BIULETYN INSTYTUTU AUTOMATYKI I ROBOTYKI
NR 27, 2009

DEPENDITA – narzędzie wspomagające badanie wiarygodności aplikacji rozproszonych

Sebastian CZECH, Zbigniew ZIELIŃSKI

Zakład Teleinformatyki, Instytut Teleinformatyki i Automatyki WAT,
ul. Kaliskiego 2, 00-908 Warszawa

STRESZCZENIE: W publikacji przedstawiono narzędzie programowe *DEPENDITA*, które zostało zbudowane w Instytucie Teleinformatyki i Automatyki WAT. Narzędzie to wspomaga badanie wiarygodności aplikacji rozproszonych poprzez programowe wszczepianie błędów i przeznaczone jest do testowania aplikacji rozproszonych, wykorzystujących standard *Sun RPC*. Zaletą *DEPENDITA* jest zastosowana technika przechwytywania wiadomości przesyłanych pomiędzy poszczególnymi elementami testowanej aplikacji, która nie wymaga wprowadzania żadnych zmian w kodzie źródłowym testowanego systemu.

SŁOWA KLUCZOWE: wiarygodność, wszczepianie błędów, zdalne wywołanie procedur, Sun RPC

1. Wprowadzenie

Wiarygodność systemów komputerowych, która stanowi cechę systemu, określającą jego zdolność do świadczenia usług w sposób niezawodny, odporny na błędy i bezpieczny, jest przedmiotem wielu badań i publikacji, na przykład [1], [2], [8], [10]. Zapewnienie odpowiedniego poziomu wiarygodności jest niezwykle istotne zarówno dla projektantów systemów, jak i użytkowników aplikacji rozproszonych. W celu oceny poziomu wiarygodności aplikacji rozproszonych koniecznie jest stosowanie narzędzi wspomagających ich testowanie [3], [4], [6], [9]. Ze względu na liczbę i różnorodność używanych technologii koniecznie jest nie tylko rozwijanie istniejących narzędzi, ale także projektowanie nowych rozwiązań, które dostarczać będą nowych możliwości. W artykule przedstawiono autorskie narzędzie o nazwie *DEPENDITA*, opracowane

w Instytucie Teleinformatyki i Automatyki WAT, które wspomaga testowanie aplikacji rozproszonych poprzez programowe wszczepianie błędów. Prezentowane narzędzie umożliwia również ocenę wiarygodności badanych aplikacji, dostarczając możliwości określenia takich atrybutów wiarygodności jak niezawodność, dostępność i odporność na błędy, co wyróżnia je spośród innych tego typu narzędzi.

2. Wszczepianie błędów

W procesie testowania wiarygodności systemów rozproszonych koniecznie jest sprawdzenie zachowania ich poszczególnych elementów w przypadku wystąpienia określonych rodzajów błędów. Może to zostać zrealizowane w sposób dynamiczny poprzez tzw. wszczepianie błędów, co pozwala nie tylko zidentyfikować defekty w systemie i przeanalizować działanie aplikacji w trakcie procesu wszczepiania, ale także wymusić uruchomienie odpowiednich mechanizmów zapewniających jej wiarygodność (np. maskowanie wad poprzez zastosowanie nadmiarowości fizycznej czy też usuwanie defektów w trakcie działania systemu).

2.1. Programowe i fizyczne metody wprowadzania uszkodzeń

Wszczepianie błędów może być realizowane poprzez wprowadzanie fizycznych uszkodzeń (np. uszkodzenie medium transmisyjnego, którym połączono urządzenia, na których uruchomiono elementy składowe badanego systemu rozproszonego) lub w sposób programowy (np. zmiana danych przesyłanych pomiędzy klientem a serwerem). Z kolei metody programowe mogą być wykonywane w trakcie budowy (kompilacji) testowanej aplikacji bądź w trakcie jej działania.

Wybór odpowiedniej techniki testowania poprzez wszczepianie błędów uzależniony jest od rodzaju błędów, czasu ich występowania w systemie oraz oczekiwanych rezultatów. Przykładowo, gdy chcemy wprowadzić do systemu błędy, które mają występować przez dłuższy czas, należy zastanowić się nad wprowadzeniem fizycznych uszkodzeń. Natomiast w przypadku potrzeby modyfikowania danych przekazywanych w testowanym systemie należy wziąć pod uwagę technikę programowego wszczepiania błędów.

W porównaniu do testowania poprzez wprowadzanie fizycznych uszkodzeń, programowe wszczepianie błędów jest mniej kosztowne, a ponadto pozwala uruchomić aplikację wstrzykującą bezpośrednio na docelowym

systemie. Pomimo elastyczności metod programowych, posiadają one kilka wad związanych z szybkością ich działania, a także brakiem możliwości wstrzykiwania błędów w miejsca niedostępne dla oprogramowania.

2.2. Charakterystyka wybranych narzędzi umożliwiających wszczepianie błędów

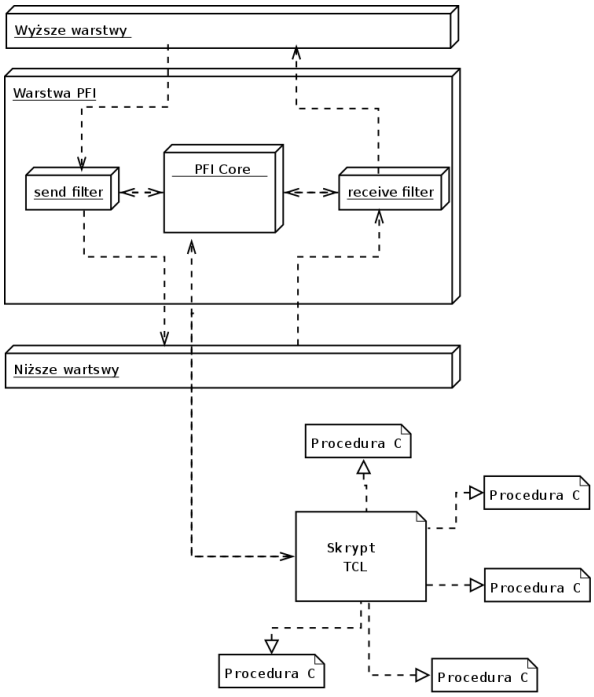
Spośród istniejących systemów wszczepiących błędy w sposób programowy w środowisku rozproszonym wybrano aplikację *Orchestra* oraz *Fail Cluster Implementation*, które nie tylko różnią się architekturą, ale także zastosowanymi technikami wprowadzania uszkodzeń oraz przechwytywania danych przesyłanych w badanym środowisku. Przegląd wybranych systemów oraz wyróżnienie ich cech charakterystycznych posłużyły do zaprojektowania własnego narzędzia, które dostarcza nowych funkcji, niedostępnych w przedstawionych narzędziach.

