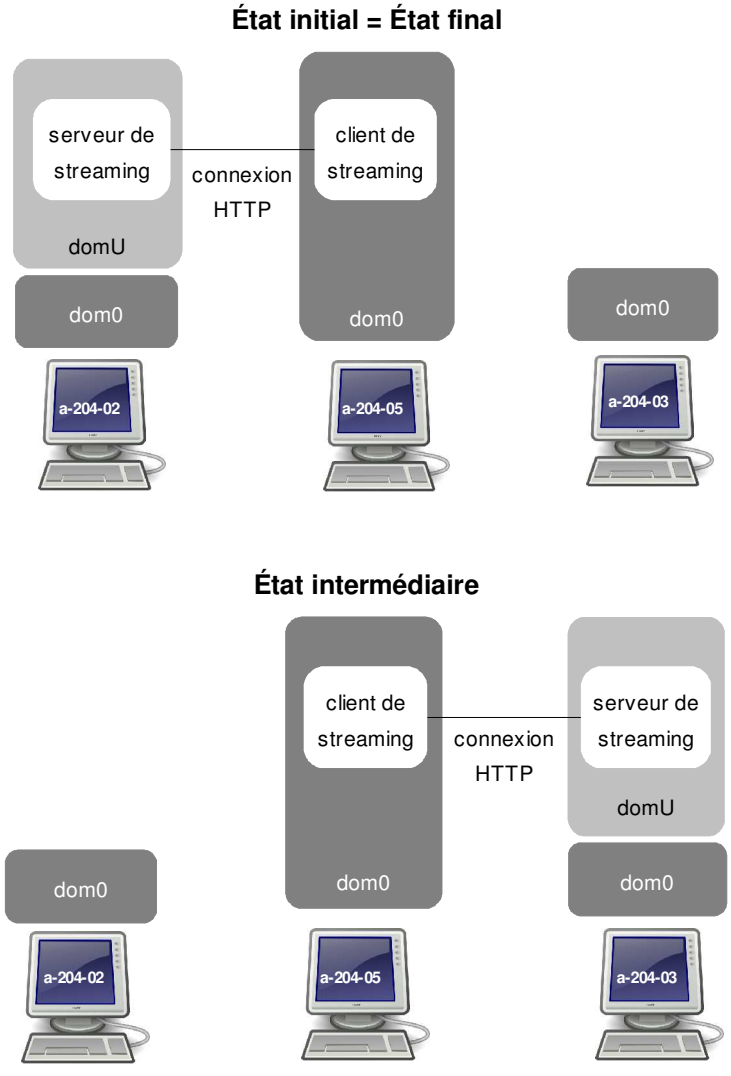


Figure 2.4.6. Scénario du test T2. *La migration du domU est opérée de dom0_a-204-02 vers dom0_a-204-03 puis de dom0_a-204-03 vers dom0_a-204-02. Le scénario sera observé à partir de dom0_a-204-03 par l'intermédiaire d'une connexion SSH (protocole Secure SHell).*

