Center for Embedded Networked Sensing

EmStar-2: The Next Generation of Programming Development Environment for 32-bit Class of Embedded Devices

Vinayak Naik[1], Lewis Girod[2], Martin Lukac[1], Nithya Ramanathan[1], Ben Greenstein[1], Eddie Kohler[1], and Deborah Estrin[1]
[1]CENS–UCLA and [2]CSAIL–MIT
http://research.cens.ucla.edu

Introduction: EmStar facilitates WSN software development for 32-bit platform

Role of EmStar in WSN

- **A development platform for 32-bit class of embedded devices**
  - EmStar provides a programming development environment for building robust WSN systems for 32-bit class of embedded devices

Stargate

AENSbox

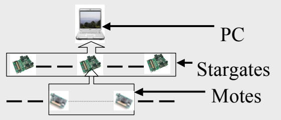
Nokia smartphone

- **A library of services commonly found in WSN systems**
  - EmStar provides a library of commonly found services in WSN, such as localization, time synchronization, reliable broadcast, etc.
- **Tools to provide visibility in large-scale networks of embedded devices**
  - EmStar provides monitoring tools to read and change status of process in networked environment via emproxy and echocat
- **A seamless transition between development, simulation, emulation, and deployment**
  - EmStar provides necessary compilation architecture and tools to enable use of the same code across various stages from development to deployment

Problem Description: Simplify software development and large-scale deployment

Unhandled issues in the current generation of EmStar

- **Increasingly involved role for 32-bit platforms in WSN unearthed new issues**
  - Traditionally, 32-bit platforms serve as micro-server or master in large-scale mote networks
  - Recently, 32-bit platforms are themselves uses as sensor nodes, e.g. MASE deployment in Mexico for seismic sensing



Stargate-based sensor node for seismic sensing

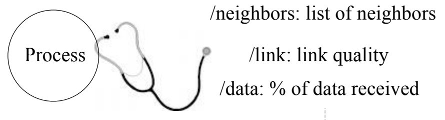
- **Multiple operating systems for 32-bit platforms**
  - All though, Linux is the most widely used operating system, Windows is used by certain devices such as smartphones
- **Incorrect memory access resulting in run-time failure**
  - Given the scarcity of memory, lack of garbage collection increases the chances run-time crash
- **Issues with multiple processes**
  - It is complex to write processes, which can deal with restart of other processes; e.g. dealing with restarted time-sync process
  - For large-scale deployment, multiple processes in each node makes system management complex

Proposed Solution: New design principles for EmStar-2

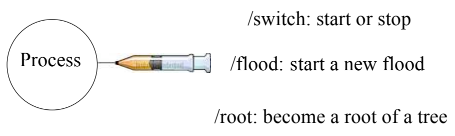
- **Python as a programming language**

	Garbage collection	Portability across OS	Handling of C structures	Code readability
C	✖	✓	✓	✓
C#	✓	✓ (Work in progress)	✓	✓
Java	✓	✓	✖	✓
Perl	✓	✓	✓	✖
Python	✓	✓	✓	✓

- **Unix domain sockets (UDS) to provide visibility into the system and control over the system**
  - UDS appear as files in file-system of node
  - UDS are portable across Linux and Windows
  - A process can expose its state via UDS



  - A process can receive commands over UDS



- **Asynchronous (non-blocking) process communication**
  - A single thread of execution can communicate with multiple processes, e.g. visibility and control tools
  - No issues related to locking, deadlock, or synchronization found in multi-threaded system
  - Use of GLIB tools for generating and handling events

```
s = socket.socket(socket.AF_UNIX, socket.SOCK_STREAM)
s.connect((FILENAME))
p = select.poll()
p.register(s.fileno(), select.POLLIN | select.POLLHUP)
while 1:
    results = p.poll(1)
        if len(results):
            if (results[0][1] == select.POLLIN):
                data = s.recv(8)
                if not len(data):
                    print("\rRemote end closing connection; exiting.")
                    break
                print "Received: " , data
            elif (results[0][1] == select.POLLHUP):
                print "Server hanged up; exiting."
                sys.exit(0)
        else:
                print "Problem occured; exiting."
                sys.exit(0)
```

- **Use of exception-handling to reduce run-time crashes**
  - In case of erroneous condition, program catches the error gracefully rather than crashing
  - A traceback of function calls is provided to user facilitate debugging
- **Ability to use devices generated by existing EmStar code**
  - EmStar-2 code can use all the device files generated by existing EmStar code, hence leveraging on large code-base of existing tools and utilities