2.2.1. *Orchestra* [3, 4]

Za pomocą systemu *Orchestra* błędy są wszczepiane w środowisku rozproszonym poprzez wprowadzanie określonych zmian w danych (pakietach sieciowych przesyłanych pomiędzy elementami badanej aplikacji). Architekturę systemu przedstawiono na rys. 1, na którym wyróżniono warstwę *PFI*, odpowiedzialną za wszczepianie błędów i znajdującą się pomiędzy warstwą aplikacji a warstwą transportową. Na każdym z urządzeń, na których pracuje testowana aplikacja, konieczne jest uruchomienie systemu *Orchestra*, który dla każdego wysyłanego komunikatu przy pomocy *PFI* uruchamia skrypt *send filter*, natomiast dla każdego otrzymanego komunikatu – skrypt *receive filter*. Wymienione skrypty mogą wykonywać trzy rodzaje operacji na przekazywanych komunikatach:

- filtrowanie komunikatów, tj. przechwytywanie oraz kontrolowanie ich zawartości;
- manipulację wiadomościami poprzez wykonanie jednej z czynności:
  - wprowadzanie opóźnień w dostarczeniu wiadomości,
  - blokowanie ich przesyłania,
  - zmianę kolejności ich dostarczenia,
  - duplikowanie komunikatów,
  - modyfikowanie przesyłanych danych,
- wprowadzanie komunikatów, tj. tworzenie nowych wiadomości.

Dla każdej z uruchomionych instancji systemu *Orchestra* konieczne jest przygotowanie odpowiednich skryptów w języku *TCL*, za pomocą którego definiowane są operacje wykonywane przez filtry uruchamiane przez *PFI*. W niektórych przypadkach niewątpliwą zaletą może okazać się możliwość rozbudowy istniejących funkcji, manipulujących komunikatami poprzez implementowanie ich rozszerzeń w języku *C*.



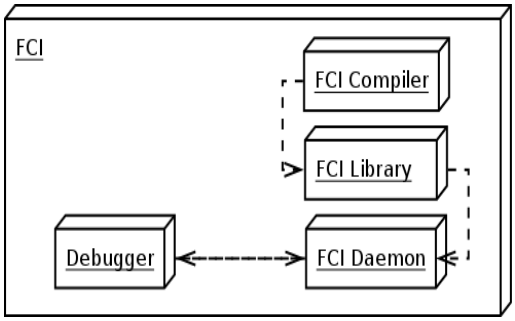
Rys. 1. Architektura systemu *Orchestra* [4]

Autorzy systemu *Orchestra* w ciekawy sposób rozwiązali problem przechwytywania danych przesyłanych w badanym środowisku. Zaimplementowane zostały specjalne biblioteki dołączane w trakcie kompilacji testowanej aplikacji, które wymuszają zmianę wykonywania odpowiednich funkcji systemowych używanych do przesyłania wiadomości, np. biblioteka *libpfisock.a*, która dołączana jest do aplikacji wykorzystujących do komunikacji interfejs gniazd, powoduje zmianę sposobu wykonywania takich funkcji jak *connect()*, *send()* czy też *recv()*.

Ze względu na architekturę oraz sposób definiowania skryptów system *Orchestra* jest elastycznym oraz skalowalnym rozwiązaniem, za pomocą którego możliwe jest wszczepianie błędów w systemach opartych na różnych technologiach rozproszonych.

2.2.2. *Fail Cluster Implementation* [6]

System *FCI* (*ang. Fail Cluster Implementation)*, którego architekturę przedstawiono na rys. 2, składa się z trzech podstawowych komponentów – kompilatora, bibliotek oraz aplikacji wszczepiającej błędy. Pierwszy ze składników kompiluje scenariusze testów, które są definiowane w języku *FAIL*, generując kod źródłowy w języku *C++* oraz pliki konfiguracyjne, które są następnie łączone z dostarczonymi wraz z systemem *FCI* bibliotekami odpowiedzialnymi za komunikację z *debuggerem*, który w środowisku testowym wykorzystywany jest do wstrzymywania działania badanego systemu. Wygenerowane archiwa zawierające kody źródłowe wraz z bibliotekami są kompilowane na każdym z urządzeń, na których uruchomiono testowany system. Skompilowana aplikacja do wstrzykiwania błędów odpowiedzialna jest za realizację zadań zdefiniowanych w scenariuszu testów, komunikację z innymi aplikacjami systemu *FCI* wszczepiającymi błędy, a także sterowanie testowanym środowiskiem przy pomocy *debuggera*.