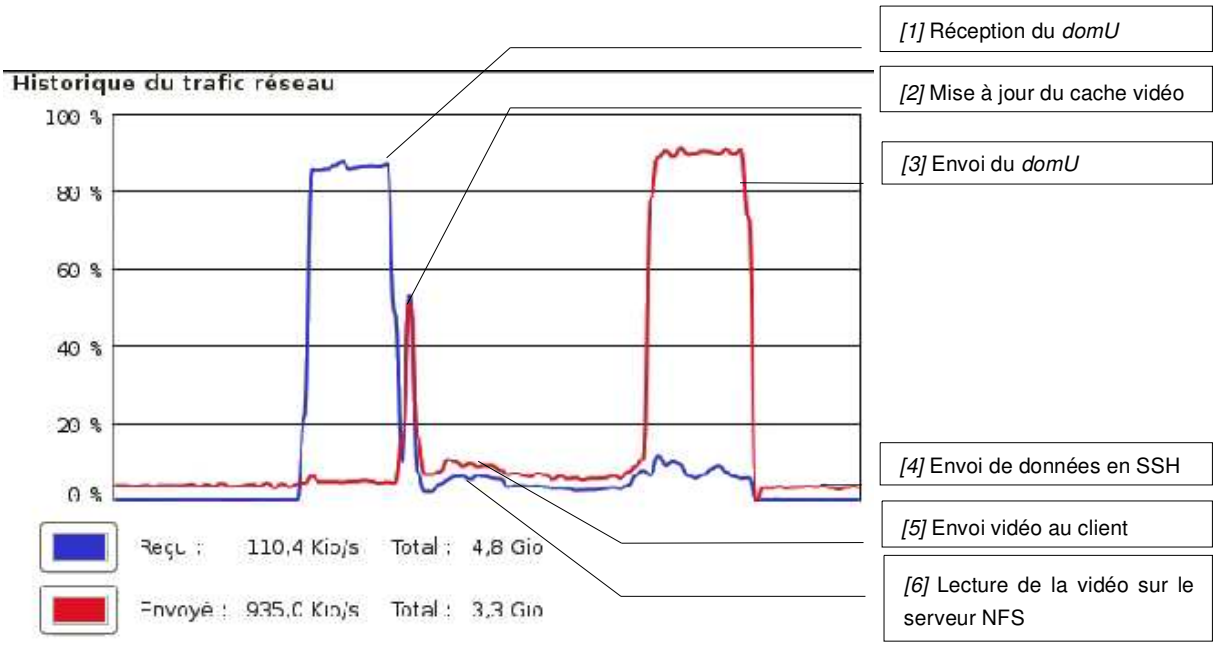


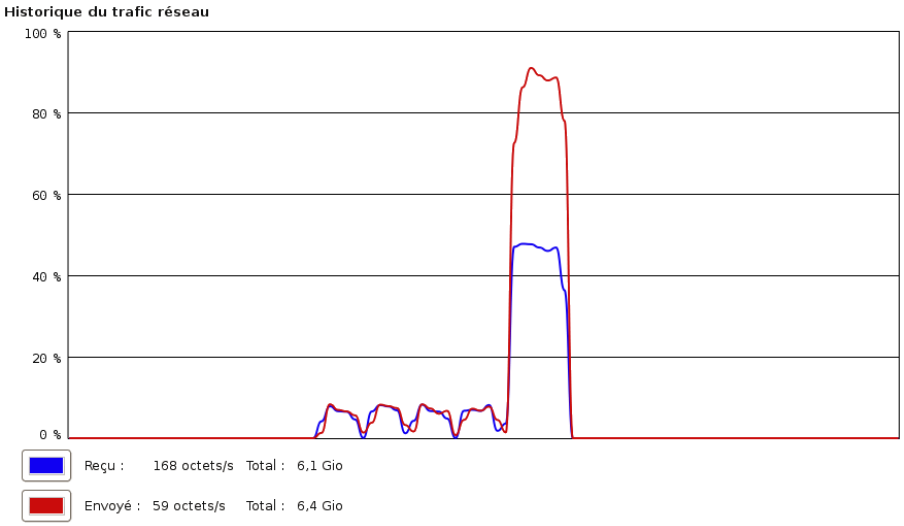
Figure 2.4.7. Activité réseau sur dom0_a-204-03 pendant la migration du domU sur lequel s'exécute le serveur de streaming vidéo. *[1] Le débit réseau le plus élevé est dû à la réception du domU et [3] à son envoi. [2] Le pic enregistré après la réception du domU s'explique par la lecture sur le serveur NFS du cache sauvé par le client, alors que le domU se trouvait encore sur dom0_a-204-02, et à la restitution de ce cache au client. [3] L'envoi du domU vers dom0_a-204-02 est conjugué à l'envoi de la vidéo vers le client, à savoir dom0_a-204-05. De même, la communication avec le serveur NFS est à considérer comme la somme des données vidéo reçus dans le but d'une transmission au client, preuve que le domU s'et des accusés de réception associés à la sauvegarde de l'état du domU. [4] Une connexion SSH permet de suivre l'évolution de l'activité réseau sur dom0_a-204-03. C'est pourquoi le flux de données émis ne passe jamais sous un certain seuil correspondant au flux de la connexion SSH. [5] Le système de fichiers du domU est situé sur le NFS, donc il lui correspond un accès réseau lors de la réception du domU par dom0_a-204-03 puis [6] l'envoi des données lues au client.*

