Une approche de test de robustesse basée sur l'injection de fautes et le monitorage

Ana Cavalli, Eliane Martins, Anderson Morais, Bruno Moreira

▶ **To cite this version:**

Ana Cavalli, Eliane Martins, Anderson Morais, Bruno Moreira. Une approche de test de robustesse basée sur l'injection de fautes et le monitorage. Colloque Francophone sur l'Ingénierie des Protocoles (CFIP), Mar 2008, Les Arcs, France. hal-00250162

HAL Id: hal-00250162
https://hal.science/hal-00250162

Submitted on 11 Feb 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Une approche de test de robustesse basée sur l'injection de fautes et le monitorage

Ana Cavalli* — Eliane Martins*, — Anderson Morais **— Bruno C.Moreira***

* *Groupe des Ecoles des Télécommunications/Institut National des Télécommunications – UMR CNRS SAMOVAR*
9, rue Charles Fourier, 91011 Évry cedex

Ana.Cavalli@int-edu.eu, bruno.chasco_moreira@hotmail.fr

** *Instituto de Computação- Universidade de Campinas*
Av. Albert Einstein 1251, 13083-970 Campinas SP Brazil
{eliane,anderson.morais}@ic.unicamp.br

RÉSUMÉ. Le test de robustesse d'un système a pour objectif de déterminer si ce dernier se comporte comme attendu en présence de conditions d'exécution erronées ou non prévues. Cet article présente une méthode pour le test de robustesse des protocoles, qui combine l'utilisation de techniques de monitorage et l'injection de fautes. Le monitorage est basé sur le test passif : les entrées et les sorties sont collectées pendant l'exécution du système étudié, puis analysées vis-à-vis de la spécification représentée par un modèle formel ou par un ensemble de propriétés. Dans cet article, l'injection de fautes, qui permet d'émuler les défaillances du réseau, est utilisée conjointement avec le monitorage pour tester la robustesse d'un protocole donné. Nous illustrons l'application de cette méthode sur un cas d'étude réel, le protocole WAP (Wireless Application Protocol) et, en particulier, sur le protocole WTP (Wireless Transaction Protocol). Afin de montrer la pertinence de la méthode proposée, les résultats obtenus de nos expérimentations sur l'injection de différents types de fautes sont présentés.

MOTS-CLÉS : Test de protocoles, test de robustesse, test passif, injection des fautes.

ABSTRACT. Robustness testing has as main objective to determine if a system has the expected behaviour in the presence of faults. This paper describes a method for protocol robustness testing that combine monitoring and fault injection. Monitoring is based on passive testing: the inputs and outputs are collected during execution of the target system and further analyzed with respect to the specification represented either by a formal model or by a set of properties. In this paper, fault injection that allows to emulate communication failures, is used with monitoring to test the robustness of a given protocol. We illustrate the method application with a real world case study, the WAP (Wireless Application Protocol). Some experiments were performed, corresponding to the injection of a certain kinds of fault. The experimental results are presented in the paper.

KEYWORDS: protocol testing, robustness testing, passive testing, fault injection.

1. Introduction

La robustesse d'un système peut être définie comme sa capacité de présenter un comportement *acceptable* en présence de fautes ou de conditions environnementales stressantes (Castanet et al., 2003).

Plusieurs approches pour le test de robustesse ont été proposées dans la littérature. Certaines s'inspirent du test de conformité incluant une phase de génération de tests basée sur le modèle formel représentant le comportement du système considéré. Dans ces approches, plusieurs types de modèles ont été utilisés (voir (Bochmann *et.al*, 1994) pour une présentation générale sur le sujet). La principale difficulté de ce type d'approches concerne la représentation formelle des fautes et leur contrôle (Khorchef *et al*., 2006). En effet, la définition d'un modèle formel pour la représentation des fautes est loin d'être triviale voire impossible (e.g, le temps entre deux entrées est plus court que le temps de réaction du système). De plus, il faut que ces fautes puissent être contrôlées pendant l'exécution par le *test harness* usuellement utilisé dans les tests de conformité.

