Techniques de monitorage pour la vérification de propriétés

Ana Cavalli
Institut Telecom/Telecom & Management SudParis

Plan de la présentation

- Techniques de monitorage (test passif)
- Domaine d'application: le protocole WAP
- Détection d'intrusion dans les réseaux ad hoc basée sur le monitorage (test passif)

Techniques de Monitorage
(Test passif)

Motivation

- Méthodes de test classiques fortement liées à la contrôlabilité de l'implantation sous test
  - Elles sont basées sur l'habilité d'un testeur que stimule l'implantation sous test et évalue la correction des réponses produites par l'implantation
- Parfois, une activité difficile
  - le testeur n'a pas une interface directe avec l'implantation
  - l'implantation est construite à partir de composants qui doivent être exécutés dans son environnement et ne peuvent pas être arrêtés ou interrompus (pour long temps) afin de les tester

Motivation

- Pourquoi utiliser le test passif ?
  - Dans le test passif, il n'est pas nécessaire que l'implantation sous test interagisse avec le testeur
  - Les traces d'exécution sont observées sans interférences sur l'exécution du protocole
- Le test passif a des applications multiples
  - Il peut être utilisé comme une technique de monitorage pour la détection et le report des erreurs
  - Aussi, pour le management des réseaux afin de détecter des problèmes de configuration, identification des erreurs ou disponibilité des ressources

Monitorage basé sur des méthodes formelles (test passif)

- Les techniques de monitorage ne sont pas nouvelles
  - Activité de recherche très active ces dernières années
  - D'habitude, les traces d'exécution de l'implantation sont comparées avec la spécification pour détecter des erreurs dans l'implantation
  - La spécification a la forme d'une machine d'états finis (FSM) et le monitoring consiste à vérifier que la trace exécutée est acceptée par la spécification
  - Architecture basée sur les POs

Monitorage basé sur des méthodes formelles (test passif)

- Nous proposons une approche “plus active”:
  - Nous partons d’un ensemble d’invariants qui représentent les propriétés les plus relevantes de l’implantation sous test
  - De manière informelle, un invariant exprime le fait que chaque fois que l’implantation sous test réalise une séquence d’entrées/sorties elle doit exhiber un comportement exprimé par l’invariante

Monitorage basé sur des méthodes formelles (test passif)

- Les invariants doivent être fournies par l’expert connaissant le protocole à tester
- Première étape: vérifier que l’invariant est correct par rapport à la spécification
- Deuxième étape: vérifier que les traces produites par l’implantation sous test respectent les invariants

Exemples

- *« Chaque fois qu'un utilisateur demande une connexion et que la connexion est assurée, si après avoir réalisé quelques opérations, l'utilisateur demande à être déconnecté, alors il est déconnecté. »*
- *« Chaque fois qu'un utilisateur demande une ressource (par exemple une page web) alors la ressource est obtenue ou on a un message d'erreur. »*

Invariants

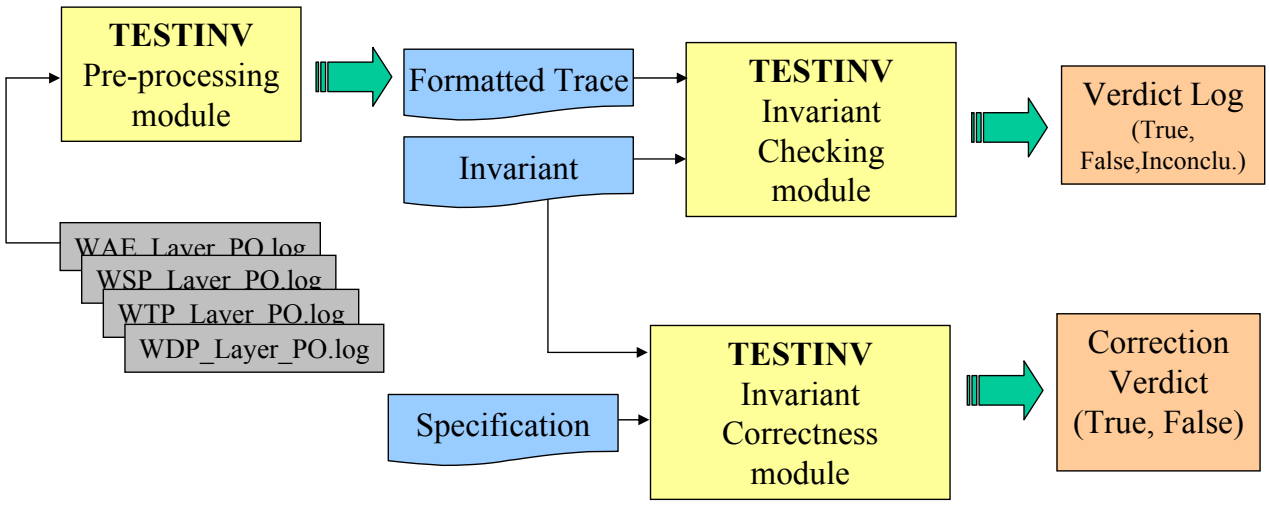
- Invariant simple
  - Une trace $i_1/o_1, \ldots, i_{n-1}/o_{n-1}, i_n/\boldsymbol{O}$ est un invariant simple si chaque fois que la trace $i_1/o_1, \ldots, i_{n-1}/o_{n-1}$ est observée, si nous obtenons l'entrée i_n alors nous obtenons une sortie appartenant à ***O***, où ***O*** est inclut dans l'ensemble des sorties attendues.
  - *i/o, *, i'/**O*** signifie que si nous détectons la transition *i/o* alors la première occurrence du symbole *i'* est suivie par une sortie appartenant à l'ensemble ***O.***
  - ***** remplace n'importe quel séquence des symboles ne contenant pas le symbole d'entrée *i'*.