R-T3	*Énoncé du test*	Programme synchrone écrivant dans un fichier durant s'exécutant sur un *domU* subissant une migration entre deux *dom0*.
	Programmes utilisés	`watch`, `xm create`, `xm list`, `xm migrate`. L'écriture d'une suite arithmétique de raison 1 dans un fichier est assuré par le script shell `t3w.sh`. Le script shell `t3r.sh`, quant à lui, vérifie que le fichier ne contient aucun saut, auquel cas l'écriture est considérée comme valide. Ces deux scripts doivent être contenus dans le système de fichiers du *domU etch-1-1*.
	Noyaux utilisés	*etch-1-1*.

	Déroulement du test	*État initial :* le *domU etch-1-1* s'exécute sur le *dom0_a-204-02*. Le script `t3r.sh` est lancé sur le *domU etch-1-1* par la commande :
		`# . t3w.sh out.txt`
		le fichier de sortie `out.txt` est spécifié en paramètre.
		État transitoire : une migration du *domU ecth-1-1* est opérée de *dom0_a-204-02* vers *dom0_a-204-03*.
		État final : le *domU etch-1-1* s'exécute sur *dom0_a-204*-03. Le processus d'écriture dans un fichier est interrompu. La commande suivante suivante lance la vérification du fichier `out.txt` par le script `t3r.sh` :
		`# . t3r.sh out.txt`
		l'exécution du programme d'écriture `t3w.sh` est validé. La migration du *domU etch-1-1,* sur lequel s'exécutait `t3w.sh`, n'a eu aucun effet sur l'exécution de `t3w.sh`.

	Succès	Oui

[1] Activité réseau sur dom0_a-204-02



[2] Activité réseau sur dom0_a-204-03

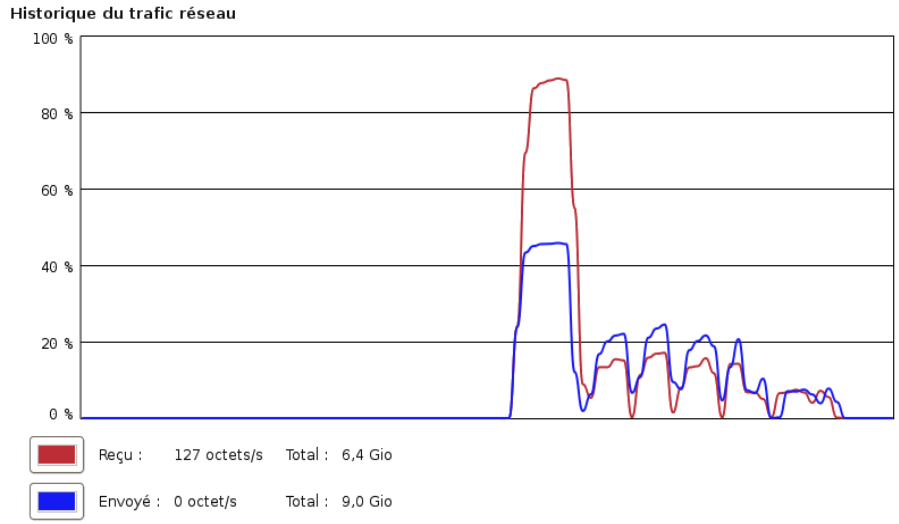


Figure 2.4.8. Activité réseau liée à l'exécution du programme d'écriture dans un fichier situé sur un serveur NFS. *Conformément à l'architecture présentée à la Figure 2.4.5, le système de fichier du domU etch-1-1 est hébergé par un serveur NFS. [1] Sur dom0_a-204-02 avant la migration, la communication en créneau enregistrée correspond à l'écriture dans le fichier situé sur le serveur NFS distant. [2] Sur dom0_a-204-03, la même forme d'activité est enregistrée sur le réseau et elle disparaît totalement sur dom0_a-204-03. Ceci s'explique par le fait que le programme d'écriture dans le fichier s'exécute sur le domU qui a subi une migration de dom0_a-204-02 vers dom0_a-204-03. L'activité sous forme de créneau provient du fait que les données à écrire sont d'abord enregistrées dans un buffer, dans la mémoire locale contenue dans le domU et donc gérée au niveau du dom0 correspondant, puis envoyées pour mettre à jour le fichier sur le serveur NFS.*