Ces limitations nous ont amenés à choisir l'injection des fautes combinée avec le monitorage comme une technique mieux adaptée pour tester la robustesse d'un système. L'approche que nous avons définie procède en deux étapes. Dans la première étape, des fautes sont injectées dans le système, des traces d'exécution du système sont ensuite collectées, dans la seconde étape, pour être analysées en utilisant des techniques de monitorage afin de vérifier si l'implantation du système inclut bien des actions de recouvrement qui lui permettent d'être résilient aux fautes injectées.

L'injection de fautes consiste en l'introduction délibérée de fautes dans un système informatique afin d'observer son comportement. L'injection de fautes est cruciale pour les tests de robustesse, car elle accélère l'occurrence de fautes et d'erreurs qu'il serait difficile de recréer avec les techniques usuelles de test, ou qui nécessiterait trop de temps pour pouvoir les observer pendant l'exécution normale du système. Ainsi, à la place d'un *test harness* conventionnel, on utilise un injecteur des fautes qui peut s'insérer soit dans l'implantation sous test (IUT) elle-même, soit dans son contexte (bibliothèques de communication, couches inférieures ou même le matériel). Dans notre approche, l'injection des fautes est utilisée pour émuler les défaillances du réseau. Les fautes injectées ne sont pas choisies de manière aléatoire. Leur choix est basé sur l'observation des défaillances réelles ou sur des erreurs spécifiques que l'implantation est supposée tolérer. C'est cette dernière option que nous considérons dans ce travail. La plupart des méthodes de test de robustesse basées sur l'injection de fautes repose sur la définition suivante : « un système (une application) est considéré robuste si ce dernier ne s'arrête pas (crash ou hang) ». Une telle définition peut s'avérer assez restrictive. En effet, une défaillance n'arrête pas nécessairement le système, mais peut conduire l'implantation à des états erronés provocant une exécution incorrecte de ses fonctionnalités. Pour palier ce problème, notre approche utilise des techniques de monitorage basées sur des invariants pour

représenter des propriétés de robustesse. Ces propriétés sont soit fournies directement par les experts du réseau, soit déduites à partir de l'expression de besoins, soit encore à partir des modèles représentant le comportement du système étudié.

Les techniques de monitorage utilisées dans ce travail sont basées sur le test passif. Le test passif suppose l'observation du comportement de l'implantation sans interférer avec son exécution. La première étape consiste à collecter un corpus de traces. Une fois ce corpus obtenu, différents algorithmes sont appliqués pour vérifier que les traces collectées sont conformes à la spécification ou qu'elles vérifient certaines propriétés. Ces algorithmes sont des adaptations des algorithmes classiques de « pattern matching ».

Pour la mise en œuvre de notre approche nous utilisons deux outils : TestInv et Firmament. Le premier outil, développé à l'INT, permet d'automatiser toutes les étapes du test passif. Le second outil, Firmament, implémente un injecteur de fautes transparent pour l'utilisateur de l'application.

Pour illustrer la faisabilité et la pertinence de notre approche, le protocole WTP (Wireless Transaction Protocol) a été choisi comme cas d'étude. Ce protocole constitue l'un des composants imbriqués du protocole WAP (Wireless Application Protocol). WAP désigne une spécification globale ouverte, il permet à des utilisateurs mobiles d'accéder facilement, et de manière très rapide à l'information et aux services de l'Internet via des dispositifs mobiles. Au total, cinq expériences ont été réalisées. Chaque expérience est composée de plusieurs exécutions. Une exécution correspond à l'injection d'une faute. Dans cet article, nous détaillons les résultats de nos expériences sur le protocole WTP qui nous ont permis de détecter une défaillance dans ce dernier. Les expériences constituent à elles-seules une contribution originale de cet article, car ce type d'étude n'a pas été mené auparavant.

L'approche que nous proposons est originale car jusqu'à présent aucun travail de recherche dans le domaine n'a combiné ces deux techniques. De plus, elle permet de vérifier qu'un système exécute des actions de recouvrement en présence de fautes sans même avoir accès direct à son implantation. Ainsi, des couches protocolaires par exemple peuvent être testées bien que le testeur ne puisse pas envoyer directement les messages à la couche sous test.