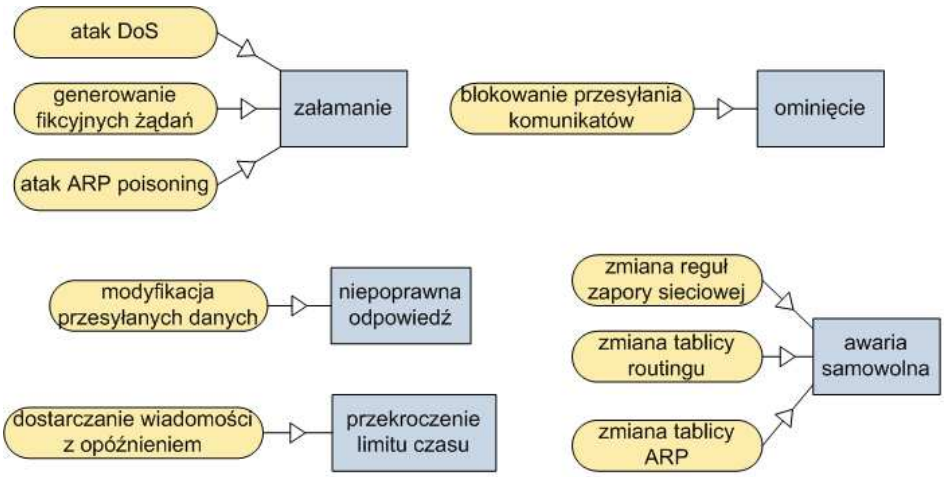
Rys. 2. Architektura systemu *Fail Cluster Implementation* [6]

Przedstawione narzędzia w różny sposób realizują programowe wszczepianie błędów w badanym środowisku rozproszonym. Zaletą systemu *Orchestra* jest brak konieczności stosowania dodatkowych narzędzi takich jak *debugger*, natomiast zaletą aplikacji *FCI* jest zbudowany mechanizm wzajemnej komunikacji aplikacji wstrzykujących błędy czy też brak konieczności przebudowy (rekompilacji) badanego systemu.

Projektując autorskie rozwiązanie, nie tylko zwrócono uwagę na zalety już istniejących narzędzi, ale także wprowadzone nowe funkcje, takie jak możliwość oceny wiarygodności badanych aplikacji czy też opracowanie nowych mechanizmów przechwytywania danych przesyłanych pomiędzy poszczególnymi elementami testowanych aplikacji.

2.3. Techniki przechwytywania danych przesyłanych w środowisku rozproszonym

W trakcie realizacji projektu narzędzia wspomagającego badanie wiarygodności aplikacji rozproszonych szczególną uwagę zwrócono na systemy oparte na standardzie *Sun RPC*, który jest jedną z wielu realizacji mechanizmu wywołań procedur zdalnych. Z kolei dla wybranej grupy systemów wyróżniono rodzaje błędów prowadzących do wystąpienia awarii, które przedstawiono na rys. 3 i wykorzystano na etapie projektu systemu wszczepiającego błędy w sposób programowy. Na rysunku 3 błędy przedstawiono przy pomocy figur o owalnym kształcie, awarie – w postaci zwykłych prostokątów, natomiast związek pomiędzy błędem, który prowadzi do wystąpienia określonej awarii, zobrazowano przy pomocy łuku.



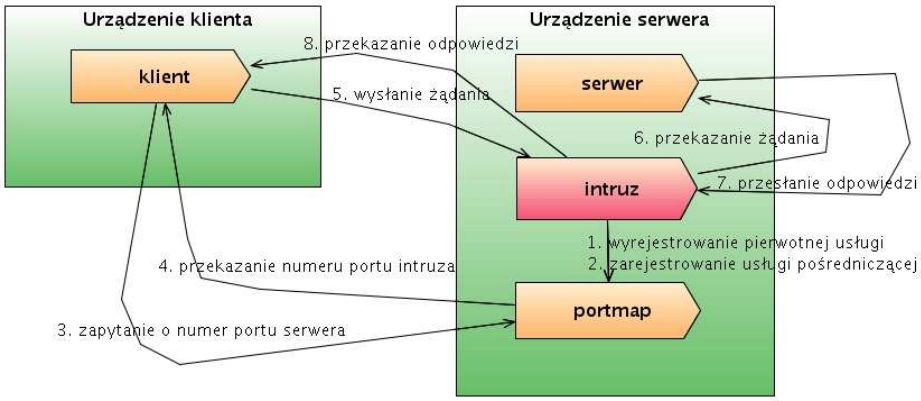
Rys. 3. Wybrane rodzaje błędów prowadzących do wystąpienia awarii w aplikacjach opartych na architekturze klient-serwer

W celu wprowadzania wyróżnionych uszkodzeń opracowano mechanizmy pozwalające przechwytywać dane przekazywane pomiędzy poszczególnymi elementami składowymi badanej aplikacji poprzez:

- wyrejestrowanie usługi pierwotnej w programie *portmap* (który odpowiedzialny jest za dostarczenie klientom informacji o usługach udostępnianych przez serwery) i zarejestrowanie usługi pośredniczącej (intruza wszczepiającego błędy) w komunikacji pomiędzy klientami a serwerem,

- zmianę numerów portów docelowych oraz źródłowych w przekazywanych pakietach sieciowych w taki sposób, by były one odbierane przez intruza wstrzykującego błędy.

Pierwszy z zaprojektowanych sposobów realizowany jest poprzez wyrejestrowanie usługi w *portmap* i zarejestrowanie usługi pośredniczącej, która odpowiedzialna jest za wstrzykiwanie błędów (rys. 4). Klient, który wywołuje określoną procedurę zdalną, wysyła zapytanie do programu *portmap* o numer portu serwera. Odpowiedź, którą otrzymuje klient, zawiera numer portu wykorzystywanego przez intruza. Następnie klient wysyła żądanie do intruza, który wszczepia określone błędy, a następnie przekazuje żądanie do serwera. Serwer realizuje określone działania i wysyła odpowiedź do intruza, który przekazuje ją do klienta.