R-T4	*Énoncé du test*	Migration d'un domU d'un *dom0_source* vers un *dom0_destination* à différents niveaux de charge processeur du *dom0_source*.
	Programmes utilisés	`watch`, `xm create`, `xm list`, `xm migrate`. Le programme Java *LoadControl* a l'interface suivant : java -jar LoadControl <target> <CPU number> <duration (in seconds)>.
	Noyaux utilisés	*dapper-1-1*.
	Déroulement du test	*État initial :* le *domU dapper-1-1* s'exécute sur le *dom0_a-204-02*. Le programme Java *LoadControl* est lancé sur le *domU dapper-1-1* par la commande : `# java –jar LoadControl 4 0 10` le lancement de ce programme provoque une augmentation de la charge processeur `0` jusqu'à la valeur `4` pendant `10` secondes. *État transitoire :* lancer le programme $\mathcal{P}_k$ (cf. *T4* in *Plan de tests*) sur le *domU dapper-1-1. A*vant la fin de l'exécution de *LoadControl* et du $\mathcal{P}_k$, lancer la migration du *domU dapper-1-1 de dom0_a-204-02 vers dom0_a-204-03*. *État final :* pour $\mathcal{P}_1$ aucun client connecté au serveur n'est déconnecté, pour $\mathcal{P}_2$ les fichiers produits sont valides.
	Succès	Oui