L'article est organisé comme suit. La section suivante présente l'approche que nous proposons basée sur le monitorage et les techniques d'injection de fautes. La section 3 décrit les caractéristiques des deux outils, TestInv et Firmament, ainsi que les détails de leurs implantations. La section 4 décrit brièvement le cas d'étude: le protocole WAP. La mise en place de l'expérimentation de notre approche ainsi que les résultats obtenus sur le cas d'étude sont présentés dans la section 5. Nous terminons par une conclusion et des perspectives potentielles à notre travail.

2. L'approche proposée pour le test de robustesse

Cette section présente l'approche que nous proposons pour le test de robustesse. Nous commençons par décrire les types de fautes que nous considérons ainsi que les invariants utilisés pour le monitorage. Enfin, une vue générale des principales étapes de l'approche sont présentées.

2.1 Injection des fautes : types de fautes

Les types de fautes que nous considérons dans cet article sont basés sur les modes de défaillance suivants (Hadzilacos *et al*., 1994):

- défaillances d'omission : ce type de défaillances peut être émulé par l'interception soit de tous les messages provenant d'un hôte spécifique soit uniquement des messages envoyés ou reçus par un hôte.
- défaillances arbitraires : les défaillances de ce type, appelées aussi défaillances Byzantines, sont émulées par la corruption des messages reçus par l'IUT. Pour injecter les fautes Byzantines, il faut déterminer quels champs du message seront corrompus et avec quelles valeurs. Pour cela, on a utilisé l'approche proposée par Ballista (Kropp *et al*., 1998), où des valeurs spécifiques aux types des données sont choisies. Ainsi par exemple, pour des champs de type entier, des valeurs tels que -1, 0 et 1 peuvent être utilisées. Ceci permet la prise en compte des fautes de communication, mais également celles d'interface qui sont très utilisées pour tester la robustesse de fonctions dont les interfaces sont bien définies.
- défaillances temporelles : les défaillances de ce type peuvent être émulées par des délais de livraison de messages plus longs (ou plus courts) que prévus.

2.2 Le test passif

Les méthodes formelles pour le test peuvent être classées en deux grandes approches : actives et passives. Dans les approches actives, le testeur possède la capacité de stimuler l'IUT et de vérifier si la réponse obtenue pour chaque stimulus est en accord avec la spécification du système. Dans les approches passives, le testeur n'a pas besoin d'interagir avec l'IUT ; son rôle se limite à l'observation (ou monitorage) de l'exécution de l'IUT sans aucune interférence. Par conséquent, le testeur a besoin de points d'observation (PO) à partir desquels il peut collecter les messages échangés entre deux entités protocolaires. Dans ce cas, le testeur a deux rôles principaux : (i) observer, collecter et filtrer l'information échangée entre deux entités protocolaires. Le testeur est donc similaire à un analyseur (ou renifleur) de paquets, il intercepte et enregistre les messages passant sur la totalité du réseau ou

une partie de ce dernier; (ii) analyser les informations collectées (traces d'exécution) pour déterminer si elles sont conformes vis-à-vis de la spécification.

L'approche du test passif proposée dans cet article est basée sur l'analyse d'invariants. Les invariants sont des propriétés que la spécification du système doit satisfaire. Par conséquent, l'IUT devrait également les remplir pour être conforme à la spécification. Les invariants peuvent être extraits de la description informelle (ou textuelle) du protocole, ou de la spécification formelle de ce dernier. Dans le cadre de nos travaux de recherche, nous utilisons les machines d'états finis (FSM) pour la représentation formelle d'un protocole.

Définition d'invariant

Invariant simple

Soit *I* (resp. *O*) un ensemble de symboles d'entrée (resp. de sortie). Les invariants d'un système sont exprimés comme une succession de symboles d'entrée et de sortie. L'invariant (i_1/o_1, ..., i_{n-1}/o_{n-1}, i_n/O) est interprété comme suit: «chaque fois qu'on observe sur l'IUT la trace (i_1/o_1, ..., i_{n-1}/o_{n-1}) et qu'on observe l'entrée i_n alors on obtient une sortie qui appartienne à *O* ». Les invariants de ce type sont appelés des invariants *simples*. Les symboles "?" et "*" peuvent également être utilisés pour la définition d'invariants plus élaborés. Le symbole "?" est utilisé de la même manière que pour le « pattern matching », c'est-à-dire, il peut remplacer n'importe quel symbole. Cependant, la signification du symbole "*" a été légèrement modifiée. Par exemple, la signification de l'invariant (*i/o,*, i'/O)* est la suivante : si on détecte la transition *i/o* dans la trace, alors la première occurrence de *i'* est suivie par une sortie appartenant à *O*. Le symbole "*" remplace donc n'importe quelle séquence de symboles ne contenant pas l'entrée *i'*.