Invariants

- Invariant d'obligation

  « si *y* se produit alors *x* doit se produire avant »

L'outil TESTINV



Spécification

Specification SDL, IF ⇒ Extended Finite State Machine (EFSM)

Simulation (Simulateur IF, ObjectGEODE)

Graphe d'accessibilité ⇒ Finite State Machine (FSM)

Traduction en un format approprié pour traiter les invariants (outil Sirius)

Automate au format Aldebaran ⇐ Réduction

(état de départ, entrée / sortie, état d'arrivé)


```
$1 3
trans msap_manager1(1) : start
$2 6
trans responder(1) : from_all_input_idisreq with idisreq from env_isapres
input idisreq from env_isapres to responder(1)
output dr from responder(1) via ipdu to coder_resp(1)
$4 12
trans msap_manager1(1) : from_idle_input_mdatreq
input mdatreq from coder_ini(1) to msap_manager1(1)
  p1 =
    id = cr
    num = zero
    data = isdu
…
```

Fichier dump obtenu après simulation de la spécification

1

```
des(0,280,138)
(0,i,1)
(0,i,2)
(0,i,3)
(0,i,4)
(1,"idisreq/dr",5)
(1,i,6)
(3,i,7)
(3,i,9)
(3,"mdatreq[{id cr, num zero,
data isdu}]/NULL",11)
(3,i,12)
(4,i,8)
(4,i,10)
(4,i,12)
(5,"idisreq/dr",13)
…
```

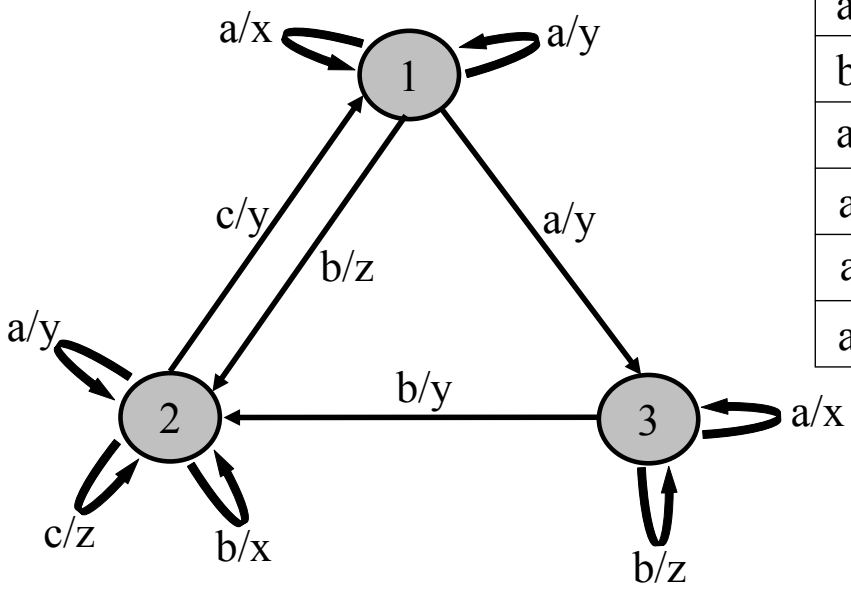
Fichier au format aldebaran

```
des(0,235,45)
(0, idisreq/dr, 2)
(0, idisreq/dr, 18)
(0, idisreq/dr, 14)
(0, "mdatreq[{id cr, num zero, data isdu}]/NULL", 3)
(0, "mdatreq[{id cr, num zero, data isdu}]/NULL", 17)
(0, "mdatreq[{id cr, num zero, data isdu}]/NULL", 34)
(0, "mdatreq[{id cr, num zero, data isdu}]/NULL", 35)
…
```

Fichier après suppression des transitions internes

2

Exemple



Invariants	Verdicts
a/?, c/z, b/{x}	*True*
b/z, a/{y}	*False*
a/x, *, b/{y, z}	*True*
a/y, ?/$\overline{\{z\}}$	*True*
a/$\overline{\{x\}}$	*False*
a/x, *, ?/$\overline{\{y\}}$	*True*

Trace

Exécution de l'implémentation

⇓ { - utilisation de points d'observations (POs)
- observation locale

Collecte trace réelle d'exécution

⇓ Pré-processing :
{ - sélection des informations nécessaires
- traduction en un format approprié

Trace exploitable

2003-06-19 16:01:52 [1] INFO:

2003-06-19 16:01:52 [1] INFO: From WTP_NULL_SESSION: Primitive Name: TR-Invoke.ind
2003-06-19 16:01:52 [1] INFO: From WTP: Ack Type: 0x01
2003-06-19 16:01:52 [1] INFO: From WTP: WTP Class: 2
2003-06-19 16:01:52 [1] INFO: From WTP: WAPAddrTuple 0x823d938
2003-06-19 16:01:52 [1] INFO: From WTP: Handle: 0
2003-06-19 16:01:52 [1] INFO: WSP Connect PDU at 0x823db38:
2003-06-19 16:01:52 [1] INFO: Octet string at 0x823db78:
2003-06-19 16:01:52 [1] INFO: data: 04 80 83 ff 7f 04 81 83
2003-06-19 16:01:52 [1] INFO: data: ff 7f 02 82 f0 02 83 ff
2003-06-19 16:01:52 [1] INFO: data: 02 84 ff
2003-06-19 16:01:52 [1] INFO: Octet string dump ends.
2003-06-19 16:01:52 [1] INFO: Session headers:

