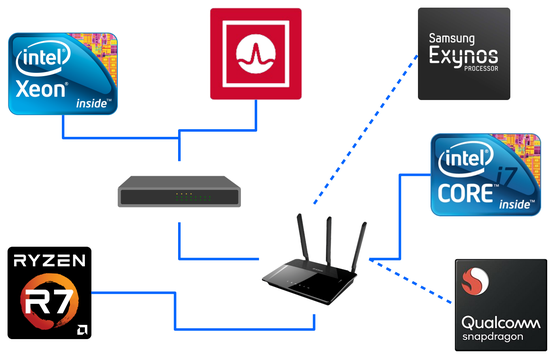
ThingsMigrate: Platform-Independent Migration of Stateful JavaScript Applications

Julien Gascon-Samson, Kumseok Jung, Shivanshu Goyal,
Armin Rezaiean-Asel, and Karthik Pattabiraman

Department of Electrical and Computer Engineering, University of British Columbia, Canada

Motivation

Modern IoT network



IoT devices are now **general-purpose**

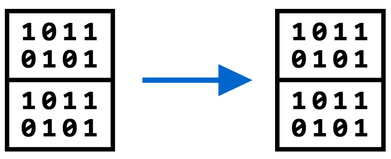
IoT devices are **resource-constrained**

IoT systems are **heterogeneous**

How to **migrate** a running process **without** considering **platform-specific details**?

Low-level Migration

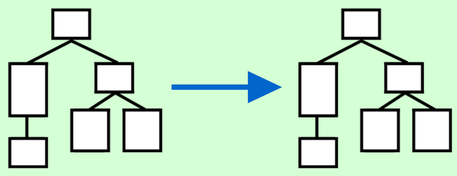
- Program counter
- Registers
- Memory Pages



Platform-dependent

High-level Migration

- Stack
- Variables
- Functions



Platform-independent

Challenges

We need:	**Entire program state**
We have:	**Hidden program state**

Closures have hidden state

We can update **balance** by invoking **bank**, but there is **no way to access balance** directly at the JavaScript layer.

The variable **balance** is hidden in a closure.

```
function PiggyBank(){
    var balance = 0
    var deposit = function(amount){
        balance += amount
    }
    return deposit
}
var bank = PiggyBank()
console.log(balance, bank.balance)
// undefined, undefined
```

```
var count = 0
setInterval(function(){
    count ++
}, 1000)
```

Event Queue is hidden

At any point in time, we have **no way of knowing:**

1. *what function is to be invoked* next
2. *how much time is left* until next invocation.

Approach

1. Instrumentation

Expose the hidden program state via code instrumentation to avoid modifying the JavaScript engine (e.g. Node.js).

2. Serialization

Given the entire program state, **serialize it into a transportable "snapshot"** such as a JSON string

3. Restoration

From the snapshot, reconstruct the program state by **generating new code** on-the-fly.

Implementation

Scope

We inject a ***Scope*** object at the beginning of each function.
A ***Scope*** object stores the current local state of the function.

```
1  var root = new Scope()
2  function PiggyBank(){
3      var scope_0 = new Scope(root)
4      var balance = 0
5      scope_0.vars.balance = balance
6      var deposit = function(amount){
7          var scope_0_0 = new Scope(scope_0)
8          balance += amount
9          scope_0.vars.balance = balance
10     }
11     scope_0.addFunction(deposit)
12     return deposit
13 }
14 root.addFunction(PiggyBank)
15 var bank = PiggyBank()
16 things.setInterval(
17     root.addFunction(function putMoney(){
18         var scope_1 = new Scope(root)
19         bank(100)
20     }), 1000)
```

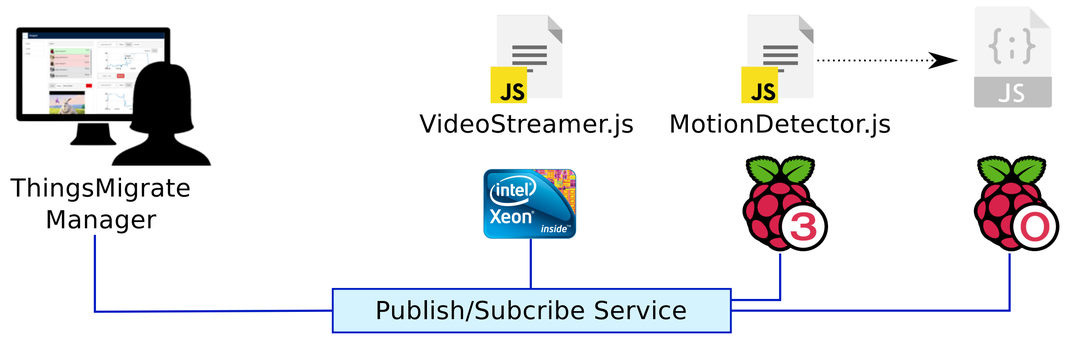
Augment Timers

We replace native timer calls with our own timers to track timer state

Updating Scope

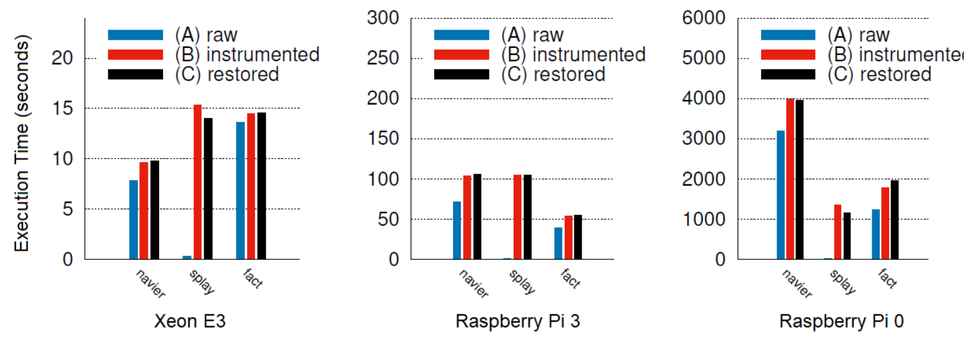
After each statement that updates a variable, we ***copy the new value*** into the Scope objects

Case Study: Motion Detector

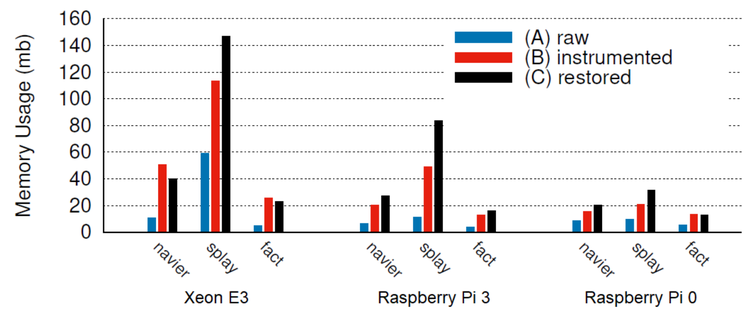


Results

Execution Time



Memory Usage



Summary

ThingsMigrate provides support for **migration of JavaScript programs**

- Stateful applications
- Platform-independent
- No VM modification

Find out more

thingsjs.juliengs.com

github.com/karthikp-ubc/ThingsJS