Invariant d'obligation

Bien que les invariants simples permettent la spécification de nombreuses propriétés des systèmes informatiques représentés par des FSM, ils ne sont pas toujours suffisants pour exprimer toutes les propriétés souhaitées. Dans le domaine des protocoles par exemple, on aimerait spécifier que l'occurrence d'un événement *e* est toujours précédée d'une séquence particulière d'événements. C'est pour cette raison que nous définissons un type supplémentaire d'invariant dit invariant d'*obligation*. Ainsi par exemple, l'invariant (1) signifie : si une sortie appartenant à *O* est observée, alors l'entrée *i* suivie de la sortie *o* a été observée auparavant.

$$i / o,*,i' / \overline{O} \qquad (1)$$

Dans le cadre de ce travail, les invariants sont représentés en utilisant l'EBNF (Extended Backus-Naur Form). Des exemples d'une telle représentation sont donnés à la Section 5.

Les invariants doivent être validés vis-à-vis de la spécification. La validation des invariants est nécessaire dans le contexte de notre approche car ces derniers sont fournis par l'équipe de test et peuvent donc être faux. En effet, ils ne sont pas

dérivés à partir d'un modèle formel. L'avantage de l'obtention des invariants par l'équipe de test est de se restreindre aux propriétés les plus pertinentes ou critiques.

2.3 Vue générale de l'approche proposée

Les différentes étapes de l'approche proposée sont décrites ci-dessous :

1ère étape : injection de fautes. Nous utilisons une approche basée sur l'insertion de code au niveau réseau qui peut affecter les messages des couches supérieures à travers l'utilisation de la capture et le filtrage de paquets. Nous posons comme hypothèse que ces fautes peuvent être provoquées par des défaillances dans le réseau ou dans l'entité paire.

2ème étape : *définition d'une architecture de test*. Mise en place de points d'observation (PO) qui seront utilisés pour la collecte de traces.

3ème étape : *application des techniques de monitorage*. Ceci implique :

- la collecte de traces d'une part, et d'autre part leur formatage pour les rendre directement utilisables par des techniques de monitorage;

- l'application des algorithmes d'analyse des invariants afin de déterminer si l'IUT a mis en place les actions de recouvrement des fautes injectées. Dans cette étape un verdict est donné suivant le succès ou l'échec de la validité des invariants.

3. Mise en œuvre de l'approche proposée

Dans cette section, nous présentons la mise en œuvre de notre approche par l'utilisation conjointe des deux outils TestInv et Firmament (voir Figure 1).

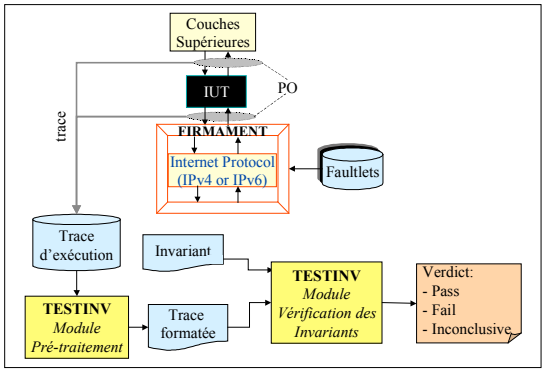

Figure 1. *Utilisation conjointe de TestInv et Firmament pour le test de robustesse.*

3.1. *TestInv*

L'outil TestInv (Bayse et al., 2005) est l'outil qui implante l'approche de test passif présentée dans la section 2.2. La figure montre deux des modules de TestInv. Le module *Pré-traitement* se charge du traitement des fichiers log contenant les corpus des traces observées. Le fichier est filtré et analysé grammaticalement afin d'obtenir une trace convenable, c'est-à-dire, une trace incluant l'information concernant les noms des primitives des entrées et sorties, ainsi que les données pertinentes aux messages échangés (l'adresse source, l'adresse de destination, l'url accédé, etc.). Ce module inclut une interface qui permet à l'utilisateur de paramétrer le format des traces à collecter. Ainsi, il n'est pas nécessaire de modifier le module *code* pour traiter les fichiers log qui proviendraient de différentes implantations, et par conséquent de différents formats. Enfin, le module *Vérification des Invaraints* détermine si les traces collectées satisfont bien la liste d'invariants donnée en entrée. L'outil a été complètement implémenté en JAVA (spécification J2SE 1.4.0).