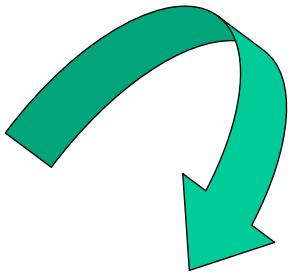
2003-06-19 16:01:52 [1] INFO: WSP PDU dump ends.
2003-06-19 16:01:52 [1] INFO: TO WTP_NULL_SESSION: Primitive Name: TR-Invoke.res
2003-06-19 16:01:52 [1] INFO: TO WTP: Handle: 0
2003-06-19 16:01:52 [1] INFO:

2003-06-19 16:01:52 [1] INFO: TO WAE: Primitive Name: S-Connect.ind
2003-06-19 16:01:52 [1] INFO: TO WAE: http_headers: 0
2003-06-19 16:01:52 [7] INFO:

2003-06-19 16:01:52 [7] INFO: FROM WAE: Primitive Name: S-Connect.res
2003-06-19 16:01:52 [7] INFO: From WAE: http_headers: 0
...

Trace réelle d'exécution

Pre-processing



Trace exploitable

#Primitive#: **TR-Invoke.ind** #Type Primitive#: In
#Primitive#: **TR-Invoke.res** #Type Primitive#: Out
#Primitive#: NULL #Type Primitive#: In
#Primitive#: **S-Connect.ind** #Type Primitive#: Out

Domaine d'application: le protocole WAP

Le protocole WAP (Wireless Application Protocol)

- Le WAP permet aux utilisateurs de terminaux mobiles à accéder et à interagir facilement a des informations Internet et des services

Application au protocole WAP

- Méthodes de test classiques fortement liées à la contrôlabilité de l'IUT
  - Si nous pouvons contrôler l'IUT, nous pouvons effectuer le test actif où des entrées sont appliquées à l'implantation et le testeur observe les sorties et donne son verdict
  - Différentes architectures sont proposées d'après le positionnement des points de contrôle et d'observation (PCOs)

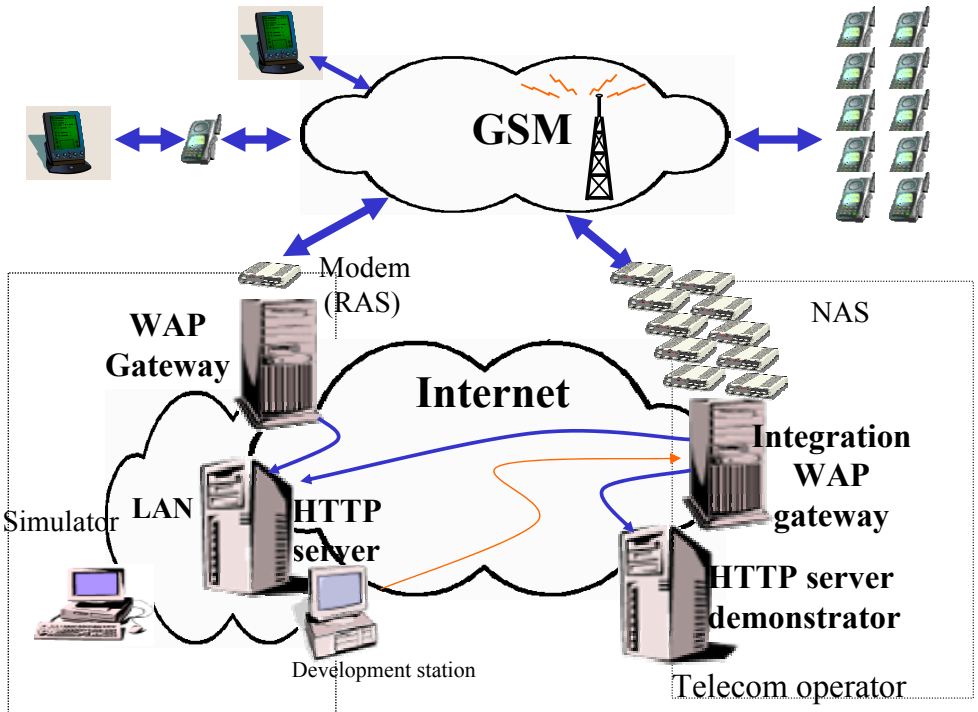
Application au protocole WAP (suite)

- Problèmes rencontrés
  - Il n 'existe aucun équipement pour effectuer des tests actifs
    - Les seuls PCOs dont nous disposions sont les boutons des téléphones cellulaires ou du simulateur de terminal téléphonique, ceci signifie que l'activité de test doit se faire manuellement
  - Aussi, la couverture des tests est limitée due à la faible contrôlabilité
  - Si nous n'avons pas de PCOs, le test passif peut être effectué par les Points d'Observation (PO) où le testeur observe les échanges de messages et décide s'il y a des erreurs dans l 'IUT

Application au protocole WAP (couches WTP et WSP) (suite)

- Mise en place d 'une pile protocolaire WAP Kannel
- Nous avons trouvé qu'il était possible de mettre de POs dans la pile WAP Kannel
  - Permet plus d'observabilité et une meilleure analyse du résultat du test
- Si nous avons des PCOs dans la pile, nous avons un contrôle complet de la passerelle WAP et aussi elle peut devenir un « lower » testeur dans une architecture de test à distance.

