Test-Framework zur softwarebasierten Fehlerinjektion, -stimulation und Protokollierung von WSN-Anwendungen

Max Frohberg, Sebastian Reinhold, Paul Poppe, Mario Schölzel
IHP
Im Technologiepark 25
Frankfurt (Oder), Germany
Email: {frohberg, reinhold, poppe, schoelzel}@ihp-microelectronics.com

***Zusammenfassung*—Tests und Verifikation sind ein fundamentaler Bestandteil im Softwareentwicklungsprozess. Im Bereich der Drahtlosen Sensornetze ist ein intensives Testen sowohl der Anwendung als auch des Gesamtsystems unabdingbar. Aufgrund ihrer physischen Erreichbarkeit sind diese nur schwer zu debuggen. Insbesondere bei der Entwicklung von Kommunikationsprotokollen kann das Verhalten im Feld unter realen Bedingungen vom zuvor simulierten abweichen. In diesem Paper wird ein Framework zur softwarebasierten Fehlerinjektion im Bereich von WSN-Anwendungen auf Grundlage einer Cross-Plattform (CP) vorgestellt. Die CP unterstützt die Portierung einer Anwendung auf eine andere Plattform durch Nutzung unterschiedlicher Betriebssysteme. Nach der Portierung wird diese mittels Fehlerinjektion und Manipulation der Datenströme getestet. Ein zusätzlicher Mechanismus zum Protokollieren unterstützt zudem die Fehlersuche. In Kombination ist somit die Eingrenzung oder sogar die Rekonstruktion der Fehlerursache möglich. Das hier vorgestellte Framework wurde für Einzel- und Netzwerktests in einem WSN und insbesondere für die Entwicklung von MAC- und Routing-Protokollen konzipiert. Es eröffnet zudem die Möglichkeit, einfache Peripherie wie Sensoren oder Aktuatoren zu emulieren, falls diese auf der Entwicklungsplattform nicht zur Verfügung stehen.**

I. Einleitung

Fehler in der Softwareentwicklung haben zum Teil fatale Auswirkungen und verursachen umso mehr Kosten, je später sie gefunden werden [1]. Ein modelgetriebenes Vorgehen kann den Entwicklungsprozess dabei unterstützen, sodass Fehler möglichst vermieden werden können. So wird beispielsweise beim V-Modell jeder Phase im Entwicklungsprozess eine Testphase zur Qualitätssicherung zugeordnet und deren Ergebnis mit den jeweiligen Anforderungen abgeglichen. Je weiter der Entwicklungsprozess voranschreitet, umso konkreter wird die eigentliche Implementierung. Fehler die hier nicht oder zu spät entdeckt werden, haben in der Regel ein schwieriges und zeitaufwändiges Debuggen der Software zur Folge.

Bei der Entwicklung im Bereich von Drahtlosen Sensornetzen (WSN) kommen zudem noch weitere Herausforderungen hinzu. So kommen zum Beispiel Cross-Compiler zum Einsatz, da Software in der Regel auf einer anderen Plattform entwickelt wird als sie später eingesetzt werden soll. Einzelne Sensorknoten können zudem über eine große Fläche verteilt und nur schwer erreichbar sein. Auch hat die energieeffiziente Entwicklung von Kommunikationsprotokollen maßgeblichen Einfluss auf die Langlebigkeit einer Applikation, da nur begrenzte Energieressourcen zur Verfügung stehen [2]. Des Weiteren muss die Software auch Fehler in der Hardware oder Peripherie erkennen und behandeln und für einen zuverlässigen Betrieb daher hinreichend getestet werden.

Im Bereich der WSN ist es im Sinne der Einsatzdauer sinnvoll, auch den Stromverbrauch zu messen und schon während der Umsetzung konsequent zu beobachten und zu bewerten. Zudem stehen zahlreiche Simulationswerkzeuge wie *OMNet++*, *Cooja* unter *ContikiOS* oder *TOSSIM* unter *TinyOS* zur Verfügung, die je nach zugrundeliegendem Modell einfache oder sogar komplexe Systeme simulieren können. Eine weitere Möglichkeit sind Testbeds, die sich insbesondere während der Entwicklung eignen, um unter anderem Stresstests von besonders kritischen Funktionen durchzuführen, Sensorknoten im Netzwerk zu testen oder schnelle Effizienzanalysen unterschiedlicher Implementierungen durchzuführen.