Rys. 4. Przechwytywanie komunikatów poprzez wyrejestrowanie usługi pierwotnej i zarejestrowanie usługi pośredniczącej w *portmap*

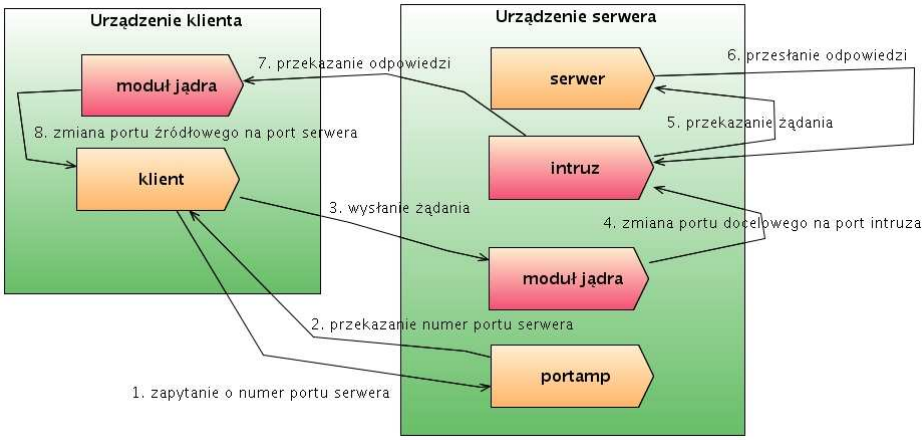
Drugi sposób polega na wykonywaniu zmian numerów portów w pakietach sieciowych odbieranych przez interfejsy sieciowe urządzeń, na których uruchomiono (rys. 5):

- aplikację serwera – zmieniany jest numer portu docelowego na port wykorzystywany przez intruza,
- aplikację klienta – zmieniany jest numer portu źródłowego na port używany przez serwer.

Zaprojektowane metody przechwytywania komunikatów w środowisku opartym na mechanizmie wywołań procedur zdalnych:

- nie wymagają przebudowywania testowanej aplikacji;

- nie narzucają potrzeby wprowadzania zmian w kodzie źródłowym badanego systemu;
- nie wpływają na inne aplikacje uruchomione na urządzeniach, na których pracują klienci lub serwer;
- nie wymagają stosowania dodatkowych narzędzi takich jak *debugger*, które przerywają działanie testowanej aplikacji.



Rys. 5. Przechwytywanie komunikatów poprzez zmianę numerów portów w pakietach sieciowych przez ładowalny moduł jądra

Ponadto istotną zaletą obu metod przechwytywania komunikatów jest ich heterogeniczność, tzn. sposób oparty na zarejestrowaniu usługi pośredniczącej w *portmap* może zostać dostosowany do innej z popularnych technologii budowy systemów rozproszonych, tj. zdalnego wywołania metod – *Java RMI*, który wykorzystuje usługę *rmiregistry*, pełniącą podobną rolę co *portmap* w standardzie *Sun RPC*. Druga z metod stosująca dodatkowy moduł jądra zmieniający numery portów docelowych oraz źródłowych może zostać dostosowana do aplikacji wykorzystujących gniazda strumieniowe bądź datagramowe.

3. Autorski system wspomagający badanie aplikacji rozproszonych

3.1. Architektura narzędzia *DEPENDITA*

Zbudowane narzędzie *DEPENDITA* do wszczepiania błędów jest systemem rozproszonym, który składa się z trzech elementów: modułu głównego,

demona oraz ładowalnego modułu jądra systemu *Linux*. Pierwszy z elementów jest podstawowym składnikiem środowiska, który współpracuje z wieloma demonami, natomiast moduł jądra wykorzystywany jest tylko w jednej z metod przechwytywania komunikatów do wprowadzania zmian numerów portów w pakietach sieciowych.

3.1.1. Moduł główny

Moduł główny odpowiedzialny jest za zarządzanie całym środowiskiem, definiowanie zadań wstrzykiwania błędów, konfigurację poszczególnych elementów składowych, monitorowanie procesu wstrzykiwania, a także ocenę wiarygodności testowanej aplikacji rozproszonej.

Użytkownik za pomocą modułu głównego konfiguruje środowisko testowe poprzez określenie adresów poszczególnych elementów składowych badanej aplikacji rozproszonej, numeru oraz wersji usługi *RPC*, a także parametrów procedur zdalnych, tj. typów danych argumentów oraz zwracanych wyników. Ponadto, za pomocą modułu głównego, tester aktywuje oraz deaktywuje uruchomione demony, a także dostarcza informacji o usłudze *RPC*, które są wykorzystywane w trakcie modyfikacji przesyłanych komunikatów.

Tester, który za pomocą interfejsu graficznego modułu głównego określa, jakiego rodzaju błędy są wszczepianie, definiuje w sposób bezpośredni parametry zadań lub wykorzystuje mechanizm losowego generowania zadań poprzez wybór odpowiednich rozkładów prawdopodobieństwa wykorzystywanych do określenia liczby oraz rodzaju zadań, czasu przerwy pomiędzy rozpoczęciem realizacji dwóch kolejnych zadań, adresów badanych urządzeń czy też liczby komunikatów, które zostaną wygenerowane lub zmodyfikowane.

Niektóre z wyszczególnionych na rys. 3 błędów wstrzykiwane są za pomocą modułu głównego, który umożliwia:

- Wykonywanie *ataku DoS* poprzez generowanie dużej liczby pakietów *HTTP*, *FTP* lub *ICMP*, wysyłanych do określonego urządzenia. Użytkownik określa rodzaj pakietów (np. *HTTP Request* lub *HTTP Response*), ich rozmiar, liczbę pakietów do wygenerowania, czas przerwy pomiędzy wysłaniem dwóch kolejnych pakietów, a także flagi protokołu *TCP* (w przypadku generowania komunikatów *HTTP* lub *FTP).*
- Wykonywanie ataku *ARP Poisoning* poprzez generowanie pakietów *ARP Reply*, które będą zawierać nieprawidłowy adres *MAC* dla określonego adresu *IP*.

W trakcie badania wiarygodności aplikacji rozproszonej, użytkownik monitoruje wykonywane zadania, które mogą zostać w dowolnym momencie

przerwane. W przypadku ich zakończenia lub wstrzymania przez testera, tworzone są raporty przedstawiające szczegółowe informacje o zrealizowanych działaniach.