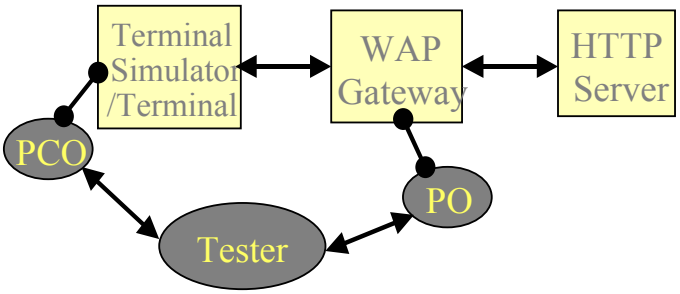
Network Architecture



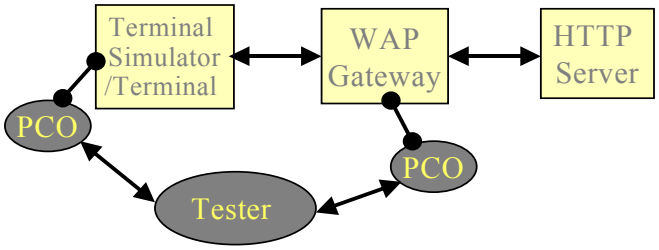
Architecture de test (avec des POs dans la passerelle)

- Mise en place des POs entre les couches WSP, WTP et WDP
- Utilisation des POs
  - Pour observer les échanges entre les couches afin de détecter des erreurs fonctionnelles

Test Architectures

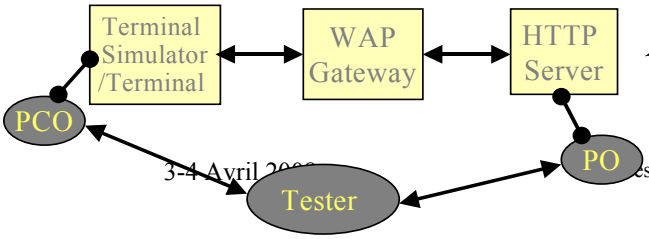


• Un PCO et un PO (dans WAP)
Objectifs
- améliorer l'observabilité
- améliorer l'analyse des résultats du test
- test d'intéroperabilité
Contrôle limité



• Deux PCOs la passerelle WAP devient un testeur actif
Objectifs
- Test de conformité
- Test d'interopérabilité

C

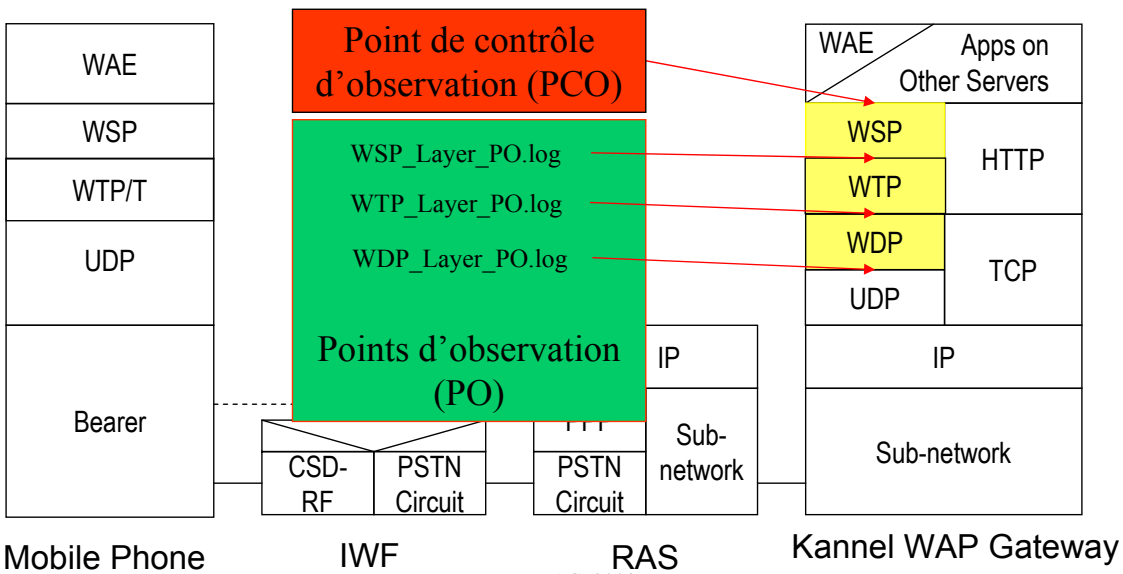


• Un PCO et un PO(coté serveur par analyse des fichiers log)

Objectifs
- test de services WAP, test d'interoperabilité end-to-end entre client et serveur
Contrôle limité

Test Architectures (suite)

- Test des couches WAP



WAP Experiments

- Nous considérons des invariants (simple et d'obligation) qui correspondent à des propriétés demandés par le WAP forum
  - Ces propriétés couvrent les aspects les plus relevants du protocole, c'est-à-dire, les phases de connexion, transfert des données et fin de connexion
  - Nous avons observé des traces d'une longueur de « 500 inputs/outputs pairs »