3.2. *Firmament*

Firmament est une évolution de l'outil ComFIRM (COMmunication Fault Injector through operating system Resources Modification) (Drebes et al., 2005) lequel explorait l'extension du kernel Linux afin de développer un injecteur de fautes de communication transparent à l'utilisateur de l'application. Firmament permet la spécification de scénarios de fautes (*faultlet*). Un *faultlet* est exécuté sur chaque paquet qui traverse le flux de communication ; il a la capacité d'inspecter et de modifier les contenus d'un paquet, l'écarter, le dupliquer ou le retarder. Un *faultlet* est exécuté sur la machine virtuelle FIRMVM, qui est un module connecté au sous-système de communication à travers l'interface Netfilter. FIRMVM est responsable d'interpréter et de traiter les commandes qui spécifient les scénarios incluant des fautes.

4. L'implantation cible

Nous avons appliqué notre approche de test de robustesse au WAP (Wireless Application Protocol) (OMA 2001). Le WAP a pour but de permettre à des terminaux mobiles, tels que téléphones mobiles, PDAs (Personal Digital Assistant), ou autres, d'accéder à l'Internet.

Le WAP est structuré en sept couches successives. Nous présentons ici uniquement la couche WTP qui est la cible des tests réalisés. La couche WTP offre des services de transactions à sa couche supérieure (WSP) lorsque le mode connecté est sollicité. Cette couche utilise un service de datagramme fourni par la couche inférieure (WDP ou UDP). WTP offre des services du type requête-réponse. Le processus qu'initie une transaction est l'entité désignée comme *Initiator*, et la

réception de celle-ci est faite par l'entité *Responder*. La communication entre ces deux entités peut s'effectuer suivant trois niveaux différents de classe (0, 1 et 2). Ces niveaux de classes conditionnent le nombre de messages (en termes d'acquittement et de résultat) échangés entre les entités. Plus le niveau de la classe est élevé, plus le service est fiable.

Les interactions observables en entrée ou en sortie de la couche WTP pour un service de classe 2 sont : Invoke (pour initier une transaction), Result (pour retourner un résultat) et Abort (pour avorter une transaction). Ces interactions peuvent être des primitives de services échangées entre le WTP et le WSP, ou alors des PDUs (Protocol Data Units) transmises entre le *Initiator* et le *Responder*. Les PDUs correspondent aux interactions présentées plus haut, auxquelles on ajoute le PDU Ack, utilisée par les deux entités (émise par l'une ou par l'autre selon le niveau de classe adopté) pour acquitter un message reçu.

Une primitive peut s'écrire sous la forme suivante : TR-<nom>.<type>(<paramètres>), où : TR désigne la couche supérieure (WSP) ; <nom> correspond aux interactions présentées précédemment ; <type> peut prendre les valeurs suivantes : req (indique une demande de service faite par WSP), ind (informe la couche supérieure qu'une demande de service a été faite par l'entité distante), res (accuse la réception d'une primitive de type indication à sa couche inférieure), cnf (signale que la demande a été effectuée avec succès). Ces paramètres varient selon les types de primitives.

5. Les expériences réalisées

Cette section présente les expériences et résultats obtenus de l'application de notre approche sur le protocole WAP. À cet effet, la passerelle (ou *gateway*) WAP a été choisie comme cas d'étude pour réaliser l'interface entre le terminal mobile, fonctionnant sur un réseau sans fil, et le serveur. Dans ces conditions, la passerelle est exposée au modèle de fautes de communication mentionné en section 2.1. Par conséquent, sa robustesse par rapport à ce type de fautes doit être testée.