Niewątpliwą zaletą zbudowanego narzędzia jest możliwość oceny wiarygodności badanej aplikacji rozproszonej, która realizowana jest przy wykorzystaniu przechwyconych przez demony wiadomości przesyłanych między klientami a serwerem testowanej aplikacji, a także utworzonych raportów, zawierających informacje o wykonanych zadaniach wstrzykiwania błędów. Zarówno przechwycone komunikaty, jak i raporty, przechowywane są w bazie danych. W trakcie oceny wiarygodności moduł główny wyznacza wartości liczbowe współczynników zdefiniowanych dla wybranych atrybutów wiarygodności, np. obliczana jest średnia liczba awarii na jednostkę czasu czy też stopień tolerancji błędów. Wartości liczbowe obliczane są przy pomocy opracowanego algorytmu, który:

- Grupuje przechwycone przez demony dane w cztery kategorie:
  - wiadomości wysyłane przez klientów w czasie, gdy nie wstrzykiwano błędów;
  - komunikaty generowane przez serwery w czasie, gdy nie wszczepiano błędów;
  - wiadomości wysyłane przez klientów w trakcie wszczepiania błędów;
  - komunikaty generowane przez serwery w trakcie wstrzykiwana błędów.
- Wyszukuje wiadomości utworzone przez klientów wtedy, gdy realizowano choć jedno zadanie wszczepiania błędów, a następnie znajduje odpowiadające im komunikaty, które wysłano wtedy, gdy nie realizowano żadnego z zadań.
- Porównuje odpowiedzi serwera dla obu znalezionych komunikatów i w zależności od tego, czy przechwycono poszukiwane wiadomości, a także czy różnią się one od siebie, aktualizuje odpowiednie wartości współczynników, np. liczbę poprawnie obsłużonych żądań klienta przez serwer w trakcie wszczepiania błędów.
- W podobny sposób sprawdza odpowiedzi serwera poprzez porównanie odpowiednich żądań klientów.

Moduł główny zaimplementowany został w języku *C++*, natomiast interfejs graficzny użytkownika zbudowano przy wykorzystaniu biblioteki *Qt*. Poszczególne zadania wszczepiania błędów, odbieranie raportów oraz przechwyconych przez demony danych, a także ocena wiarygodności badanej aplikacji, realizowane są przy wykorzystaniu wątków *QThread*, które są elementem składowym biblioteki *Qt*. Uruchomione wątki komunikują się

z głównym procesem aplikacji, który odpowiedzialny jest za obsługę interfejsu graficznego za pomocą mechanizmu sygnałów oraz slotów (tj. metod obsługujących sygnały), dostępny jest on w bibliotece *Qt* i został oparty na wywołaniach zwrotnych. Ponadto w celu synchronizacji dostępu do współdzielonych zasobów przez poszczególne wątki stosowano semafory *QSemaphore*, które również dostarczane są z biblioteką *Qt*.

3.1.2. Demon

Demon wykorzystywany jest do przechwytywania komunikatów przesyłanych między klientami a serwerem testowanej aplikacji, a także w realizacji zadań wszczepiania błędów i wysyłania raportów do modułu głównego. Został on zaimplementowany w języku *C*, lecz nie posiada interfejsu użytkownika, bowiem wszelkie operacje związane z badaniem wiarygodności definiowane są za pomocą modułu głównego.

Większość z wyszczególnionych na rys. 3 błędów wstrzykiwana jest za pomocą demonów, za pomocą których możliwe jest:

- Modyfikowanie komunikatów przekazywanych między klientami a serwerem – zmiany dotyczą wiadomości przesyłanych przez określone urządzenia w wybranym kierunku (tzn. od klienta do serwera lub odwrotnie). Dane mogą być modyfikowane w sposób losowy lub określony przez użytkownika.
- Przechwytywanie wiadomości i dostarczanie ich do adresata z określonym opóźnieniem. Użytkownik określa liczbę komunikatów, których przekazanie zostanie opóźnione, a także czas przetrzymywania wiadomości przez demona.
- Tworzenie spreparowanych klientów generujących określoną liczbę żądań wysyłanych do serwera. Pakiety wysyłane są ze zdefiniowaną przez użytkownika częstotliwością przez wybraną liczbę wątków.
- Blokowanie przesyłanych komunikatów między klientami i serwerem. Użytkownik określa liczbę oraz rodzaj zablokowanych wiadomości przesyłanych przez określone urządzenia w wybranym kierunku.
- Rejestrowanie nieistniejących usług w programie *portmap*. Usługi rejestrowanie są z wybraną przez użytkownika częstotliwością przez określoną liczbę wątków.
- Wstrzykiwanie błędów prowadzących do wystąpienia awarii samowolnych poprzez odpowiednią zmianę tablicy routingu, reguł zapory sieciowej *iptables* lub tablicy *ARP*.

W przypadku stosowania metody przechwytywania komunikatów poprzez zarejestrowanie usługi pośredniczącej w programie *portmap*, demon odpowiedzialny

jest także za wprowadzenie odpowiednich zmian w liście zarejestrowanych usług, a także za przywrócenie pierwotnych ustawień po zakończeniu działań.

Zadania realizowane przez demony wykonywane są przy pomocy wątków standardu *POSIX*, natomiast dostęp do współdzielonych przez wątki zasobów (np. lista zadań wszczepiania błędów) synchronizowany jest za pomocą semaforów standardu *POSIX*. Dla każdego odebranego żądania klienta demon uruchamia nowy wątek jego obsługi, za pomocą którego realizuje następujące funkcje:

- Tworzy odpowiednie gniazda sieciowe (strumieniowe dla protokołu *TCP* lub datagramowe dla protokołu *UDP*) do komunikacji z klientem.
- Analizuje odebrany pakiet poprzez sprawdzenie zgodności formatu przesyłanych danych ze strukturą pakietów dla protokołu *RPC*. W przypadku braku zgodności, przesyła wiadomości do adresata, natomiast w przeciwnym wypadku:
  - realizuje wszystkie zadania wszczepiania błędów, które oznaczone są jako aktywne;
  - przesyła przechwycony komunikat do modułu głównego (niezależnie od tego, czy realizowane jest jakiekolwiek zadanie wstrzykiwania błędów);
  - po zakończeniu lub wstrzymaniu wykonywania zdefiniowanych przez użytkownika zadań, demon wysyła do modułu głównego raporty.