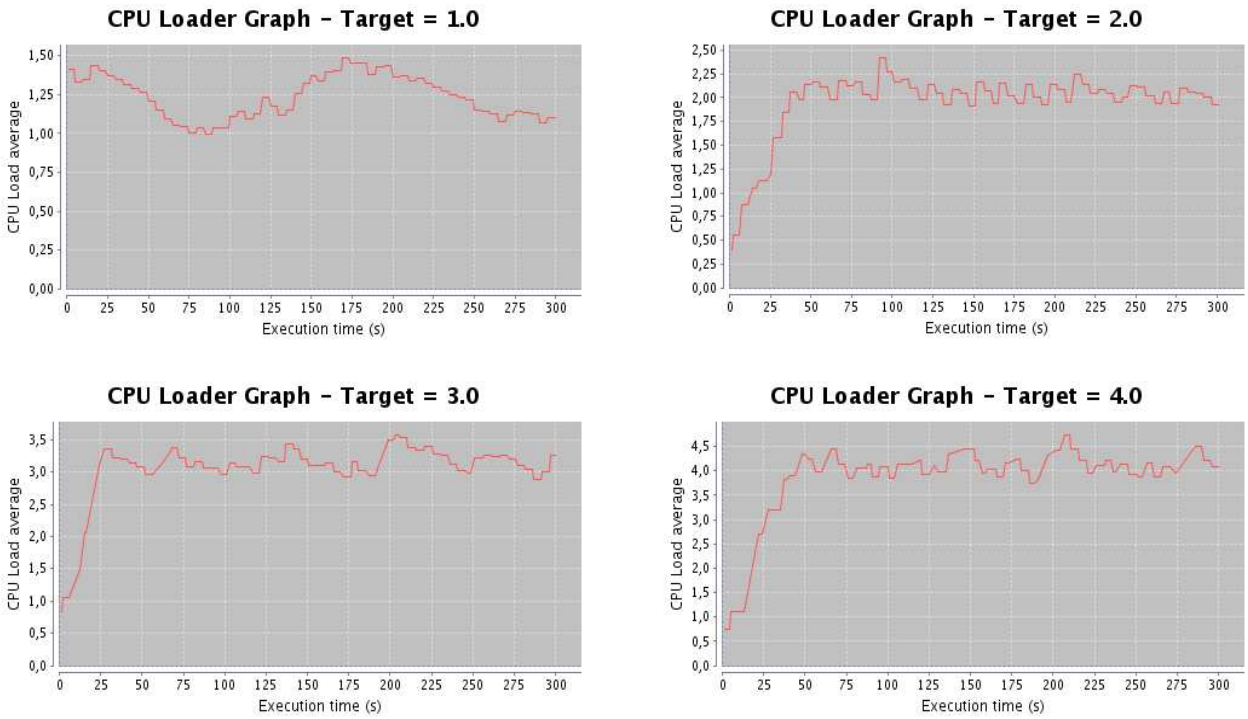


Figure 2.4.9. Performances du programme de montée de charge. *Afin de procéder au test 4, il est nécessaire de développer un programme de montée en charge en Java. Le fichier /proc/loadavg du système Linux Ubuntu contient des informations moyennées utiles au calcul de la charge actuelle.*
Ainsi il permet d'apprécier la charge de la machine en examinant le nombre de tâches exploitant et/ou attendant le CPU. Voici la structure de ce fichier : loadavg1 loadavg5 loadavg15 nr_running nr_threads last_pid. Les trois premières colonnes correspondent au nombre moyen de threads en état R (TASK_RUNNING) ou D (TASK_UNINTERRUPTIBLE) durant la dernière minute (loadavg1), les 5 dernières minutes (loadavg5) et les 15 dernières minutes (loadavg15).
nr_running donne le nombre de tâches en état TASK_RUNNING au moment de la lecture de /proc/loadavg.
nr_threads est le nombre de tâches présentes dans le système au moment de la lecture de /proc/loadavg.
last_pid est le dernier identificateur de tâche utilisé.
Le comportement du programme de montée en charge est très simple. Il prend en argument la charge visée ainsi que la durée de maintien de cette charge. Durant cette durée, il consulte le fichier /proc/loadavg afin d'obtenir la charge moyennée sur une minute (colonne 1). Dans le cas où cette dernière est inférieure à l'objectif, il crée des Threads Java se contentant d'effectuer des opérations mathématiques à base de nombres aléatoires. Puis, si la charge actuelle s'avère dépasser la charge visée, il supprime autant de Threads que nécessaire afin de rétablir l'équilibre. Les courbes ci-dessus montrent l'évolution de la charge au cours de quelques exécutions du programme et illustrent le contexte dans lequel la migration du domU est lancée.

2.2 Tests de performances du processus de migration

R-T5	*Énoncé du test*	Mesure comparative de la durée de la migration d'un *domU*.

	Programmes utilisés	`watch`, `xm create`, `xm list`, `xm migrate`.
		Le programme *RMI* est divisé en deux parties : le client et le serveur. Dans un premier temps, le serveur RMI est démarré par la commande
		`# . ./server.sh`
		et fournit une méthode
		`public SizeableObject pingPong(SizeableObject)`
		qui renvoie son argument sans autre traitement, elle sera appelée à distance par le client. Dans un deuxième temps, un appel RMI à la méthode `pingPong` est effectué par le client grâce à la commande
		`# . ./client.sh size server`
		`size` représente la taille de l'argument de type `SizeableObject` envoyé au `server`.

	Noyaux utilisés	*dapper-1-1*.
	Déroulement du test	*Préliminaire à l'appel RMI :* lancer le client RMI sur *dom0_a-204-02* et le serveur correspondant sur *dom0_a-204-03*. *Expérience RMI :* chronométrer l'appel RMI de dom0_a-204-02 vers *dom0_a-204-03* avec un argument de de taille 64 Mo. *Préliminaire à la migration :* le *domU dapper-1-1,* configuré avec 64 Mo de mémoire RAM, s'exécute sur *dom0_a-204-02*. *Expérience migration :* chronométrer la succession de la migration du *domU dapper-1-1* de *dom0_a-204-02* vers *dom0_a-204-03* puis de *dom0_a-204-03* vers *dom0_a-204-02*. *Conclusion :* l'étude comparative des deux expériences montre que la migration d'une machine virtuelle dure 50 % plus longtemps qu'un appel équivalent en RMI. La migration se fait à chaud *i. e.* le programme s'exécutant sur le *domU* n'est que brièvement interrompu (*cf.* R-T7).
	Succès	La durée de deux migrations successive exprimée en fonction de la durée d'un appel RMI.