5.1. L'architecture des tests

La Figure 2 présente l'architecture mise en place pour la réalisation des tests de robustesse. Elle est composée de la passerelle WAP, mise en œuvre par un logiciel *open source*, Kannel[1], version 1.4.1. Cette passerelle a été utilisée dans le cadre de travaux précédents (Bayse et al 2005) , elle contient donc les points d'observation (PO) à travers lesquelles les informations, nécessaires pour la validation des invariants vis-à-vis de l'IUT, sont capturées dans des fichiers *log*. Chaque fichier contient les traces locales d'exécution d'une couche particulière du WAP.

[1] http://www.kannel.org/

Dans notre cas, le rôle du client (terminal mobile) est réalisé à travers un simulateur de téléphone WAP. Il s'agit du *Nokia Mobile Browser Simulator*[2] *4.0* (NMB4.0). C'est un logiciel qui est capable de simuler un téléphone WAP et qui dispose d'une interface graphique représentant un terminal mobile. L'interface fournie au client est un *browser* qui lui permet de demander à un serveur des pages en format WML. L'injecteur de fautes est situé dans la même machine que Kannel. Il capte tous les messages en entrée et en sortie comme l'indiquent les flèches différenciées (↑↓).

5.2. Description des tests

L'objectif de l'approche est de révéler la présence de fautes de robustesse dans l'implantation d'un protocole. Comme ces fautes internes peuvent être révélées en présence des fautes externes injectées, on a réalisé cinq expériences, composées de plusieurs *exécutions* (*run*), où chacune correspond à l'injection d'un type de fautes. Il est important de signaler qu'à chaque nouvelle expérience, on redémarre l'environnement de tests et l'IUT pour s'assurer que les éventuelles erreurs activées précédemment n'affectent plus son exécution.

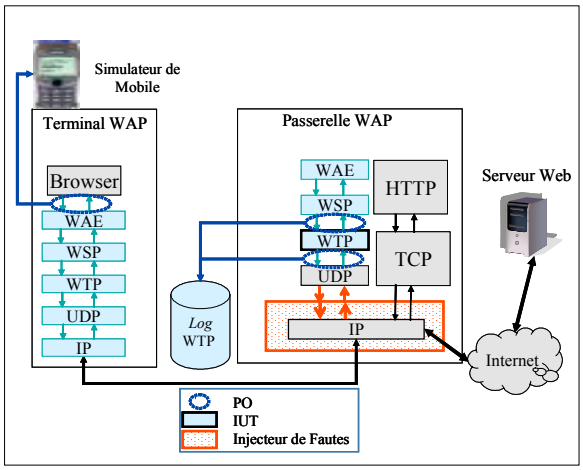

Figure 2. *L'architecture de test***.**

L'analyse des résultats est basée sur les éléments suivants: la sortie du browser vers l'utilisateur et les logs WTP collectés par les POs. Le NMB émet aussi un verdict PASS or FAIL selon que la page requise est fournie ou non en réponse. Les logs WTP sont utilisés par l'outil TestInv pour effectuer l'analyse des invariants présentés en Section 2.2.

[2] http://www.forum.nokia.com/info/sw.nokia.com

Pour les tests, étant donné notre intérêt pour les mécanismes de détection et de traitement d'erreurs spécifiés, nous nous sommes focalisé sur les services de transaction de classe 2 du WTP qui offrent des services fiables. Le scénario normal typique d'échange des messages entre le client et le serveur est montré dans la Figure 3(a), tandis qu'en 3(b) on présente le scénario en présence des fautes. Nous pouvons d'ores déjà établir un invariant simple pour vérifier ce scénario :

$$S_1 ::= RcvInvoke / \mathrm{TR}-Invoke.ind, *, \mathrm{TR}-Result.req / \{Result\}$$

Cet invariant exprime la propriété suivante : à chaque fois qu'une séquence d'entrées/sorties initiée par la paire RcvInvoke/TR-Invoke.ind est dans la trace (log WTP), alors la première occurrence de l'entrée TR-Result.req doit être suivie de la sortie Result. Le symbole "*" indique que plusieurs interactions peuvent avoir lieu entre la première et la dernière paire.