3.1.3. Ładowalny moduł jądra systemu *Linux*

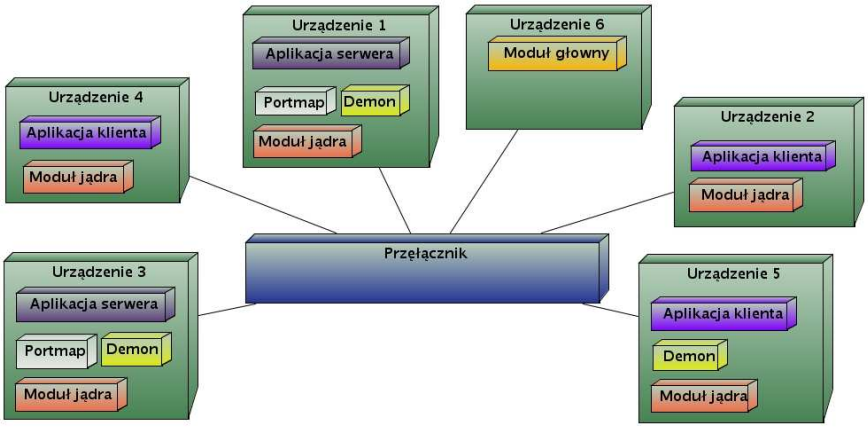
W przypadku stosowania metody przechwytywania komunikatów przekazywanych między klientami a serwerem poprzez zmianę numerów portów w pakietach sieciowych wykorzystywany jest dodatkowo ładowalny moduł jądra dla systemu *Linux*. Moduł zaimplementowano w języku *C*, wykorzystując szkielet operacji na pakietach sieciowych *Netfilter*, który jest częścią jądra systemu od wersji 2.4. W module jądra wykorzystano jedną z funkcji obsługi pakietów, tj. *NF_INET_PRE_ROUTING*, która w przypadku odebrania dowolnego pakietu przez interfejs sieciowy wywołuje samodzielnie zaimplementowaną funkcję *nf_hook_redirect*, odpowiedzialną za:

- sprawdzenie, czy odebrany pakiet sieciowy posiada nagłówek protokołu *Ethernet*, *IP*, *TCP* lub *UDP*;
- sprawdzenie dla każdego odebranego pakietu *TCP* lub *UDP*:
  - numeru portu docelowego – jeśli zgodny jest z numerem portu, który wykorzystywany jest przez serwer, zamieniany jest na numer portu używanego przez demona;

- numeru portu źródłowego – jeśli zgodny jest z numerem portu, który wykorzystywany jest przez demona, zamieniany jest na numer portu używanego przez serwer.

3.2. Konfiguracja środowiska *DEPENDITA* do wszczepiania błędów

W systemie wszczepiającym błędy (rys. 6) uruchomiona jest jedna instancja modułu głównego, natomiast demony pracują na urządzeniach, na których uruchomiony został serwer lub klient. Ponadto w przypadku zastosowania metody przechwytywania komunikatów przekazywanych między klientami a serwerem poprzez zmianę numerów portów w pakietach sieciowych konieczne jest uruchomienie ładowalnego modułu jądra systemu *Linux* na każdym z urządzeń, na których pracują klienci lub serwery testowanego systemu.



Rys. 6. Konfiguracja środowiska *DEPENDITA* do wszczepiania błędów

Komunikacja pomiędzy modułem głównym a demonami odbywa się przy wykorzystaniu interfejsu gniazd strumieniowych, za pomocą którego moduł główny wysyła definicje zadań do realizacji, a także polecenia konfiguracyjne (np. przerejestrowanie usługi w *portmap*). Z kolei demon wysyła do modułu głównego raporty dotyczące wykonanych zadań, a także przechwycone dane przekazywane między klientami a serwerem. Numery portów, które są wykorzystywane przez gniazda strumieniowe, definiowane są w modyfikowalnych plikach konfiguracyjnych.

3.3. Testy przykładowych aplikacji rozproszonych

Zbudowane narzędzie *DEPENDITA* przetestowano dla dwóch aplikacji rozproszonych, zaimplementowanych przy wykorzystaniu standardu *Sun RPC* – aplikacji *Echo* oraz *Studenci*. W pierwszej z nich serwer dla otrzymanych danych wykonuje określone obliczenia i dostarcza ich wyniki do klienta wysyłającego żądanie. Z kolei druga z aplikacji umożliwia zarządzanie listą studentów oraz ocen z przedmiotów, które przechowywane są w bazie danych.

Dla każdego z wymienionych systemów w środowisku testowym uruchomiono jedną instancję aplikacji serwera, natomiast na dwóch innych urządzeniach uruchomiono klientów. Na dodatkowej maszynie uruchomiono ponadto moduł główny, natomiast na urządzeniu, na którym pracuje serwer aplikacji, działał również demon wstrzykujący błędy.