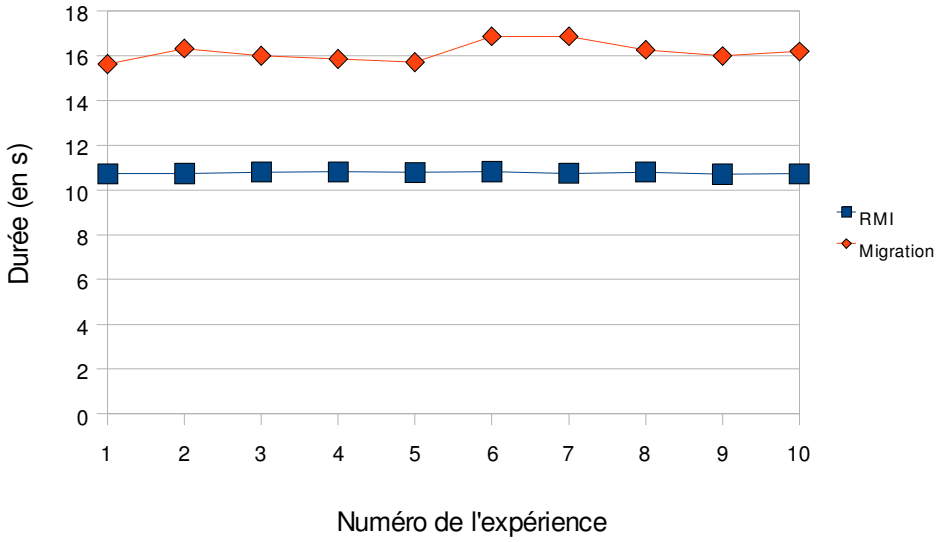


Figure 2.4.10. Mesures de la durée de deux migrations successives et de la durée d'un appel RMI. *Le graphique résume les résultats d'expériences successives de migrations et d'appels RMI équivalents. La durée moyenne de deux migrations successives vaut 16,2 secondes alors que la durée d'un appel RMI équivalent vaut 10,8 secondes. Cette différence s'explique par le fait que la migration n'interrompt pas les programmes qu'elle héberge.*

R-T6	*Énoncé du test*	Calcul du délai introduit par la migration dans l'exécution d'un programme s'exécutant sur un *domU*.
	Programmes utilisés	`watch`, `xm create`, `xm list`, `xm migrate`.
	Noyaux utilisés	*etch-1-1*.
	Déroulement du test	
	Succès	Le délai introduit par la migration exprimé en fonction de la durée d'une migration.

R-T7	*Énoncé du test*	Mesure de la durée de l'interruption de service produite par une migration de *domU* et doit s'exécuter sur une durée supérieure à la durée d'une migration.
	Programmes utilisés	`watch`, `xm create`, `xm list`, `xm migrate`. *Ping* utilise la commande `ping` pour solliciter un troisième *dom0* noté *dom0_référence.*
	Noyaux utilisés	*dapper-1-1*.
	Déroulement du test	*Ping* s'exécute sur un *domU* hébergé sur le *dom0_source*. Le *domU* en question subit une migration de *dom0_source* vers *dom0_destination*. Pendant toute la durée de son exécution *Ping* sollicite dom0_référence de manière synchrone. La durée de l'interruption de service introduit par une migration de *domU* est la différence entre le nombre de requêtes *Ping* envoyées par et le nombre de requêtes *Ping* reçues par le *dom0_référence*.
	Succès	Succès le temps moyen d'interruption, sur 20 mesures, est de 1 seconde.