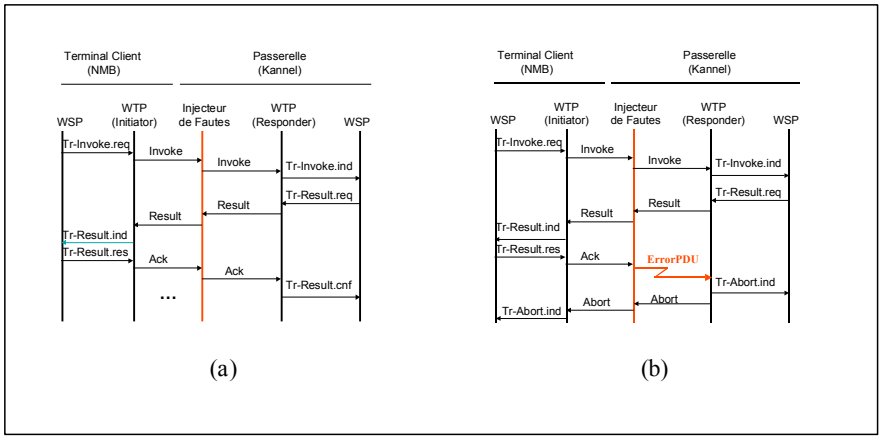

Figure 3. *Scénarios requête-réponse pour les services WTP classe 2 : (a) scénario normal ; (b) scénario avec corruption de la PDU Ack.*

Une fois que la PDU Result est envoyée, le *Responder* fait une attente temporisé pour la confirmation de la part de l'*Initiator*, ce qu'il fait en envoyant la PDU Ack. À la réception de l'événement correspondant (RcvAck), le *Responder* envoie la confirmation à la couche WSP. La propriété suivante, exprimée par un invariant d'obligation, établit que la confirmation à l'utilisateur local (TR-Result.cnf) n'est envoyée qu'après l'envoi et la confirmation de la réponse :

$$O_1 ::= ?/ Result, *, RcvAck / \overline{\{\mathrm{TR}-Result.cnf\}}$$

À présent, considérons le cas où les messages peuvent être corrompus par le canal de communication. Lorsque la couche inférieure WDP détecte un PDU erroné (erreur de *cheksum*, taille trop courte ou trop longue, etc.), un événement d'erreur RcvErrorPDU est reçu par WTP. Dans ce cas, le *Responder* doit envoyer une PDU

Abort à l'*Initiator*. Ce comportement peut être exprimé par l'invariant simple suivant :

$$S_2 ::= RcvErrorPDU / \{Abort\}$$

Un autre mécanisme de traitement d'erreur utilise le *timer* de retransmission (R). Lorsque le *Responder* transmet une réponse (PDU Result), il déclenche le timer R pour attendre la confirmation pendant un certain temps. À l'échéance de R (indiqué par l'événement TimerTO_R), le *Responder* retransmet la réponse et reprend l'attente de confirmation. Au bout d'un nombre maximum de retransmissions, la transaction est avortée et une indication d'échec est envoyée au WSP. Cette propriété est exprimée par l'invariant simple suivant :

$$S_3 ::= TimerTO_R / \{Result, \mathrm{TR} - Abort.ind\}$$

Les propriétés présentées jusqu'ici représentent uniquement les aspects fonctionnels. Or en présence de fautes, le système peut se retrouver dans des situations de blocages (*deadlocks*) où l'IUT n'accepte aucune entrée et ne produit aucune sortie, ou encore des situations où l'IUT avorte son exécution suite à une erreur non traitée. Pour cela, on a défini deux entrées supplémentaires représentant ces situations de blocage et d'avortement de l'exécution: PANIC et HANG. La sortie est considérée comme NULL dans les deux cas. Deux invariants simples (S_4 et S_5) ont été créés pour vérifier la présence de ces événements sur la trace. Notons que contrairement aux invariants présentés préalablement, la validité de S_4 et S_5 ne garantit pas la robustesse de l'IUT.

$$S_4 ::= HANG / \{NULL\}$$

$$S_5 ::= PANIC / \{NULL\}$$

Dans cet article, nous nous limiterons à la présentation de ces propriétés (S_4 et S_5) car elles permettent déjà de montrer l'utilité de l'approche. Bien sur, la notation permet aussi de représenter d'autres aspects, comme par exemple, les paramètres de primitives et PDUs. Plus de détails sont disponibles dans (Bayse 2006).