Testy aplikacji rozproszonych zrealizowano w kilku etapach. Pierwszym krokiem było uruchomienie klientów oraz serwera badanego systemu, a także modułu głównego oraz demona. Następnie aktywowano demona i rozpoczęto wykonanie określonych operacji przez klientów. W trakcie ich realizacji demon pracujący na urządzeniu, na którym uruchomiono serwer, przechwytywał wszystkie dane wysyłane i odbierane przez serwer, a następnie dostarczał je do modułu głównego. Zebrane dane wejściowe oraz wyjściowe zdalnych procedur realizowanych przez serwer zostały wykorzystane w dalszej części prowadzonych testów, tj. w trakcie oceny wiarygodności badanej aplikacji. W kolejnym kroku zdefiniowano za pomocą modułu głównego (rys. 7) zadania wszczepiania błędów np.:

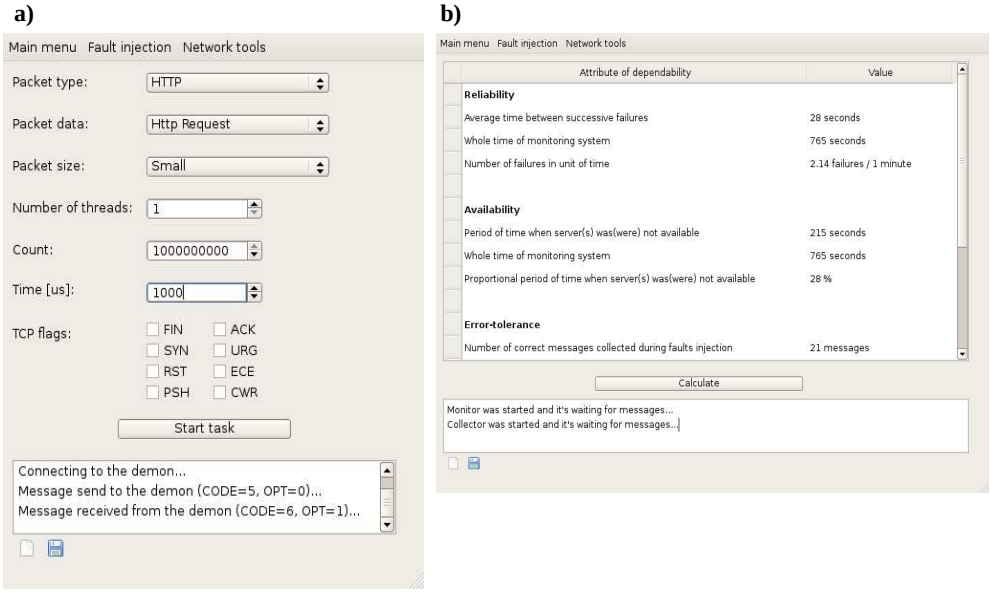
- Modyfikowanie danych liczbowych przesyłanych przez klienta (serwer wykonywał obliczenia na zmienionych wartościach, stąd dostarczone wyniki były niepoprawne).
- Blokowanie dwóch kolejnych wiadomości wysyłanych do serwera (dla protokołu TCP klient wyświetlał informacje o braku możliwości dostarczenia komunikatów, natomiast dla protokołu UDP klient w przypadku braku otrzymania odpowiedzi ponawiał próbę dostarczenia wiadomości co 5 sekund, dzięki czemu trzeci komunikat wysłany do serwera został poprawnie odebrany).
- Opóźnienie dostarczenia jednej wiadomości do serwera o 6 sekund (dla protokołu TCP serwer wykonał poprawnie operację obsługi żądania, natomiast dla protokołu UDP zrealizował ją dwukrotnie, bowiem po okresie 5 sekund klient ponownie wysłał żądanie).

Istotnym elementem realizacji tego etapu testów było wprowadzanie identycznych danych wejściowych i wybór takich samych rodzajów operacji realizowanych przez klientów jak w pierwszym etapie, w którym nie realizowano żadnego z zadań wszczepiania błędów. Powyższy warunek musiał

być spełniony, by w kolejnym kroku możliwe było porównanie przechwyconych żądań oraz odpowiedzi, a następnie wyznaczenie wartości współczynników charakteryzujących wybrane atrybuty wiarygodności np.:

- Dla atrybutu „niezawodność” wyznaczano średnią liczbę awarii systemu na jednostkę czasu.
- Dla atrybutu „dostępność” obliczano część całkowitego okresu czasu pracy systemu, w którym serwer nie był dostępny.
- Dla atrybutu „odporność na błędy” wyznaczano stopień tolerancji błędów.

a) **b)**



Rys. 7. Interfejs użytkownika modułu głównego systemu wszczepiającego błędy – okno kreowania nowego zadania wstrzykiwania (rysunek a) oraz wyniki wykonania algorytmu oceny wiarygodności badanej aplikacji (rysunek b)

Kolejnym etapem testów było wprowadzanie losowych uszkodzeń. Typy zadań, adresy atakowanych urządzeń, liczba realizowanych operacji wszczepiania błędów oraz ich rodzaje, a także czas rozpoczęcia realizacji zadań określane były przy wykorzystaniu rozkładów prawdopodobieństwa.

Ostatnim etapem testów było wykonanie próby doprowadzenia do załamania aplikacji serwera w wyniku wykonania ataku *DoS* czy też generowanie fikcyjnych żądań dostarczanych do serwera. Przykładowo serwer aplikacji *Studenci* przestał odpowiadać na żądania klientów w wyniku

uruchomienia zadania generowania pakietów *FTP* o rozmiarze 1430 bajtów, wysyłanych co 2000 µs przez trzy wątki.

4. Podsumowanie

Za pomocą zbudowanego systemu wspomagającego badanie wiarygodności aplikacji rozproszonych opartych na standardzie *Sun RPC* możliwe jest:

- wszczepianie błędów wpływających na przesyłane wiadomości pomiędzy poszczególnymi urządzeniami poprzez ich blokowanie, modyfikowanie lub dostarczanie z opóźnieniem;
- przeprowadzenie próby „załamania" serwera aplikacji poprzez wykonanie *ataku DoS* na wybrane urządzenie lub generowanie fikcyjnych żądań wysyłanych bezpośrednio do serwera;
- wpływanie na komunikację pomiędzy poszczególnymi elementami składowymi badanej aplikacji rozproszonej poprzez wprowadzanie zmian w tablicy routingu, reguł zapory sieciowej *iptables* lub tablicy *ARP*.