Invariant simple

Soit I l'ensemble des entrées, et O l'ensemble des sorties

$$Inv_S = i_1/o_1,\ldots,i_{n-1}/o_{n-1},i_n/O$$

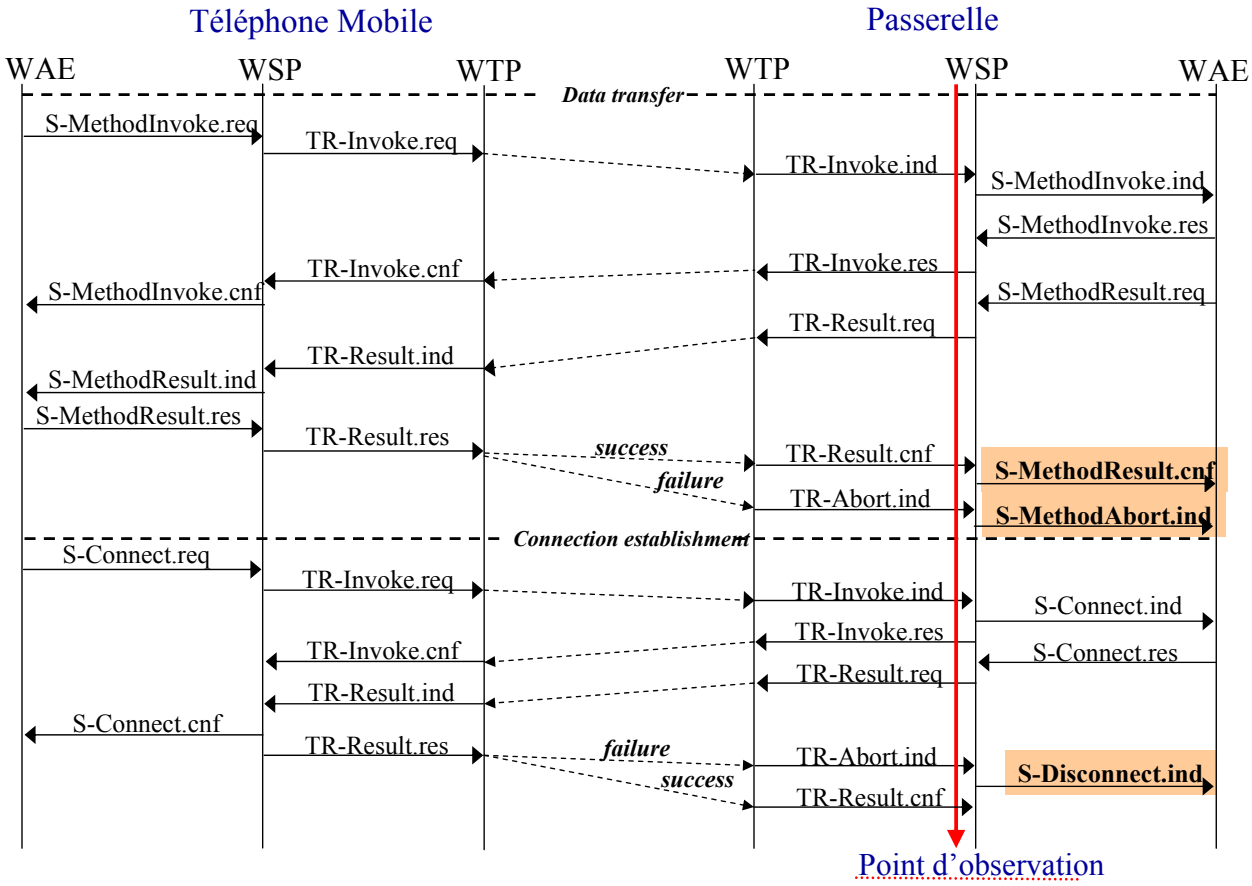
$i_k \in I$ avec $1 \leq k \leq n$,
$o_j \in O$ avec $1 \leq j \leq n-1$,
$O \subset O$, et
$\{?, *\} \notin O$ et $i_n \neq ?$

si on observe la séquence $i_1 / o_1, \ldots, i_{n-1} / o_{n-1}, i_n$ alors l'entrée i_n est suivie d'une sortie appartenant à l 'ensemble O

Demande de connexion interrompue ou transaction abandonnée

TR-Invoke.ind/?,*,TR-Abort.ind/{S-Disconnect.ind, S-MethodAbort.ind}

? : une entrée ou une sortie
* : une suite de paires d'entrées et sorties



Augmentation du pouvoir expressif des invariants

- Définition de la notion de signature contextuelle
  - Dépendance entre les invariants
  - Contraintes sur les variables
  - Notion d'état
- Opérateurs logiques
- Utilisation du langage ORBAC (logique temporelle et deontic) pour exprimer des propriétés de sécurité

Derniers travaux

- Corrélation de traces pour vérifier des propriétés globales
- Test de robustesse basé sur l'injection de fautes et le monitorage (en collaboration avec E. Martins, Univ. Campinas)

Détection d'intrusion dans les réseaux ad hoc basée sur le monitorage

Vulnérabilités des réseaux Ad Hoc

- Absence d'infrastructure
  - Routeur, serveur DNS, autorité de certification
- Medium de transmission sans fil
- Forte mobilité
- Capacités limitées
  - Bande passante, autonomie, puissance de calcul

Les types d'attaques

- Attaques courantes dans les réseaux mobiles et sans fil:
  - Sniffing (données, localisation, etc..)
  - Usurpation d'identité (Spoofing IP, ARP, etc…)
  - Modification
  - Insertion => création de boucles
  - Déni de service (DoS)

Les types d'attaques

- Attaques inhérentes au routage Ad Hoc:
  - Non coopération (Selfishness)
  - Création d'un tunnel ou de connexions privées (Wormhole)

Mécanismes de sécurité actuels

- Cryptographie et authentification:
  - prévention mais pas de détection
  - diminue le nombre d'attaques sans les éliminer
  - protection contre certains type d'attaques
  - ne permet pas de détecter et traiter les noeuds compromis