Les résultats obtenus jusqu'à présent nous permettent déjà de faire plusieurs constatations. Tout d'abord, ces résultats ont permis de démontrer que considérer uniquement les observations au niveau de l'application n'est pas suffisant pour déterminer la robustesse d'un système. En effet lors d'une expérimentation, l'application s'est terminée avec succès, sans pour cela que les propriétés soient satisfaites. Plus précisément, S_3 a eu un verdict FAIL car l'IUT envoie un message de plus que ceux prévus dans la spécification. Une autre expérience nous a permis d'observer un comportement d'arrêt sous défaillance (*fail stop*) du protocole (S_5 est vrai), ce qui est parfaitement acceptable. Enfin, le seul cas de défaillance observé et considéré comme le plus critique concerne le blocage de l'IUT (S_4 est vrai) qui entraîne le blocage du client à son tour. Cependant, les propriétés S_1 à S_3 sont satisfaites dans ce cas.

6. Conclusions et Perspectives

Dans cet article nous avons présenté une approche pour le test de robustesse combinant l'injection des fautes par logiciel avec le monitorage. À notre connaissance, aucune des approches existantes de test de robustesse ne propose une telle combinaison. L'injection des fautes permet, entre autres, un meilleur contrôle des fautes comparée aux testeurs usuellement utilisés pour les tests actif. Par opposition aux techniques d'analyse de résultats utilisées traditionnellement dans les études basées sur l'injection des fautes, le monitorage permet une analyse plus complète et plus fine des résultats.

L'approche que nous proposons utilise des invariants pour représenter les propriétés attendues de l'implantation en présence de fautes. L'utilisation des invariants permet d'analyser le comportement d'un système modélisé par des machines à états finis. Comme le modèle des machines à états finis est largement utilisé pour spécifier des protocoles de communication, l'approche est donc très utile et accessible aux utilisateurs. À travers cette étude et moyennant quelques adaptations des outils existants, nous avons montré qu'il était possible d'appliquer l'approche proposée sans grande difficulté. Dans nos travaux futurs, nous envisageons d'étendre l'approche proposée pour la prise en compte des aspects temporels qui sont difficiles à modéliser par des expressions régulières.

Remerciements

Les auteurs tiennent à remercier la CAPES pour le soutient de l'un des auteurs. Nous remercions également Amel Mammar pour ses remarques et correction du français.

7. Bibliographie

Bayse E., Cavalli A., Nunez M., Zaidi F., « A Passive Testing Approach based on Invariants: Application to the WAP », *Computer Networks*, 48, pp247-266, 2005.

Bayse E., Méthodologie de test passif par invariants. Application au protocole WAP, Thèse de doctorat, Institut National des Télécommunications en co-accréditation avec l'Université d'Evry-Val d'Essone, 2006.

Bochmann G.v., Petrenko A., « Protocol Testing: Review of Methods and Relevance for Software Testing », *Actes des* International Symposium on Software Testing and Analysis (ISSTA), August 17-19, 1994, Seattle, WA, USA, pp 109-124.

Castanet R., Waeselynk H., Techniques avancées de test de systèmes complexes: test de robustesse, rapport de recherche, 2003, CNRS-AS23.

Drebes R.J., Jacques-Silva G., Fonseca da Trindade J., Weber T.S., « A Kernel-based Communication Fault Injector for Dependability Testing of Distributed Systems », *Actes*

des Parallel and Distributed Systems: Testing and Debugging (PADTAD-3), 2005, Haifa, Israel.

Hadzilacos V., Toueg S., A modular approach to fault-tolerant broadcasts and related problems. Technical Report TR 94-1425, Cornell University, Dept. of Computer Science, Cornell University, Ithaca, NY 14853, May 1994.

Khorchef F. S., Berrada I., Rollet A., Castanet R., « Cadre formel pour le test de robustesse. Application au protocole SSL », *Actes des CFIP'2006* , Tunisie, Octobre 2006, pp199-202.

Open Mobile Alliance (OMA, ex-WAP Forum). WAP Wireless Transaction Protocol Specification, WAP-224-WTP-20010710-a, July/2001.