Sowohl Testbeds als auch Simulationen haben jedoch einen entscheidenden Nachteil. Sie können das Verhalten einer Anwendung unter nicht deterministischen realen Bedingungen oder Umwelteinflüssen nur sehr begrenzt simulieren. Die Fehlersuche bei einem abschließenden Test unter realen Bedingungen gestaltet sich daher oft schwierig. Hier stehen nur wenig Informationen und Möglichkeiten zum Debuggen zur Verfügung und auch der Zugriff auf die Sensorknoten ist begrenzt.

Die in [3] vorgestellte Cross-Plattform (CP) unterstützt für solche Szenarien den Softwareentwicklungsprozess durch ein eingebettetes Test-Framework, dessen erster Entwurf in diesem Paper vorgestellt wird. Es ermöglicht Integrations- und Systemtests zur Laufzeit auf einer Zielplattform auszuführen, Peripherie oder Ereignisse zu emulieren, Fehler zu injizieren und Ergebnisse für die spätere Auswertung zu speichern.

II. Konzept

Änderungen durch ein Testsystem sollten so wenig wie möglich Einfluss auf die eigentliche Anwendung haben. Auch Anpassungen an den einzelnen Betriebssystemen (OS) sollten vermieden werden. Eine Anwendung auf Basis der CP greift

über OS-Wrapper auf OS-Funktionen zu. Sowohl Ereignisse die vom System erzeugt, als auch Funktionsaufrufe die von der Applikation ausgeführt werden, durchlaufen dabei die OS-Wrapper. Daher eignet sich diese Schicht um ein anwendungs- und plattformunabhängiges Testsystem zu implementieren. Grundlegende Funktionen wie das Speichern von Daten, virtualisierte Timer und ein integriertes drahtloses Code-Update stehen zur Verfügung.

A. Datenmanipulation

Die Architektur der CP wird genutzt um Aufrufe und Parameter in den OS-Wrappern bzw. Callback-Funktionen zu manipulieren. Dazu werden sogenannte Saboteure für einzelne Funktionen definiert und Funktionsparameter oder Rückgabewerte vor der eigentlichen Ausführung zur Laufzeit abgefangen. Diese beeinflussen gezielt das Verhalten vor der weiteren Verarbeitung oder ersetzten sogar ganze Funktionen. Die Ausführung eines Saboteurs kann durch die Nutzung von vordefinierten Templates mit Bedingungen oder Systemereignissen verknüpft werden. Somit lassen sich Hard- und Softwarefehler sowie Ereignisse gezielt erzeugen und zum Beispiel für Paketverlustsimulationen oder Lasttests unter realen Bedingungen nutzen.

B. Aufbau

Für Tests werden die Sensorknoten in einem drahtlosen Testnetzwerk organisiert. Ein weiterer Sensorknoten vermittelt als Gateway zwischen diesem Testnetzwerk und einem Hostsystem. Zum Kontrollieren und Steuern der Testszenarios werden die Code-Update-Funktionen und Kommunikationsprotokolle der CP genutzt.

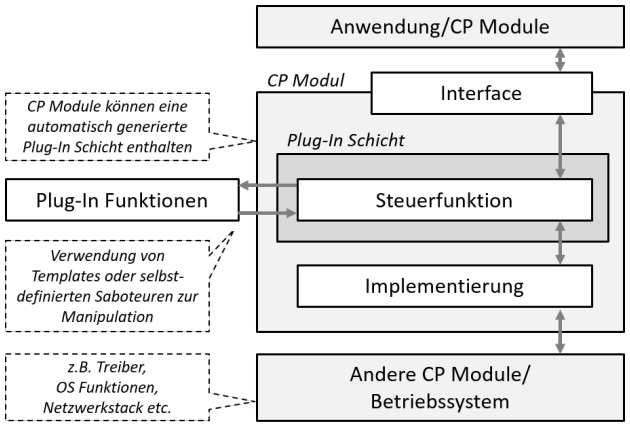

Abbildung 1. Pluginschicht in der Implementierung einer Funktion innerhalb eines Moduls in der CP zwischen OS-Wrapper und Anwendung