Niewątpliwą zaletą narzędzia *DEPENDITA* jest zastosowanie techniki przechwytywania wiadomości przesyłanych pomiędzy poszczególnymi elementami testowanej aplikacji, która nie wymaga wprowadzania żadnych zmian w kodzie źródłowym testowanego systemu czy też jego przebudowywania. Ponadto obie metody nie wpływają na sposób działania i komunikowania się innych aplikacji, które mogą być uruchomione na urządzeniach, na których pracują klienci lub serwer.

Niektóre funkcje systemu (np. generator pakietów) mogą być wykorzystywane w środowisku innym niż *Sun RPC*. Warto również zwrócić uwagę na funkcjonalność, która nie była dostępna w znanych systemach testowania wiarygodności, czyli możliwość oceny ilościowej wybranych atrybutów wiarygodności testowanej aplikacji.

Istotną cechą zbudowanego narzędzia, pomimo jego złożoności (liczba linii kodu źródłowego modułu głównego, demona oraz ładowalnego modułu jądra systemu wynosi w sumie ponad 30 000), jest heterogeniczność, tzn. sposób konstrukcji aplikacji pozwala rozbudowywać ją poprzez obsługę innych technologii tworzenia aplikacji rozproszonych czy też wstrzykiwanie nowych typów błędów.

Literatura

[1] AVIZIENIS A., LAPRIE J.C., RANDELL B., *Fundamental Concepts of Dependability*, http://www.cert.org/research/isw/isw2000/papers/56.pdf, 1-6 (kwiecień 2009).

[2] CRISTIAN F., *Understanding Fault-Tolerant Distributed Systems*, University of California, San Diego La Jolla, Maj 1993, 25-28, pp. 2-8.

[3] DAWSON S., JAHANIAN F., MITTON T., *ORCHESTRA: A Fault Injection Environment for Distributed Systems*, University of Michigan, pp. 1-30.

[4] DAWSON S., JAHANIAN F., MITTON T., *ORCHESTRA: A Probing and Fault Injection Environment for Testing Protocol Implementations*, University of Michigan, pp. 56.

[5] DHANJANI N., CLARKE J., *Bezpieczeństwo sieci. Narzędzia*, Wydawnictwo HELION, Gliwice 2006, str. 245-296.

[6] HOARAU W., TIXEUIL S., *FAIL – A language-driven tool for fault injection in distributed systems*, pp. 1-17.

[7] HSUEH MEI-CHEN, TSAI T.K., RAVISHANKAR K.I., *Fault injection techniques and tools*, Miesięcznik „Computer", Kwiecień 1997, pp. 75-82.

[8] LAPRIE J.-C., *Dependable computing and faut fault tolerance: concepts and terminology*, Avenue du Colonel Roche, Toulouse, pp. 2-9.

[9] ROSENBERG H.A., SHIN K.G., *Software fault injection and its Application in Distributed Systems*, University of Michigan, pp. 208-217.

[10] SUJECKA M., WISZNIEWSKI B., *Mechanizmy zapewnienia wiarygodności w standardzie CORBA*, Miesięcznik „Software" 2.0, nr 10/2001, str. 50-57.

[11] TANENBAUM A.S., MAARTEN VAN STEEN, *Systemy rozproszone – zagadnienia i paradygmaty*, Wydawnictwo Naukowo-Techniczne, Warszawa 2006, str. 71-88, 374-426.

[12] WHEELER K.B., MURPHY R.C., THAIN D., *Qthreads: An API for Programming with Millions of Lightweight Threads*, University of Notre Dame South Bend, Indiana, USA, pp. 1-8.

[13] *Multithreaded Programming Guide*, Sun Microsystems, rozdział *Basic Threads Programming*, http://dlc.sun.com/pdf/816-5137/816-5137.pdf, 28-53 (kwiecień 2009).

[14] *Netfilter FAQ (Frequently Asked Questions)*, Dokumentacja techniczna Netfilter, http://www.netfilter.org/documentation/index.html (kwiecień 2009).

[15] *Programming Interfaces Guide*, Sun Microsystems, *Interprocess Communication*, http://dlc.sun.com/pdf/817-4415/817-4415.pdf, 129-133 (kwiecień 2009).

[16] *Remote Procedure Call (RPC)*, *Struktura ramek dla protokołu RPC*, http://www.rhyshaden.com/rpc.htm (kwiecień 2009).

[17] *RPC: Remote Procedure Call – Protocol specification, version 2, Request For Comments 1057*, Sun Microsystems, http://www.ietf.org/rfc/rfc1057.txt (kwiecień 2009).

[18] *Qt Reference Documentation (Open Source Edition)*, Dokumentacja techniczna biblioteki Qt 4.1, http://doc.trolltech.com/4.1/index.html (kwiecień 2009).

[19] *Qwt – Qt Widgets for Technical Applications*, Specyfikacja klas biblioteki Qwt 5.1.1, http://qwt.sourceforge.net/index.html (kwiecień 2009).

[20] *XDR: External Data Representation Standard, Request For Comments 1832*, Sun Microsystems, http://www.ietf.org/rfc/rfc1832.txt (kwiecień 2009).

DEPENDITA – the tool for testing and evaluation of dependability of distributed applications

ABSTRACT: In the paper, the software tool called *DEPENDITA*, which was built at the Institute of Teleinformatics and Automatics, is presented. The tool may be used for dependability evaluation of distributed applications by software fault injection and it is dedicated for testing of distributed systems, which uses Sun RPC standard. The main advantage of DEPENDITA is the use of special techniques of messages capturing which do not require any changes in the source code.

KEYWORDS: dependability, fault injection, remote procedure call, distributed system

Praca naukowa finansowana ze środków przeznaczonych na naukę w latach 2008-2011 jako projekt badawczy nr O 514051135.

Praca wpłynęła do redakcji: 09.11.2009.