- Echec/insuffisance des propositions existantes

  => besoins de mécanismes de détection

Intrusion: definition

- Une intrusion est une tentative délibérée, non autorisée, d'accéder ou de manipuler des informations ou des systèmes, et de les rendre non fiables ou inusables (par violation des politiques de sécurité)

Systèmes de détection d'intrusion

Traditionnellement de 2 types:

- Approche par signature
  - On examine les informations échangées par les noeuds à la recherche d'attaques qui correspondent à des motifs connus (patter matching)
- Approche comportementale
  - Détection de comportements qui 's'éloignent' du comportement normal d'un noeud

Approche par signature

- Traite de détecter des comportements qui sont proches à la signature d'une intrusion connue
  - Exemple: "Network grep" – cherche des chaînes de caractères dans les connections réseaux qui pourraient indiquer un attaque en progression

Approche comportementale

- Objectif: détecter des événements "unusual"
- Analyser le réseau ou système et établir ce qui est un comportement normal
  - Appliquer de mesures statistiques ou heuristiques aux subsequent événements et déterminer s'ils sont conformes au modèle/statistique "normal"
  - Si les événements se situent en dehors d'une fenêtre de probabilité "normal" alors il faut générer une alarme

Paradigme faux positif / négatif

- Faux positif
  - On appelle faux positif quand le système produit une alerte incorrecte
    => perte de temps et de ressources
  - l'objectif est d'obtenir 0% de faux positifs
- Faux négatif
  - Un attaque réel reste non détecté

Systèmes de détection d'intrusion

- Inconvénients de l'approche par signature:
  - Ne permet de détecter que des attaques connues
  - Difficile de maintenir les signatures à jour
  - Absence d'entité centralisée pour surveiller le trafic
  - Peut être dupée
- Inconvénients de l'approche comportementale:
  - Pas de distinction claire entre comportements normal et anormal
  - Nécessite beaucoup de données
  - Efficacité inférieure
  - Trop de faux positifs

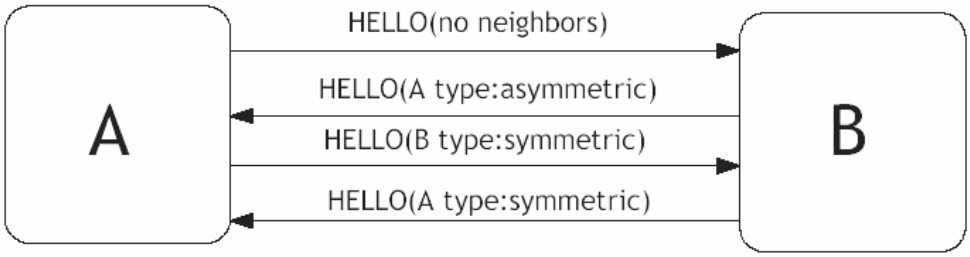
Le protocole OLSR (Optimized Link State Routing)

- Protocole pro-actif
  - Des paquets de contrôle sont périodiquement diffusés dans le réseau
    => mise à jour continue des tables de routage
- Utilisation de 'Relais Multi Points'
  - Limite l'inondation à l'intérieur du réseau
  - Les routes sont optimales
  - Les routes sont immédiatement disponibles

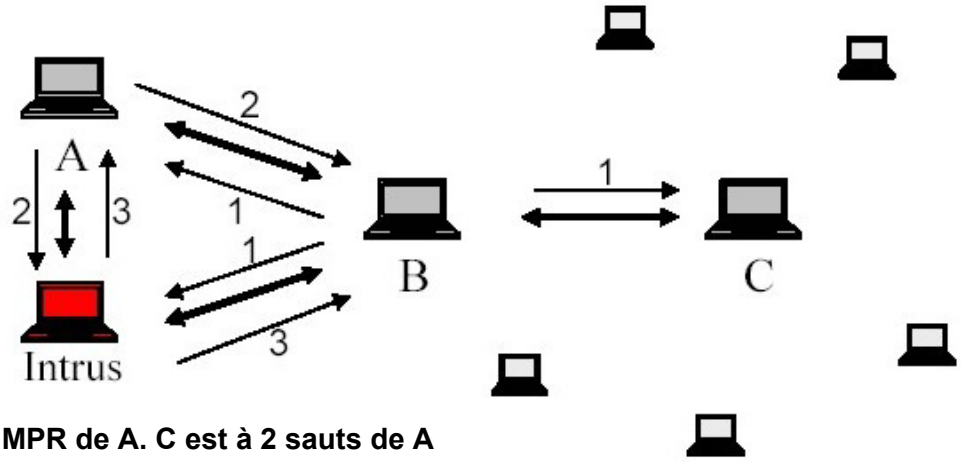
Neighbour discovery

- Emission périodique de paquets *hello*
  - Les *hello* contiennent la liste des nœuds entendus et le type de lien

=> Les nœuds connaissent leurs voisins et les voisins à deux sauts



Exemple d'attaque sur OLSR



B est MPR de A. C est à 2 sauts de A

1. Envoi de messages Hello par B

2. Envoi de messages Hello par A

3. Insertion d'un message Hello par l'Intrus annonçant à A, B, C un lien symétrique

Conséquences :

- **Sélection de l'Intrus comme MPR par A et B**
- **Le trafic de A vers C passera par l'Intrus**