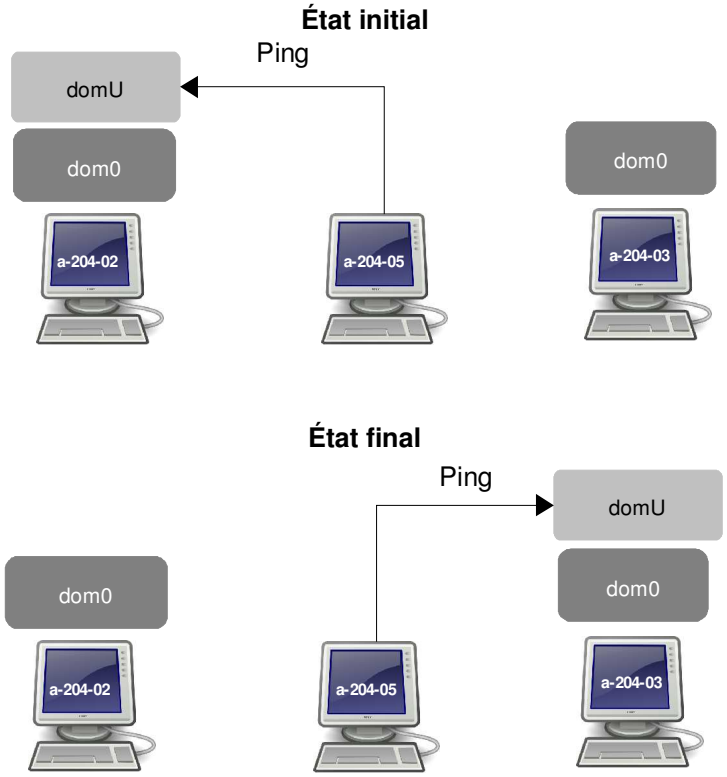


Figure 2.4.11. Ping de dom0_référence vers le domU. *Lors de l'expérience le domU est migré de dom0_sourec vers dom0_destination.*