C. Testablauf

Das Test-Framework unterscheidet zwei Zustände, eine *Ruhephase* in der das System wie ursprünglich implementiert läuft und eine *Testphase* in der die Saboteure aktiv sind. Nach dem Systemstart befinden sich alle Sensorknoten solange in der Ruhephase, bis das Gateway den Test global startet. Ein Testdurchlauf wird durch ein zuvor definiertes Zeitintervall begrenzt und nach dessen Ablauf automatisch angehalten. Diese Daten werden in einem externen Speicher abgelegt und mit Zusatzinformationen wie Zeitstempel und Infocodes zum Verursacher versehen. Somit kann der Fehler oder zumindest die Stelle an der er aufgetreten ist identifiziert werden. Am Ende jedes Testzyklus werden die Daten zur Auswertung über das Gateway zurück zum Hostsystem gesendet.

Die Implementierung eines Tests wird, im weitesten Sinne analog zu Hooks aus der Softwareentwicklung, durch eine Pluginschicht realisiert. Diese kann zur Compile-Zeit aktiviert werden und steht dann für die zu testende Funktion zur Verfügung. Die Pluginschicht steuert zur Laufzeit den Programmfluss und wechselt je nach Zustand des Testsystems zwischen Originalfunktion und Pluginfunktion. In einer Pluginschicht können mehrere Pluginfunktionen implementiert werden, deren Ausführung durch eine Steuerfunktion kontrolliert wird. Die prinzipielle Manipulation eines solchen Aufrufs innerhalb eines CP-Moduls ist in Abbildung 1 dargestellt.

D. Emulation

Eine begrenzte Hardwareemulation wird durch die Nutzung der OS-Wrapper innerhalb der CP möglich, um zum Beispiel Sensoren oder Bus-Kommunikation zu simulieren, falls diese zwar auf der Ziel-, nicht jedoch auf der Entwicklungsplattform zur Verfügung stehen. Ein Saboteur ruft dafür eine API-Funktion in der CP auf und erzeugt somit ein System-Ereignis.

III. Ausblick

Der Testablauf ist in der aktuellen Umsetzung durch Pointer-Manipulationen realisiert. Diese könnte in Zukunft vereinfacht und durch Makros zur Compile-Zeit festgelegt werden. Ein erneutes Aufspielen der Test-Anwendung wird vermeidbar, wenn ausschließlich die Testparameter vor jedem neuen Testlauf konfiguriert werden. Zudem ist eine automatisierte Stapelverarbeitung von Testszenarien mit unterschiedlichen Parametern im gesamten Netzwerk angedacht.

Danksagung

Diese Arbeit ist Teil des DIAMANT-Projekts am IHP und wurde durch das *Bundesministerium für Bildung und Forschung* unter der Referenznummer *03IPT601X* gefördert.

Literatur

[1] K. A. Briski, P. Chitale, V. Hamilton, A. Pratt, B. Starr, J. Veroulis, and B. Villard, “Minimizing code defects to improve software quality and lower development costs.,” tech. rep., IBM Rational Software, October 2008.

[2] M. Ram and S. Kumar, “Analytical energy consumption model for mac protocols in wireless sensor networks,” in *2014 International Conference on Signal Processing and Integrated Networks (SPIN)*, IEEE, February 2014.

[3] M. Frohberg, P. Poppe, N. Vetter, and M. Schölzel, “Cross-plattform zur hardware- und betriebssystemunabhängigen implementierung von anwendungen und protokollen,” in *15. GI / ITG Fachgespräch Sensornetze*, pp. 47–48, Institut für Informatik und Computational Science, september 2016.