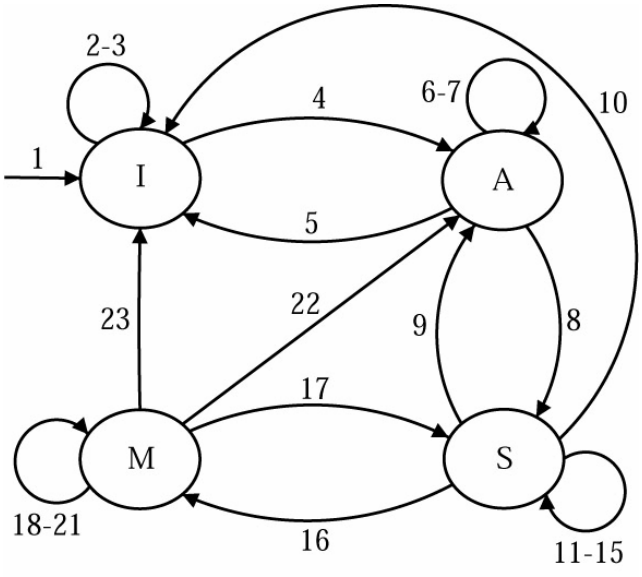
Machines à états finis étendues

- Spécification d'OLSR sous forme d'EFSM (Extended Finite State Machine)
- Les EFSM (Extended Finite State machine) sont caractérisés par:
  - des événements d'E/S avec ou sans paramètres
  - un prédicat à satisfaire
  - des actions à effectuer

Machines à états finis étendues

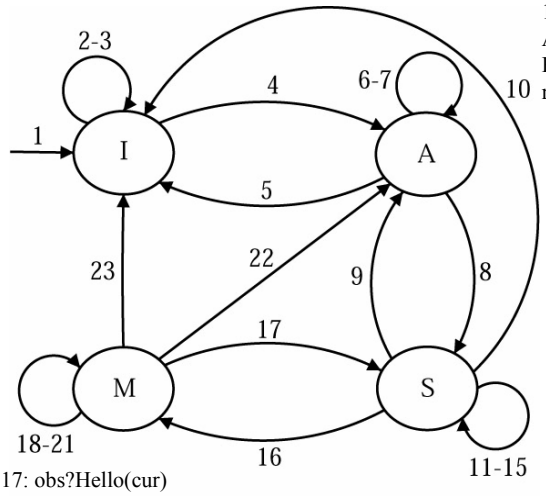
- Transitions définies par :
  - Une entrée et une sortie (avec ou sans paramètres)
  - Un prédicat sur des variables
  - Des actions sur des variables
- Traces d'exécution (séquence de couples d'entrée/sortie) du système sous test

EFSM d'OLSR



OLSR EFSM derivée du RFC 3626

Exemple



10: UpdateTimerOut
A: reset UpdateTimer;
Remove(obs,AsymList);
reset SentHello; remove(obs,MprList)\\

8: obs?Hello(cur)
P: (cur=ASYM AND SentHello=true)
OR (cur=SYM AND obs$\in$AsymList)\\
A: reset UpdateTimer

11: HelloTimerOut / cur!Hello(obs)
A: set obs=SYM; reset HelloTimer; remove(obs,MprList)

12: HelloTimerOut / cur!Hello(obs)
A: set obs=MPR; reset HelloTimer; add(obs,MprList)

13: cur!Data()
P: obs$\in$MprList

14: obs?TC(cur)
P: cur=MPRSEL AND obs$\in$MprList

15: obs?Hello(cur)
P: cur=SYM OR cur=ASYM

17: obs?Hello(cur)
P: cur=SYM OR cur=ASYM
A: reset UpdateTimer; Remove(obs,MprSelList)

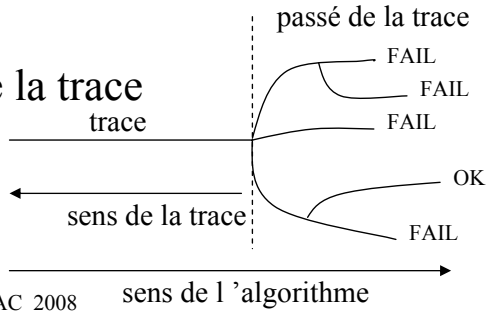
16: obs?Hello(cur)\\
P: cur=MPR\\
A: Add(obs,MprSelList);
reset UpdateTimer; reset TcTimer

Processus de détection

- Le processus de vérification/détection consiste à comparer les traces d'entrées/sorties (messages reçus et envoyés) avec la spécification
  - la trace doit être acceptée comme un mot de l'EFSM
- La comparaison est effectuée à l'aide d'un algorithme de recherche en arrière (backward checking) que nous avons défini

Backward checking : principe général

- Idée : « pour comprendre le présent, il faut étudier le passé »
- Backward checking : algorithme en deux temps :
  - remontée de la trace d'événements issue de l'implémentation
  - remontée dans le passé de la trace



Backward checking

L'algorithme est principalement composé de 2 phases:

- A partir d'un événement donné, on parcourt la trace en arrière jusqu'à trouver les différentes configurations initiales possibles
- A partir de ces configurations, on explore en arrière, chaque chemin possible de la spécification jusqu'à la fin ou jusqu'à trouver une erreur
  - Un algorithme "d'élagage" permet de réduire l'espace de recherche

Exemple

Insertion de faux messages ‘Hello ’