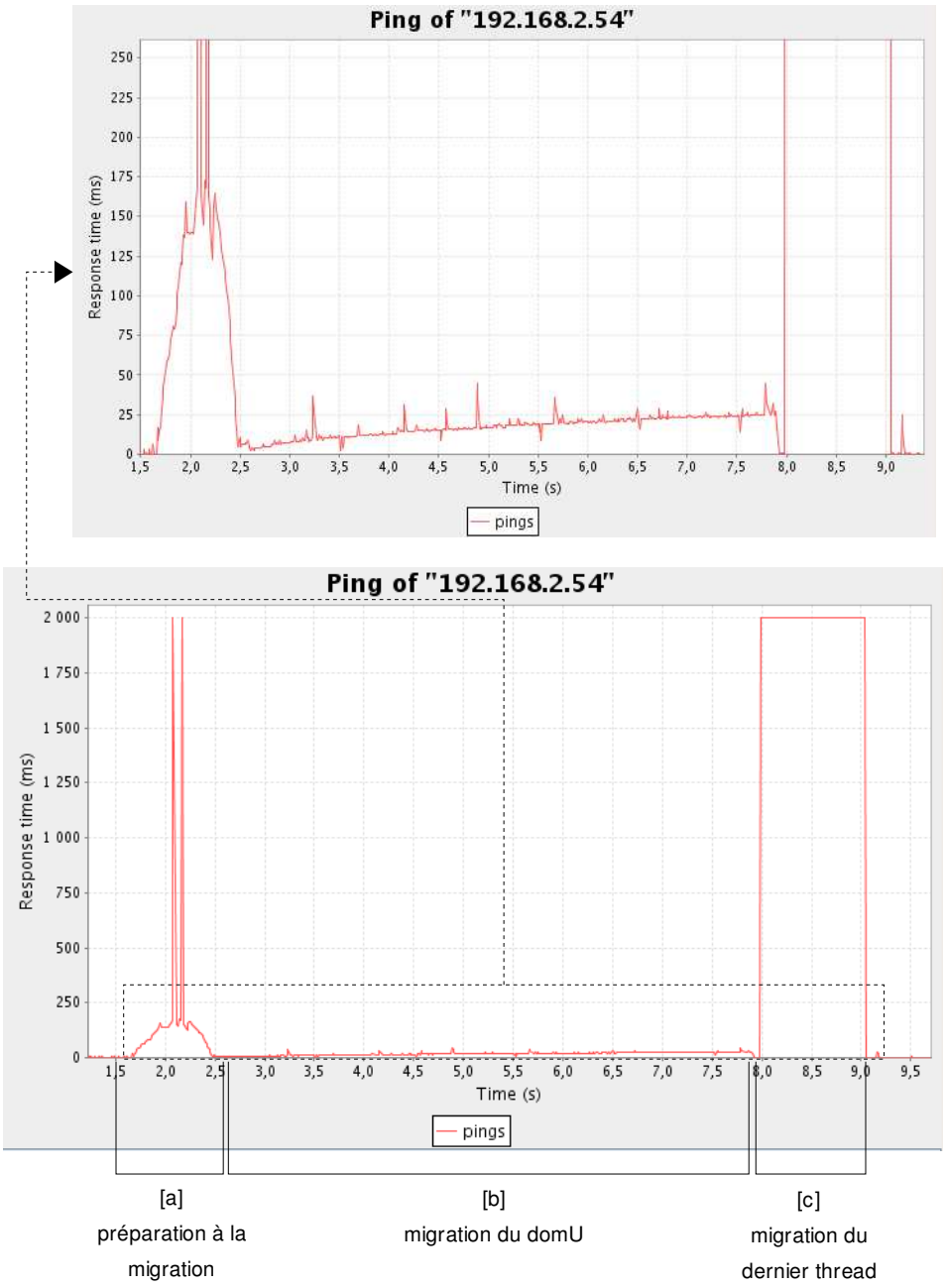


Figure 2.4.12. Mesure de l'interruption introduite par la migration. Le graphique présente le temps de réponse à un ping, dont la fréquence est de 1 ping toutes les 10 ms le timeout vaut 2 s, d'un domU en cours de migration à partir d'une machine physique distante (cf Figure 2.4.11). En dehors de la migration, le temps de réponse est en général inférieur à 10 ms. [a] La première phase de la migration consiste à vérifier la connection avec le dom0 distant vers lequel ont veut migrer. La présence de deux timeout lors de cette phase peut s'expliquer par l'exécution de deux opérations atomiques de longue durée. [b] Lors de la migration du domU, le temps de réponse au ping augmente sensiblement. Ceci s'explique par l'augmentation du nombre de tâches intrinsèques à la migration a effectuer au niveau du domU pour assurer le transfert complet du domaine. [c] La troisième phase correspond à la migration des dernier thread qui s'exécute et qui s'arrêtera pour 1 s durée de l'interruption du domU figurée ici par une colonne de pings atteignant chacun leurs timeout.

2.3 Test de l'utilisation du mécanisme de migration dans Tune

TUNe n'ayant pas d'influences sur les mécanismes mis en œuvre lors de la migration, les résultats de ces tests sont similaires à ceux décrits dans les sections précédentes. Les tests de validation concernant TUNe ont été effectués mais ne sont pas mentionnés par souci de concision. Toutefois, les sources présentes sur le serveur SVN permettent de les rejouer.

D'autre part, le test T2 est rejoué en enveloppant les machines virtuelle dans des wrapper pour les intégrer à une architecture TUNe. La migration est effectivement déclenchée lorsque la charge processeur dépasse un certain seuil. Une sonde spécifique (cf. fichier `Probe.java`) a été développée afin de monitorer la charge processeur et déclencher un événement en cas de dépassement du seuil fixé. La migration n'a pu être effectuée que dans un sens.

Problème	Le wrapper de la machine virtuelle *domU* en est désolidarisé lors de la migration.
Cause	La version de TUNe ne permet pas de reconstruire le wrapper sur la machine destination une fois le domU migré.
Solution	La nouvelle version permettra la construction du côté destination de la migration.

3 Conclusion

La matrice de couverture suivante permet de visualiser la validation de l'architecture Xen et TUNe mise en place.

		Exigences											
		EF1	EF2	EF3	EF4	EF5	EF6	EF7	EF8	EF9	EF10	EF11	EF12
Tests	T1	•											
	T2	•	•										
	T3	•		•									
	T4	•	•	•	•								
	T5	•				•							
	T6	•					•						
	T7	•						•					
	T8	•	•						•				
	T9	•		•						•			
	T10	•			•						•		
	T11	•										•	
	T12	×											×

Figure 3.1: Matrice de couverture.

Le symbole • indique que l'exigence est vérifiée par le test considéré.
Le symbole × indique que le test correspondant n'a pas été mis en œuvre.