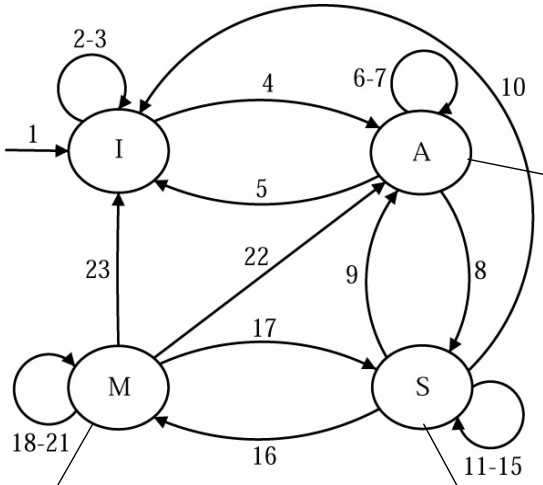
- Une trace correspondante possible est:

  (start)
  HelloTimerOut / cur!Hello()
  UpdateTimerOut
  obs?Hello(cur) / cur=SYM

I. 1) Partant du dernier évenement: recherche des transitions correspondantes (8,15,17)

Exemple

=> correspond avec les transitions n° 8,15 &17



10: UpdateTimerOut
A: reset UpdateTimer;
Remove(obs,AsymList);
reset SentHello; remove(obs,MprList)

8: obs?Hello(cur)
P: (cur=ASYM AND SentHello=true)
OR (cur=SYM AND obs∈AsymList)
A: reset UpdateTimer

11: HelloTimerOut / cur!Hello(obs)
A: set obs=SYM; reset HelloTimer; remove(obs,MprList)

12: HelloTimerOut / cur!Hello(obs)
A: set obs=MPR; reset HelloTimer; add(obs,MprList)

13: cur!Data()
P: obs∈MprList

14: obs?TC(cur)
P: cur=MPRSEL AND obs∈MprList

17: obs?Hello(cur)
P: cur=SYM OR cur=ASYM
A: reset UpdateTimer;
Remove(obs,MprSelList)

16: obs?Hello(cur)
P: cur=MPR
A: Add(obs,MprSelList); reset TcTimer

15: obs?Hello(cur)
P: cur=SYM OR cur=ASYM
A: reset UpdateTimer

Exemple

I.2) Recherche des configurations antérieures possibles:

State: A; Parameters: cur = SYM, obsЄAsymList

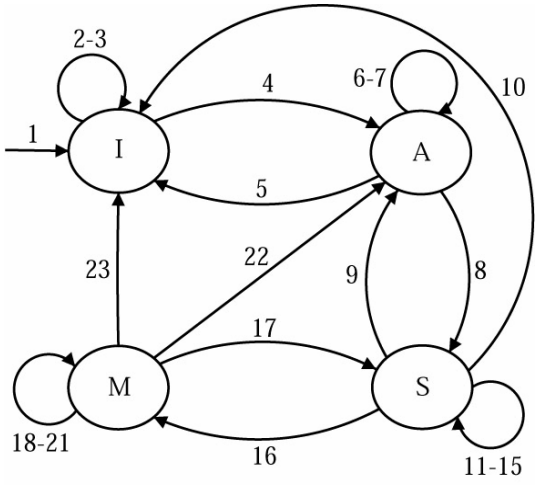
State: S; Parameters: cur = SYM

State: M; Parameters: cur = SYM, obsЄMprList

II.1) On recommence le procédé sur la transition précédente (UpdateTimer)

Exemple

=> Aucune corresponds!



10: UpdateTimerOut
A: reset UpdateTimer;
Remove(obs,AsymList);
reset SentHello; remove(obs,MprList)

8: obs?Hello(cur)
P: (cur=ASYM AND SentHello=true)
OR (cur=SYM AND obsЄAsymList)

11: HelloTimerOut / cur!Hello(obs)
A: set obs=SYM; reset HelloTimer; remove(obs,MprList)

12: HelloTimerOut / cur!Hello(obs)
A: set obs=MPR; reset HelloTimer; add(obs,MprList)

13: cur!Data()
P: obsЄMprList

14: obs?TC(cur)
P: cur=MPRSEL AND obsЄMprList

15: obs?Hello(cur)
P: cur=SYM OR cur=ASYM

17: obs?Hello(cur)
P: cur=SYM OR cur=ASYM
A: Remove(obs,MprSelList)

16: obs?Hello(cur)
P: cur=MPR
A: Add(obs,MprSelList);
reset TcTimer

Exemple

II.2) Il n'y a pas de transition satisfaisant les contraintes

(les transitions qui contiennent l'évènement *UpdateTimerOut* ne mènent pas aux états A, S or M)

=> Violation de la spécification par une erreur de transfert!

Discussion

- Permet de détecter de manière efficace des attaques on link state (les plus importants pour les protocoles proactifs)
- Pas de faux positifs
- Approache exhaustif

- Pas d'identification des erreurs
  - erreurs de conformité / failles de sécurité?
- Ne permet pas de détecter des attaques qui ne violent pas la spécification (par ex. DoS)

Perspectives

- Utiliser de manière complémentaire une approche par signature
  - Identification des erreurs de conformité
  - Détection d'attaques spécifiques (DoS)
  - Découverte et enregistrement de nouvelles vulnérabilités

Approche par Signature

- Extraction des propriétés relevantes du RFC
- Transformation dans la forme d' invariantes

1. Vérification que la spécification satisfait l' invariante
2. Vérification que les traces respectent l' invariante

Exemples d'invariantes de conformité

- « Every node in the symetric 2-hops neighborhood of N must have a symmetric link towards MPR(N) »
- « A node with null *willingness* must never be MPR of any node »

Problèmes ouverts

- Améliorer l'efficacité de l'algorithme (real-time use)
- Comment réagir face à des misbehaving nodes?
- Comment identifier, exclure des noeuds malicieux?

=> Pas de réponse pour le moment

- QUESTIONS?